IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 25, 2020, accepted October 3, 2020, date of publication October 12, 2020, date of current version October 23, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3030537

Resilient Service Chains through Smart Replication

DAVID PEREZ ABREU, KARIMA VELASQUEZ, LUÍS PAQUETE, MARILIA CURADO, AND EDMUNDO MONTEIRO, (Senior Member, IEEE)
Centre for Informatics and Systems, Department of Informatics Engineering, University of Coimbra, 3030-290 Coimbra, Portugal

Corresponding author: David Perez Abreu (dabreu@dei.uc.pt)

The work of David Perez Abreu and Karima Velasquez was supported by the Portuguese funding institution FCT - Foundation for Science and Technology under the Ph.D. grants SFRH/BD/117538/2016 and SFRH/BD/119392/2016, respectively. This work was supported in part by the European Regional Development Fund (FEDER), through the Regional Operational Programme of Lisbon (POR LISBOA 2020); in part by the Competitiveness and Internationalization Operational Programme (COMPETE 2020) of the Portugal 2020 framework [Project 5G with Nr. 024539 (POCI-01-0247-FEDER-024539)]; in part by the national funds through the FCT - Foundation for Science and Technology, I.P., within the scope of the project CISUC - UID/CEC/00326/2020; and in part by the European Social Fund, through the Regional Operational Program Centro 2020. The work presented in this paper was partially carried out in the scope of the project SNOB-5G: Scalable Network Backhauling for 5G (reference CENTRO-01-0247-FEDER- 045929) leading to this work is co-financed by the ERDF - European Regional Development Fund through the Operational Program for Competitiveness and Internationalization - COMPETE 2020, the Center Portugal Regional Operational Program - CENTRO 2020 and the Portuguese Foundation for Science and Technology - FCT under the MIT Portugal Program.

ABSTRACT The Internet of Things paradigm enables a new set of smart end-user applications. The Cloud-Fog-Mist-Internet of Things infrastructure provides communication, compute, and storage support for these applications. However, this complex, heterogeneous, and distributed landscape requires orchestration and management mechanisms in order to guarantee their proper functioning. One particular factor to manage is the capacity to provide service resilience even in the presence of failures in components of the substrate infrastructure. This research proposes a set of mechanisms to formalize, orchestrate, and embed a batch of service requests for chained Virtual Functions to fulfill the specific requirements of applications while enhancing their availability and ultimately their resilience. In detail, this work introduces a formal grammar to describe customized Service Chains, allowing the definition of replicas for different Virtual Functions, and an Integer Linear Programming model for Virtual Function embedding that prioritizes the use of nodes with higher availability. Additionally, an alternative heuristic is presented to handle more complex scenarios by taking advantage of the multi-tier scenario comprising the Cloud-Fog-Mist-Internet of Things. Simulation results for the embedding mechanisms show that it is possible to increase the resilience of chained Virtual Functions, while balancing the load of the infrastructure nodes.

INDEX TERMS Cloud, fog, IoT, embedding, resilience, service chain.

I. INTRODUCTION

The Cloud computing paradigm adoption by network operators and service providers has been massive, given its benefits in cost-savings, enhancement in work and management response, business agility, and Quality of Service (QoS) [1]. Despite the initial success of Cloud adoption, in recent years, there has been a tendency shift to bring computational resources and services towards the edge of the network to fulfill the requirements of emerging paradigms such as the Internet of Things (IoT). In this scenario, there is a larger scale of heterogeneous devices and lower latency that could represent a significant challenge for the traditional Cloud environments [2].

The Fog emerges as a solution to improve Cloud-based services by offering a distributed and federated compute model to decentralize the deployment, management, and orchestration of services and applications across the entire network infrastructure. Thus, the Fog computing paradigm lays on a tiered model that enables ubiquitous-access scalable computing resources. This model aids the placement of context-aware services and applications in computational nodes, which are set between smart end-devices and centralized Cloud systems. As soon as the Fog computing paradigm was adopted, the use of geographically dispersed, low-latency computational resources increased the need for more specialized and dedicated nodes closer to the end-users;

The associate editor coordinating the review of this manuscript and approving it for publication was Kashif Sharif.

thus, the concept of Mist nodes emerged. The Mist nodes are deployed in the Mist computing layer which resides in the peripheral of the network infrastructure, even closer to the end-users [3].

The Cloud-Fog-Mist-IoT service infrastructure can be represented as a set of hardware and software components organized in tiers (i.e., from the Cloud in the top to the IoT in the bottom going through the Fog and Mist tiers) that enables the deployment and interconnection of smart end-user applications and devices empowering low-latency and context awareness data processing in a distributed way. The capabilities, service types, and deployment models available in the Cloud-Fog-Mist-IoT have been influenced by virtualization techniques usually known as the softwarization of the Cloud to IoT continuum [4]. Particularly, the network softwarization [5] allows the design, development, test, management, and deployment of services and applications via Network Functions (NFs) using the available resources (i.e., hardware or software components) in the infrastructure to route the network flows through the right components [6]. This approach could be extrapolated along all the tiers in the Cloud to IoT continuum in order to provide services and applications to end-users via a chain of Virtual Functions (VFs). Some examples of the softwarization approach on this domain are Network Function Virtualization (NFV) [7], Software-Defined Networking (SDN) [8], and Service Function Chaining (SFC) [9].

Services, such as video streaming, online gaming, mobile connectivity, and IoT applications, are composed of different software and hardware components usually hosted on top of a network infrastructure organized according to the Cloud to IoT continuum approach. Consequently, the network operator is responsible for the embedding, management, and orchestration of a set of services that can be instantiated and used by various service providers and their final users, frequently applying softwarization techniques via VFs with the objective of fulfilling service and application requirements [6], [9], [10]. These VFs can be focused on different domains, such as network infrastructures or computational activities. The VFs are grouped in structures known as Service Chains (SCs), which combine the specific functionality requirements needed to fulfill more complex service and application requirements.

The landscape, where the SCs are deployed, lies on an extremely diverse substrate infrastructure[1] composed of information and communication devices (i.e., Cloud nodes, Fog nodes, Sensors, Actuators) and links (i.e., wired and wireless channels), which have to be orchestrated to host a plethora of services and applications with different performance as well as functional requirements. In such a complex scenario, resilience becomes a key factor to orchestrate and guarantee the continuity of the services and applications even in the face of failures. One possibility to increase the resilience level of the services is to use replicas, such that, in the event of a node failure, the replica can be activated and the service can maintain its availability [11].

End-to-End (E2E) services and applications in the Cloud to IoT continuum can be described by a VF Forwarding Graph that links the endpoints through a set of interconnected VFs. Accordingly, the reliability and availability of the E2E services/applications are based on the behavior of their functional blocks (i.e., VFs and their communication links) [12]. Therefore, when replicas are considered as a resilience mechanism for Service Chains, it is possible to apply two methods: 1) the replication is applied to the entire chain, or 2) the replication is applied to one or more VFs along the chain. Besides the method used during the replication phase, it is also essential to consider where to place the replicas (e.g., deploy the primary and backup Virtual Function in different nodes to avoid such that in the case of a failure in said node, the replica can be activated) and how to formally represent the Service Chain requests considering the possibility of having replicas.

This work presents an embedding framework for Service Chains that includes the following contributions:

1) A formal grammar to verify the correctness of Service Chains from Virtual Functions available via a catalog. The VFs can be listed in a particular order or not, and the number of replicas desired for each VF can be specified;
2) A formal mechanism for embedding VFs in the substrate network infrastructure based on a bi-level Integer Linear Programming (ILP) model, aimed at increasing the acceptance rate while maximizing the resilience of the service by embedding the VFs in the nodes with the highest availability factor;
3) A heuristic based on Fluid Communities for embedding the VFs in the substrate network infrastructure, that is less time and resource consuming than the formal model, especially when handling larger scenarios; and
4) An evaluation of the proposed mechanisms using simulation.

The proposed framework is designed for single service provider scenarios, but could be adapted to a federated environment with some modifications to fulfill the specific requirements of that context. The paper is structured as follows. Section II presents a review of the related work. Section III describes a grammar that allows customized and formal Service Chain definition. A bi-level ILP model for Service Chain embedding and handling of the replicas is introduced in Section IV. A heuristic based on fluid communities is presented in Section V. The evaluation setup is described in Section VI and the experimental results are discussed in Section VII. Finally, conclusions are presented in Section VIII.

II. RELATED WORK

There has been some work previously done in VFs embedding; more specifically, concerning how to embed VFs requested in the substrate communication topology, with the

[1]The physical nodes and links that are virtualized

objective of increasing the resilience of the Service Chains. This section presents some works on embedding in general before introducing works focused on the area of resilient embedding.

Chowdary *et al.* [13] propose two heuristics to increase the acceptance ratio and revenue for virtual network embedding, while decreasing cost for the substrate network. Resilience is not considered in this work.

Beck and Botero [14] tackle the NFV resource allocation problem by dividing it into two phases: service chain composition and service chain embedding. Their algorithm is not lead by any metric of the links or nodes in the substrate network, thus finding the first available set of nodes in the substrate network to embed the service chain without any leverage in the selection process.

Three evolutionary algorithms for service embedding in the Fog are proposed by Guerrero *et al.* [15], and three objectives were drawn: minimizing latency; minimizing free resources; and optimizing service spread (even distribution of services). These proposals suffer from scalability and modularity problems, like any other basic genetic algorithm [16].

Bays *et al.* [17] use knowledge of the substrate network to create a virtual infrastructure abstraction allowing the representation of Virtual Network requirements, while also proposing an embedding model ensuring physical feasibility. The solution is presented as an ILP problem, which as a stand-alone solution could not be well suited for complex scenarios.

Mehraghdam *et al.* [18] present a model to specify network function chaining requests before introducing an embedding solution. The resilient component of the NFs, as well as, the tiered approach that drives the Cloud to IoT continuum are not considered in the model.

So far, the works analyzed focus on VFs embedding, without considering the availability of the Service Chains. The following works are aimed at increasing the resilience of the Service Chains.

Khan *et al.* [19] tackle the resilience VF embedding problem. They propose a multi-path link embedding mechanism to maximize resiliency while minimizing resource wastage with replicas. This solution relays on the existence of disjoint paths in the substrate network in order to guarantee its success.

Rahman and Boutaba [20] also analyze the survivable virtual function embedding problem. They propose a hybrid heuristic, as well as a baseline heuristic. The proposal is based on single substrate link failures and does not deal with node failures.

Proactive and reactive approaches are also studied by Souza *et al.* [21], who model them as the well-known multidimensional knapsack problem. This solution assigns resources to each request, but even though there are computational resources available to recover from the failure, there is still the need to deploy the proper NF instance to said resources to reactivate the service, particularly when using the reactive approach.

Aidi *et al.* [22] propose the use of replicas to increase the resilience of service chains. They also present two heuristics for more complex scenarios. They aim to maximize the number of replicas and spread them as much as possible. However, they only consider a single node failure.

Lera *et al.* [23] introduce an embedding policy aimed at increasing the service availability and the QoS. They use communities to divide the network infrastructure. The graph partition technique applied to create the communities among the Fog nodes is based on the work of Newman and Girvan [24]. This community creation mechanism could result in un-even communities, having small communities with saturated nodes, and monster communities with under-used nodes.

From the reviewed literature, some observations arise. First, simulation seems to be the main method for evaluation [13], [15], [19], [21]–[23], which is due to the complex scenario comprised by the Cloud to IoT continuum. Mathematical solutions are also broadly used, following ILP or MILP models [13], [17], [19], [22]; even though these models perform well in offline studies, they suffer from time constraints when applied to online scenarios. Although the embedding problem has been previously addressed, SCs resiliency has been mostly overlooked [13]–[15], [17], [18]. The works considering resiliency can be grouped into link failures [19], [20], and node failures via resource allocation [21] or replication [22], [23].

Unlike the proposals analyzed in this section, this work offers an embedding framework comprised by: 1) a method to formalize customized Service Chain requests, taking into consideration the communication between the comprising Virtual Functions; 2) a formal mechanism for Virtual Function embedding aimed at maximizing the availability; and 3) a heuristic for Virtual Function embedding to increase the resilience in more complex scenarios. Both the formal mechanism, based on ILP, and the heuristic, take into consideration information about the substrate network (i.e., the availability factor of the nodes, and the tiered infrastructure of the Cloud to IoT continuum) for the embedding process.

III. FORMALIZING VIRTUAL FUNCTION CHAINS

For delivering services and applications in the Cloud to IoT continuum, a set of VFs and infrastructure components have to be instantiated or activated so the corresponding communication flows can be routed between the endpoints. Two challenges arise when the overall process of deploying these services and applications is appraised: 1) how to standardize and formalize a request of a service chain considering the communication and dependencies between the VFs?; and 2) how to embed the service chain and its components efficiently into the substrate communication infrastructure?

The embedding process considered in this research is described in Figure 1. The service provider offers a set of

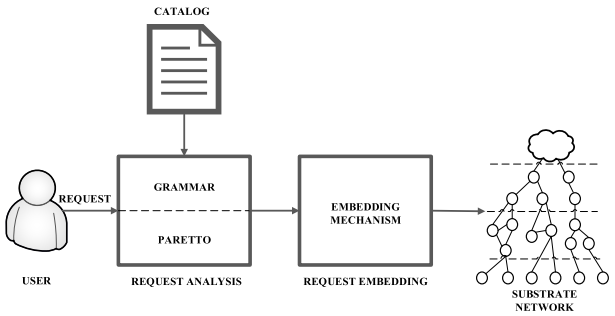


FIGURE 1. Embedding framework.

VFs via a catalog, which is maintained by an instance of the Management and Network Orchestrator (MANO) [6], from where users can select them. Users form their requests by grouping different VFs, forming an SC according to a set of rules defined by a grammar (module *Request Analysis*).

The grammar verifies that the VFs are organized correctly for each SC. In case that there is only one possible resulting SC from the set of VFs selected by the user (i.e., the VFs in the SC were declared as a fixed order), the resulting SC goes to the *Request Embedding* module. If there are different possible SCs generated by a given set of VFs, a Pareto analysis is used to select the SC that fulfills the user's requirements, minimizing resource consumption (i.e., CPU, memory, and bandwidth). The SC selected by the Pareto analysis goes to the next module in the framework. The SC resulting from the *Request Analysis* module can be represented by the grammar, by a simple Directed Acyclic Graph (DAG), by a.xml/.json file, or by any other method defined by the service provider. In the following module (*Request Embedding*) the embedding mechanism selects the nodes inside the substrate network to embed each of the VFs from the SC. In this work, two different mechanisms are proposed in Section IV and Section V respectively. Other embedding mechanisms with different optimization goals can be instantiated in this module. The final step is the actual embedding of the VFs in the substrate network, comprised of the layered structure of the Cloud-Fog-Mist-IoT, according to the directions from the *Request Embedding* module.

The rest of this section describes the solution designed to tackle the first challenge previously mentioned, particularly, a context-free grammar to deal with the representation of the Service Chain requested, enabling the possibility to specify replicas for the VFs. This corresponds to the first module of the framework (*Request Analysis*). The resulting Service Chains will be used as an input for the embedding mechanisms presented in the following sections, which deal with the second challenge and comprise the second module of the framework (*Request Embedding*).

In the Cloud to IoT continuum considered for this research, the substrate infrastructure where the VFs chains will be instantiated and embedded is modeled as a graph denoted by $G = (N, L)$, where N and L are the sets of nodes and links of the communication infrastructure, respectively. Each physical node $n \in N$ has a computing capacity Ω_n (i.e., CPU, memory, storage); similarly, each physical link $\ell \in L$ has a transmission capacity Γ_l (i.e., propagation rate, bandwidth). The set of Virtual Functions V that could be instantiated in the infrastructure is accessed via a catalog, maintained by an instance of the MANO, made available by the infrastructure provider. Detailed information about each virtual function $v \in V$, such as their resource requirements ω, is also accessible via the catalog. Service chains are deployed in the infrastructure as a sequence of VFs, where a service s denotes a chain of Virtual Functions $vf_1, vf_2, \ldots, vf_{|V|}$ connected via a set of virtual links E. The order of the VFs that compose the service chain could be fixed or variable according to a given criterion that represents the interaction between the components; for example, in an IoT application that requires sensing, analytics, and storage services, the order of the last two components could swap depending on the application domain or user.

An infrastructure provider receives requests to deploy services, which are modeled as a graph $S = (V, E)$, where V denotes the set of VFs, and E represents the set of edges that connect the VFs. These requests specify particular requirements that must be fulfilled, such as the number of replicas of a particular VF to increase its resilience and the order of a set of VFs inside a given chaining request. Additionally, it is considered that a VF could favor a specific tier (i.e., Cloud, Fog, Mist, IoT) for its embedding.

Besides of the assumptions already presented, some additional considerations were taken into account to model the Cloud to IoT environment: 1) all the nodes in the communication infrastructure have the capacity to execute network, computational, storage, or even sensing functions (i.e., each Virtual Function can be embedded in any node); 2) functions and services are used interchangeably, considering that the required functionalities to fulfill the desired actions can be virtualized and executed across all the network infrastructure; 3) the amount of resources, as well as the availability per node, decreases from the nodes in the Cloud to those in the IoT in the hierarchical tiered infrastructure, taking into account the constraints present in each tier; and 4) the Cloud tier has infinite resources and its availability is not affected by failures.

A context-free language was designed in order to have a standard and formal method to represent and validate application/user's chaining requests. With this representation, it is possible to build customized complex requests composed by a set of ordered/un-ordered VFs to provide services. Each chain request is formed by a different kind of *modules*. Various of these modules can be placed in a chain request to denote a particular set of requirements for a given service. To process/recognize the Virtual Function - Chain Composition (VF-CC) requests, the production rules of the former mentioned grammar, in Backus–Naur Form (BNF), are listed from (1) to (12).

The terminals of the grammar, plus the empty set, are given in bold font. The grammar enables the definition

of VF service chains. The chains can have a fixed order (i.e., *fix_order*) defined by the user or not, giving the service providers the possibility to rearrange the VFs to their convenience (i.e., *opt_order*). This grammar also enables the use of *replicas* for each VF in the chain, specifying the number of copies desired. Furthermore, there is the possibility to request the deployment of the VFs in a particular *tier* of the infrastructure: *cloud*, *fog*, *mist*, or *iot*.

The proposed grammar allows a flexible description of the service chains by enabling the possibility to define strict order or not of the VFs inside the chain. Furthermore, the user can specify the number of replicas for each VF in the chain and the tier in the network infrastructure where the VF should be deployed. This creates the possibility to define tailored Service Chains according to the needs of the user. It is important to notice that it is not restrictive to use the grammar to define the service chains; it is possible to use another method for the definition, such as a DAG, and only using the grammar to check the correctness of the service chain. This would offer the service provider more freedom to select the method that is better suited for the description of their service chains. The grammar described deals with the first challenge introduced at the beginning of this section, specifically about how to standardize and formalize a service chain request with support of replication for the VFs. In the next section, the embedding challenge of the requested service chain is discussed.

$$\langle \text{start} \rangle \models \textit{service}\{\langle \text{chain} \rangle\} \tag{1}$$

$$\langle \text{chain} \rangle \models \langle \text{order} \rangle\ \langle \text{chain} \rangle \mid \langle \text{modules} \rangle\ \langle \text{chain} \rangle \mid \langle \text{order} \rangle \mid \langle \text{modules} \rangle \tag{2}$$

$$\langle \text{order} \rangle \models \textit{fix\_order}\{\langle \text{modules} \rangle\} \tag{3}$$

$$\langle \text{modules} \rangle \models \langle \text{optorder} \rangle \mid \langle \text{replica} \rangle \mid \langle \text{term} \rangle \tag{4}$$

$$\langle \text{optorder} \rangle \models \textit{opt\_order}\{\langle \text{term} \rangle\ \langle \text{moreterm} \rangle\} \tag{5}$$

$$\langle \text{replica} \rangle \models \textit{replica}\{\langle \text{chain} \rangle : \langle \text{num} \rangle\} \tag{6}$$

$$\langle \text{moreterm} \rangle \models , \langle \text{term} \rangle\ \langle \text{moreterm} \rangle \mid \epsilon \tag{7}$$

$$\langle \text{moremod} \rangle \models , \langle \text{chain} \rangle\ \langle \text{moremod} \rangle \mid \epsilon \tag{8}$$

$$\langle \text{term} \rangle \models \langle \text{vf} \rangle\ \langle \text{tier} \rangle \tag{9}$$

$$\langle \text{tier} \rangle \models \textbf{cloud} \mid \textbf{fog} \mid \textbf{mist} \mid \textbf{iot} \mid \epsilon \tag{10}$$

$$\langle \text{vf} \rangle \models \boldsymbol{vf_1} \mid \boldsymbol{vf_2} \mid \ldots \mid \boldsymbol{vf}_{|v|} \tag{11}$$

$$\langle \text{num} \rangle \models \mathbf{1} \mid \mathbf{2} \mid \mathbf{3} \mid \ldots \mid \boldsymbol{n} \tag{12}$$

IV. MANAGING REPLICAS IN VIRTUAL FUNCTION CHAINS USING AN EXACT SOLUTION

The service chain embedding process requested via the context-free grammar described previously is detailed in this section. The main requirement considered for the embedding method of the VFs in the substrate network is the resilience of the Service Chains components (i.e., VFs) via the replication of all or part of them in disjoint physical nodes of the substrate infrastructure using an optimization approach. A bi-level formulation of the optimization problem is considered. On the first level, the goal is to maximize the acceptance rate of the Service Chains; on the second level, the aim is to maximize the availability of the Service Chains by placing the VFs in the most reliable nodes.

TABLE 1. Parameters and variables for the ILP model.

Parameters	
Parameter	**Description**
V	Set of virtual functions
S	Set of service chains requested composed by virtual functions $v \in V$
R	Set of instances of virtual functions (i.e., replicas) for a single service chain
N	Set of nodes in the physical topology
Ω	Capacity set. Ω_n is the resource capacity for $n \in N$
F	Availability set. F_n is the availability for $n \in N$
I	Instance matrix of size $\|S\| \times \|V\|$. I_s, v indicates the amount of instances of each virtual function $v \in V$ in a service chain $s \in S$
ω	Requirement set. ω_v is the resource requirement for the virtual function $v \in V$
T	Tier set. T_n indicates the tier in the infrastructure to which node $n \in N$ belongs to
Variables	
Variable	**Description**
A	Acceptance vector. $A[s]$ equals 1 if $s \in S$ is accepted.
P	Placement matrix. An $\|R\| \times \|V\| \times \|S\| \times \|N\|$ matrix

Table 1 introduces the parameters and variables used in the optimization solution. Service chains $s \in S$ are composed by virtual functions $v \in V$. Each virtual function can be replicated up to R times. The virtual function instances are to be deployed in nodes $n \in N$ belonging to the physical topology. For the feasibility restrictions, the abstraction of *resource unit* is applied, where a resource unit reflects the resources (i.e., CPU, memory, storage) of node $n \in N$, depicted in Ω; or the resource requirements (i.e., CPU, memory, storage) of virtual function $v \in V$ listed in ω.

A. MAXIMIZING ACCEPTANCE RATE

Equation (13) depicts the goal of the first optimization level, which is maximizing the acceptance ratio. Vector A holds the variables indicating the accepted service chains $s \in S$ (i.e., $A[s]$ equals 1 if $s \in S$ is accepted). In case of shortage of resources, it would lead to the prioritization of the chains that request a lower amount of resource units; thus, it penalizes the requests that include an excessive amount of replicas for their virtual functions.

$$\max \sum_{s \in S} A_s \tag{13}$$

The following constraint guarantees that only the replicas requested are placed.

$$\sum_{r \in R} \sum_{n \in N} P_{i,n}^{s,v} \leq I_{s,v} \quad \forall s \in S,\ \forall v \in V \tag{14}$$

The next constraint limits the placement of the replicas in different nodes. In case of a failure in a node, the replica is not affected and can be activated.

$$\sum_{n \in N} P_{i,n}^{s,v} \leq 1 \quad \forall s \in S,\ \forall v \in V,\ \forall r \in R \tag{15}$$

Equations (16) and (17) are interdependent and ensure that only the service chains placed are accepted for embedding. $M \in \mathbb{N}$ is a constant with a large value usually called "big M".

$$\sum_{n \in N} P_{r,n}^{s,v} \leq M \times A_s \quad \forall s \in S,\ \forall v \in V,\ \forall r \in R \tag{16}$$

$$A_s \leq \sum_{v \in V} \sum_{r \in R} \sum_{n \in N} P_{r,n}^{s,v} \quad \forall s \in S \tag{17}$$

Equation (18) guarantees that all the replicas of the virtual functions $v \in V$ that belong to the accepted service chain $s \in S$ are placed.

$$A_s \times I_{s,f} = \sum_{r \in R} \sum_{n \in N} P_{r,n}^{s,v} \quad \forall s \in S\ \forall v \in V \tag{18}$$

Equation (19) enforces the feasibility constraints. It only allows the placement of virtual functions $v \in V$ when node $n \in N$ has enough resources to host them.

$$\sum_{s \in S} \sum_{v \in V} \sum_{r \in R} P_{r,n}^{s,v} \times \omega_s \leq \Omega_n \quad \forall n \in N \tag{19}$$

B. MAXIMIZING AVAILABILITY

The second level of the optimization problem aims to maximize the availability of the Service Chains, and thus their resilience. The idea is to recover from failures by activating replicas of affected VFs which should be deployed in disjoint nodes with higher availability. F_n holds the availability indicator of each node $n \in N$ in the physical topology. This factor is periodically updated via orchestration mechanisms executed by the MANO, that collects and maintains information regarding the availability of the infrastructure nodes. This information is used to calculate the most recent availability values and update the value of F_n periodically with the current state of the infrastructure.

The availability is computed as the ratio between the uptime of the nodes and the aggregate of the expected values of uptime and downtime. Equation (20) shows the calculation for the availability.

$$Availability = \frac{E[uptime]}{E[uptime] + E[downtime]} \tag{20}$$

Equation (21) shows the formulation for the objective function of this level of the optimization problem. T_n indicates the tier in the network infrastructure to which node $n \in N$ belongs to; there are four possible tiers and their associated values go from lower to higher as the tiers go closer to the user. The tiers and their respective values are: Cloud (0.6), Fog (0.8), Mist (0.9), and IoT (1). These values represent a penalty associated to the propagation rate between the nodes according to the tier where they belong, based on previous work [25], [26]. Thus, this objective function tries to balance the embedding in nodes with the highest availability factor and also those nodes that are closer to the user, hence taking advantage of the entire network infrastructure by performing a vertical search (i.e., between all the tiers) instead of focusing only in the Fog nodes.

$$\max \sum_{s \in S} \sum_{v \in V} \sum_{r \in R} \sum_{n \in N} P_{r,n}^{s,v} \times F_n \times T_n \tag{21}$$

Equation (22) is added to ensure that the results from the first optimization problems are kept. This means, the service chains $s \in S$ that were accepted in the first step are fixed, and the selection of the nodes $n \in N$ can be changed in order to satisfy the second optimization goal.

$$A_s \times I_{s,v} = \sum_{r \in R} \sum_{n \in N} P_{s,n}^{r,v} \quad \forall s \in S,\ \forall v \in V \tag{22}$$

In order to keep the results from the first level, the acceptance vector resulting from the first optimization level is fixed and used as a parameter for the second optimization level. Constraints depicted from (15) to (19) are kept at this stage to guarantee feasibility.

V. AN HEURISTIC BASED ON FLUID COMMUNITIES BY TIERS

The bi-level formulation presented for Service Chain embedding is more suited for offline scenarios taking into consideration the time required to get a solution in complex online scenarios. Additionally, the optimal node(s) in the topology to embed VFs end up being overloaded. Thus, an alternative heuristic based on a graph and partition analysis is presented in this section.

Graph theory is frequently used to tackle problems in a vast range of engineering and computer science applications. A graph usually allows representing almost any physical situation involving discrete objects and a relationship between them (e.g., a communication network infrastructure) [27]. A relevant feature of performing graph representation is the partition or community structure that allows the study of edges and vertices that belong to a cluster with common interests and/or metrics [28].

An approach that has proven to be useful to deal with load balancing is building communities in the communication network topology [23], [29], [30]. Specifically, Lera *et al.* [23] proposed a service placement policy focused on enhancing the availability and QoS of applications using graph partition. The community detection used for the placement mechanisms was the method proposed by Newman and Girvan [24], which progressively removes edges from the original graph to identify the communities on it. The method removes the most valuable edge, usually the edge with the highest betweenness centrality, at each step. Thus, the graph breaks down into pieces, and the tightly knit community structure is exposed. In this research, we adopted a different approach to build more reliable and balanced communities using the Fluid Community method.

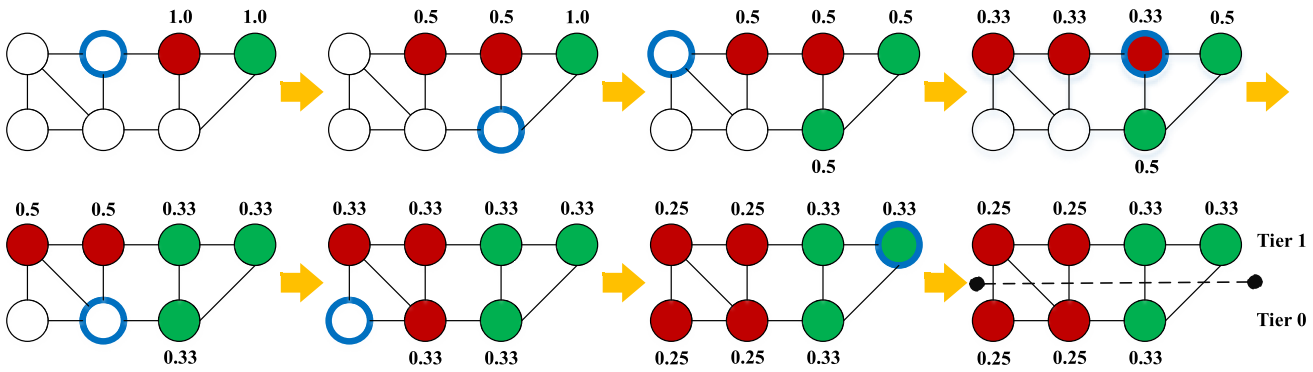


FIGURE 2. Workflow of Communities for $K = 2$ communities and *tiers* $= 2$ (Adapted from [31]).

A. BUILDING FLUID COMMUNITIES

The Fluid Communities (FluidC) algorithm is a Community Detection (CD) algorithm that could be applied to any graph to create groups called *communities* as sets of vertices densely interconnected that at the same time are sparsely connected with the rest of the graph [31]. Particularly, the FluidC algorithm creates the communities by mimicking the behavior of several fluids, expanding and pushing one another in the graph until an equilibrium is found. The algorithm receives an input parameter K, which indicates the number of fluids, i.e., communities, that are going to be created. Given a graph $G = (N, L)$ with N the set of vertices and L the set of edges, the algorithm will initialize K fluid communities $C = \{c_1, \ldots, c_K\}$ with $0 < K \leq |N|$. Each community $c \in C$ is initialized with a distinct and randomly selected vertex in N. The density $d(c)$ of a community $c \in C$ is defined according to (23), as the inverse of the number of nodes in the community c.

$$d(c) = \frac{1}{|c|} \tag{23}$$

For a vertex n, the update rule returns the community or communities with maximum aggregated density d within the ego network [2] of n. This rule is formally defined by (24) and (25), where n is the vertex being updated, C'_n is the set of candidate communities, each of which could be the new community for n, $B(n)$ are the neighbours of n, $d(c)$ is the density of community c, $c(z)$ is the community to which vertex z belongs to, and $\delta(c(z), c)$ is the Kronecker delta.

$$C'_n = \arg\max_{c \in C} \sum_{z \in \{n\} \cup B(n)} d(c) \times \delta(c(z), c) \tag{24}$$

$$\delta(c(z), c) = \begin{cases} 0 & \text{if } c(z) \neq c \\ 1 & \text{otherwise} \end{cases} \tag{25}$$

Figure 2, adapted from Parés *et al.* [31], describes the process of building the communities. Assume it selects the red node and the green node shown in the topology in Figure 2. After the first iteration, the density of both communities is 1, the maximum possible value for density. The node highlighted in blue represents the vertex where the update rule will be evaluated. For the following iterations, the algorithm will randomly select a node from the neighbours of the communities, which will be assigned to the community with the highest density value. In the case that there are several communities with the same density value, it will randomly select one of those communities. For the example depicted on Figure 2 the FluidC converges after one complete superstep .[3] The last stage depicted in Figure 2 shows the final two communities in a tiered environment with two tiers. The tiered communities approach is used for the embedding mechanism described in the next subsection.

The FluidC algorithm allows the definition of the desired amount of communities and also that there will not be a *monster* community ,[4] in comparison with the rest of communities in the graph. An adaptation from the original Fluid Communities (FluidC) algorithm is used in this work to partition the topology graph. Algorithm 1 describes the process to build the communities.

Algorithm 1 Build Communities

Result: k communities for a given network infrastructure

```
1 topology ← get_topology()
2 remove_node(topology, Cloud_node)
3 k ← highest_modularity(topology)
4 for i = 1 to k do
5 |   communities[i] ← FluidC(topology,k)
6 end
7 fcommunities ← highest_performance(communities)
8 return fcommunities
```

The algorithm begins by collecting the network topology information, in line 1, to organize the nodes by communities.

[2]Ego networks consist of a main node ("ego") and the nodes directly connected to it.

[3]An iteration over all the vertices of the graph.

[4]A community significantly larger than the rest.

The Cloud node is an independent community, handled differently considering the model assumption described in Section III (i.e., infinite resources and absence of failures), which is why it is not taken into consideration for this process (line 2).

The highest modularity value for the topology given is used to specify the value of k (see line 3), required as an input value for the FluidC algorithm, following the approach used by Pares *et al.* [31]. The modularity is a metric that measures the strength of the division of a graph, often used in optimization to detect community structure in a network. As the modularity grows, the groups inside the topology have denser connections between the nodes inside the groups, and sparser with the nodes in other groups, enabling a resilient communication between the nodes inside a community or group [32].

Considering that the FluidC algorithm evaluates the update rule using a random approach, the results of invoking this algorithm are not deterministic. Thus, the algorithm is invoked k times (lines 4 and 6) before choosing the final community set that will be used for the embedding process. The resulting communities set will be the one with the highest performance, in line 7, where the performance measures the ratio of the number of intra-community edges plus inter-community non-edges with the total number of potential edges [28]. Line 8 returns the final set of communities.

The communities created are balanced in the sense that they all have a similar amount of nodes (i.e., the standard deviation of the average community size is smaller than other community-building processes), which is a desired characteristic for an embedding process that takes into account resilience and load balancing. The characteristics of the communities created are discussed in Section VI.

B. EMBEDDING SERVICE CHAINS IN THE FLUID COMMUNITIES

The *Service Chains* have to be embedded in the network infrastructure considering the set of communities selected after invoking Algorithm 1. This corresponds to the second challenge described in Section III. Each *sc* will be embedded in a single community, respecting the resource constraints of the nodes and avoiding placing replicas of the same *vf* in a single node (except in the case of embedding in the Cloud node). The nodes in the community are also sorted according to the tier to which they belong (i.e., Fog, Mist, IoT); the Cloud node has its own community (see the last stage depicted on Figure 2).

The embedding process, named Fluid Communities by Tiers (FCT), and depicted in Algorithm 2, begins by getting the information about the topology and its tiered hierarchical structure, as seen in lines 1 and 2, respectively. This information is provided via a catalog of the network updated continuously by the MANO module of the Cloud to IoT infrastructure manager. With this information, the communities are built by invoking Algorithm 1 in line 3. Line 4 obtains the values that correspond to the *SCs* and their constituent *VFs*, such as response time, resource consumption, hardware and software specific requirements. The tiered communities *tcomm*, initialized in line 5, contains the list of nodes that belong to the communities sorted by tiers.

The actual embedding process takes place between lines 6 and line 34, for each *sc* requested and their *VFs*. The *sc* variable refers to a single Service Chain inside the set *SCs*. The first step of the embedding is to initialize the community pointer, *commp*, denoting the community where the *vf* is to be embedded (line 7). For each *vf*, a flag variable (*vf_deployed*) controls if the *vf* was successfully embedded in a substrate node of a given community. The tiered communities are sorted by their free computational resources, from highest to lowest, in line 10. The tiered communities ordered, *tselect*,

Algorithm 2 Fluid Communities by Tiers - FCT

Result: Embedding of SCs and their VFs

```
1  topology ← get_topology()
2  ttiers ← get_tiers(topology)
3  communities ← build_communities()
4  SCs, VFs ← get_service_chains()
5  tcomm ← sort_communities(communities,ttiers)
6  foreach sc in SCs do
7      commp ← commp  mod size(communities)
8      foreach vf in VFs do
9          vf_deployed ← False
10         tselect ← highest_resource(tcomm,commp)
11         if get_resource(vf) ≤ max(tselect[0]) then
12             if embedded(vf,tselect[0]) then
13                 vf_deployed ← True
14             end
15         end
16         if vf_deployed = False then
17             if get_resource(vf) ≤ max(tselect[1]) then
18                 if embedded(vf,tselect[1]) then
19                     vf_deployed ← True
20                 end
21             end
22         end
23         if vf_deployed = False then
24             if get_resource(vf) ≤ max(tselect[2]) then
25                 if embedded(vf,tselect[2]) then
26                     vf_deployed ← True
27                 end
28             end
29         end
30         if vf_deployed = False then
31             embedded(vf,Cloud)
32         end
33     end
34 end
35 return final_embedding
```

is an array of three elements that represents the nodes inside the community that belong to the Fog, Mist, and IoT.

From line 11 to line 15, the algorithm verifies if the given *vf* can be hosted by the node with highest available resources in the previously selected tier (line 11). If the embedding process is successful (line 12), which means that the *vf* instance was deployed in a disjoint node from its replicas, the flag variable *vf_deployed* is updated to *True* in line 13. If the embedding process was not successful, two new attempts are launched for the following two tiers with more available resources, as seen in the blocks from line 16 to 22 and from line 23 to 29. If all previous attempts fail, the embedding will take place in the Cloud, as depicted in lines 30 and 31. The result from the embedding process is returned in line 35.

In summary, the embedding process deploys a *vf* and their replicas in disjoint nodes, prioritizing the tiers with more available resources in order to balance the load between the Fog, Mist, and IoT, and just considering the Cloud if all previous attempts failed. Thus, the mechanism also promotes the embedding of the VF in nodes with high availability but also closer to the end-users in order to enhance the response time of services and applications. These algorithms are the core of the proposed embedding mechanism that is validated in the next sections.

VI. EVALUATION

The proposed embedding mechanisms were validated using YAFS simulator [33] given its strong support for Fog critical features and its capacity to introduce failures during the simulation [34], which allows evaluating the availability of the Service Chains. The experiments were conducted on a PC with 32GB 2400MHz DDR4 RAM and 2.80GHz Intel Core i7-7700HQ with 4 cores and 8 threads (2 threads per core) processor. The PC was running Microsoft Windows 10 Pro (Build 18363) operating system. The IBM CPLEX Optimizer version 12.9 [35] was used for the ILP model, and Python 2.7.16 was used for YAFS.

For this work, two partition graph methods (i.e., Fluid Communities and Newman-Girvan) were initially considered. The main idea was selecting the graph partition method that provides the best results regarding load balancing and resilience for the communities building process. Both partition methods were evaluated over different synthetic random network topologies (i.e., Barabasi-Albert, Complete, Lobster, PowerLaw, Star, and Tree), each with 50 nodes. The validation process consisted of a batch of experiments to compare metrics including performance [5] (higher values are better), communities length, average and standard deviation of the communities size; the number of communities for both methods was 5. The target was building communities similar in size so there were no tiny or monster communities as an outcome.

[5]Ratio of the number of intra-community edges plus inter-community non-edges with the total number of potential edges.

TABLE 2. Graph partition evaluation.

Method	Topology	Perf	Comm. Size	Mean	SD
FluidC	Tree	0.83	[7,13,7,13,10]	10.00	2.68
	Barabasi	0.82	[10,10,10,10,10]	10.00	0.00
	Lobster	0.83	[7,8,12,14,9]	10.00	2.60
	PowerLaw	0.82	[11,10,11,10,8]	10.00	1.09
	Star	0.18	[1,1,47,1,1]	10.20	18.40
	Complete	0.19	[9,8,11,14,8]	10.00	2.28
Girvan	Tree	0.83	[11,11,9,13,6]	10.00	2.36
	Barabasi	0.78	[16,12,7,13,2]	10.00	4.93
	Lobster	0.83	[12,13,6,7,12]	10.00	2.89
	PowerLaw	0.79	[19,6,7,11,7]	10.00	4.81
	Star	0.18	[47,1,1,1,1]	10.20	18.39
	Complete	0.84	[46,1,1,1,1]	10.00	18.00

The results from this previous evaluation step are listed in Table 2 and a visual representation is displayed in Figure 3. Although the values are similar, FluidC shows a lower standard deviation for the communities length. It is also noticeable from the boxplots in Figure 3 that overall FluidC creates communities more balanced regarding their size. These observations led to the selection of FluidC for creating the communities. The final topology selection was the Barabasi-Albert since it has been used before in similar studies [23], [29].

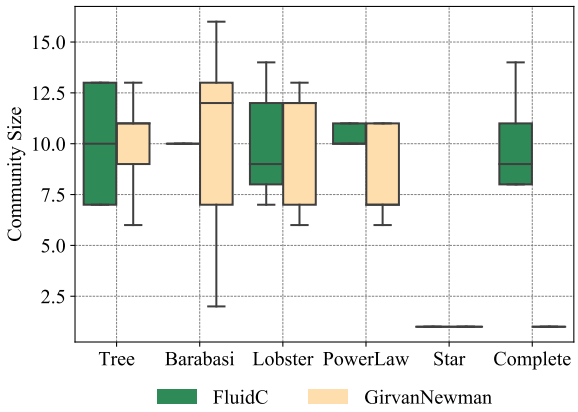


FIGURE 3. Comparison of the number of nodes per community for the FluidC and Girvan-Newman methods for different communication topologies.

After this analysis, a graph to model the substrate communication network was generated following a random Barabasi-Albert method, according to the complex network theory [36]. 49 nodes are spread between the IoT, Mist, and Fog, and an additional node (for a total of 50 nodes) represents the Cloud. The Cloud node is the node connected to the Fog nodes with the highest betweenness centrality in the graph; on the other hand, the nodes with the lower betweenness centrality correspond to the gateways in the IoT. The nodes directly connected to the gateways belong to the Mist, and the rest of the nodes constitute the Fog. In respect of the failures, the simulation time was set to 50000 time units, and there is a failure set for each 2500 time unit. Thus, there

are 20 failures during the simulation. This represents 40% of the nodes in the topology.

The availability factor for each node is assigned randomly according to some ranges that correspond to the layer to which the node belongs to. Thus, nodes closer to the edge have a lower availability factor than nodes closer to the core. This reflects the fact that nodes in the IoT have a higher probability of failure while the nodes in the Cloud are less prone to failure, and an availability uptime of 99.999% of the times is required, following the *five nines* principle [37].

Simulation parameters are listed in Table 3. The tier weight parameter guides the selection of the nodes in the embedding process so that the search space is explored vertically (see Tier set in Section IV). YAFS' resource unit is used to specify VFs demands. This unit is defined as a vector that contains the capacity of different computational resources to be used by the VFs (e.g., number of cores for CPU, GB for memory, or TB for the hard disk). Regarding the node resources, the amount is randomly selected between 10 and 25 resource units for the Fog, Mist, and IoT tiers; however, this value is weighted so that the IoT nodes are closer to 10 resource units and the Fog nodes are closer to 25 resource units.

TABLE 3. Simulation parameters.

	Parameter	Value
Cloud	Tier weight	0.6
	Availability (%)	99.999
	Resources (units)	∞
Fog	Tier weight	0.8
	Availability (%)	88 - 98
	Resources (units)	10 - 25
Mist	Tier weight	0.9
	Availability (%)	77 -87
	Resources (units)	10 - 25
IoT	Tier weight	1
	Availability (%)	66 - 76
	Resources (units)	10 - 25
VFs	Requirements (units)	1 - 5
	Execution (instr/req)	20000 - 60000
	Message size (bytes)	1500000 - 4500000
Failures	Time between failures (time units)	2500

For the Service Chains, five different examples were used based on typical Smart Cities and IoT services. These chains are: 1) Web services, 2) Video streaming, 3) Voice-over-IP, 4) Online gaming, and 5) Generic IoT application. These chains can be used as the foundation of more complex Smart City-based applications, such as smart surveillance (which could be based on video streaming) or smart lighting (that can be based on a generic IoT for sensing and actuating chain). The Service Chains used, and their belonging Virtual Functions, are listed in Table 4. Similar base service chains were used in other works [38]–[40].

Three different redundancy models are used for the Service Chains [12]: **End2End**, one replica for each VF in the chain; **vsReplicas**, some of the VFs in the chain have replicas; and **NoReplicas**, where no replica is used for any VF in the chain.

Four scenarios were defined by varying the network load with regards of the Service Chains: 1) **tiny**: 5 SCs, 2) **small**: 10 SCs, 3) **medium**: 15 SCs, and 4) **large**: 20 SCs. Similar loads were used in experimental cases before [29]. For comparison purposes, the ILP model and the FCT heuristic are validated against the well-known First-Fit (FF), as it is done in other works [29], [30], [41]. All the scenario setup, as well as the source code, is available via a GitLab repository [42].

TABLE 4. Service Chains.

Service Chain	Chained VFs
Web services	NAT-FW-TM-WOC-IDPS
Video streaming	NAT-FW-TM-VOC-IDPS
Voice-over-IP	NAT-FW-TM-FW-NAT
Online gaming	NAT-FW-VOC-WOC-IDPS
Generic IoT application	GSEN-DAGG-DACOM-DANA-DB

NAT: Network Address Translation; FW: Firewall;
TM: Traffic Monitoring; WOC: WAN Optimization Controller;
IDPS: Intrusion Detection Prevention System;
VOC: Video Optimization Controller; GSEN: Generic Sensor;
DAGG:Data Aggregator; DACOM: Data Compresion;
DANA: Data Analytics; DB: Data Base.

VII. RESULTS AND ANALYSIS

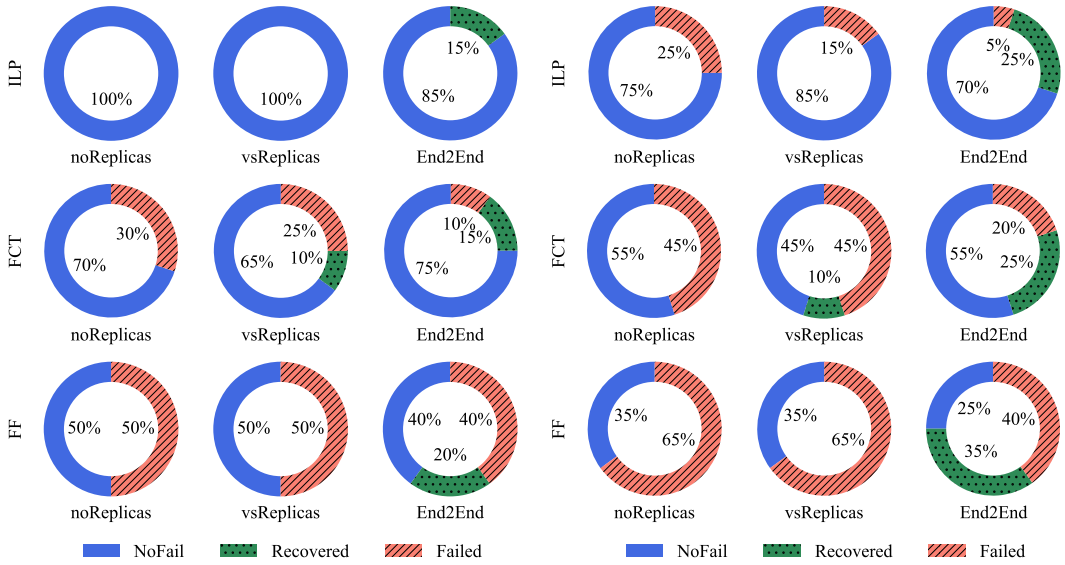
This section presents the results obtained from the simulations.The failures in the nodes were randomly generated before running the simulations to maintain the same scenario between repeated simulations. Correspondingly, the placement is statically executed before the simulations; hence the VFs are placed in the same nodes during the different simulations. This way, the reports presented in this section are the average of 30 repeated simulations where the conditions are kept the same to minimize any statistical error. This section shows a subset of the results obtained, given that the trend is the same for the scenarios, and the discussion can be extrapolated to any of them. All the data and additional plots are available for download via the GitLab repository [42].

The three replication strategies used follow the models of redundancy discussed in Section VI: 1) noReplicas, where no replicas are used for all the SCs; 2) vsReplicas, where some of the VFs inside the SC have replicas (the replication ratio is two VF replicas per chain); and 3) End2End, where all the VFs in the SC have replicas.

An important factor to take into consideration is the additional load added to the infrastructure with the different replication strategies (i.e., noReplicas, vsReplicas, End2End). As more replicas are added, the resources of the nodes are depleted sooner, thus forcing the embedding of the VFs in different nodes (i.e., more nodes in the topology are used), even though the embedding mechanism remains the same. The fact that the embedding mechanisms try to find disjoint nodes for different instances of a given VF magnifies this behavior.

A. FAILURE RATIO

The first experiment is aimed at evaluating the resilience of the Service Chains embedded in the network infrastructure.



(a) Medium scenario

(b) Large scenario

FIGURE 4. Service Chains failure ratio.

Three outcomes are possible: NoFail (no VFs from that SCs are affected by the node failures), Recovered (some VFs are affected but their replicas could be activated), and Failed (one or more VFs are affected but there is no replica or the replica(s) is also stricken by the failures). Figure 4 shows the results regarding the failure ratio, where the rows represent the embedding mechanisms and the columns the replication methods. The failure ratio refers to a node failure that affected a Service Chain, preventing a successful communication among its VFs; because a VF was embedded in the failing node, or because the failing node was a critical part of the communication path. Figure 4a depicts the results for the medium scenario, while Figure 4b does the same for the large scenario. Results for all the scenarios are listed in Table 5. As explained before, the load changes as more replicas are added to the infrastructure, influencing the embedding process.

As expected, the noReplicas method, for all mechanisms, could not recover from failures since there were no replicas to activate. It is also noticeable the trend that ILP showed the lowest number of failures, followed by FCT and then FF. For the vsReplicas, FF was not able to recover from failures in both scenarios. For the large scenario, the ILP mechanism was also unable to recover from failures, but the number of failing SCs was significantly lower. FCT was able to recover from some failures in both scenarios. Particularly, for the large scenario (see Figure 4b) the amount of failing SCs is lower for vsReplicas than noReplicas. This is due to some replicas taking place in nodes with higher availability factor, but that ultimately failed, and being placed in nodes with less availability that turned out not failing during the simulation.

With End2End, since there are replicas for all the VF, all the mechanisms were able to recover some SCs affected by the failures. For ILP in the medium scenario, although all the SCs were able to complete (i.e., recovered after node failures), it is noticeable that for the End2End method, 15% of the SCs were affected by failures. This is caused by the extra load included by the replicas. Using this method, more load is added to the nodes, forcing the embedding of some VFs in different nodes that were affected by failures. Nonetheless, the failing SCs were able to activate the replicas for the corresponding VFs. Using End2End, the amount of SC affected by failures is more significant for all the mechanisms, given the higher number of VF instances (i.e., primary and backup ones) that are embedded.

From Figure 4, ILP was always the most effective mechanism regarding the failures, having a higher success rate for all the scenarios. In the case of FCT, it is noticeable that there are fewer SC affected when using the End2End method. For this case, it is important to remember that FCT bases its embedding decision on the available resources by tier. Thus, by changing the load on the network, the embedding decision process is affected, thus changing the nodes selected for the embedding. In any case, FCT showed a behavior close to ILP, with only an additional of around 10% of failing SCs, while being superior to FF which had 40% of SCs affected by failures, when using the End2End method.

For all the methods and scenarios, FF was the mechanism that showed more failures and less capacity to recover; ILP was the mechanism with better results. It is noteworthy that as the load grows the number of failing Service Chains increase; and the load also affects the embedding process, thus the same VF instance can be embedded in different nodes even though the same mechanism is used.

B. NODE UTILIZATION

The next experiment had the objective of evaluating the load balancing of the different mechanisms. Figure 5 shows

TABLE 5. Results - SC Failures (in percentage %).

Replication	Scenario	Status	Mechanism		
			ILP	FCT	FF
NoReplicas	Tiny	NoFail	100	95	75
		Recovered	0	0	0
		Failed	0	5	25
	Small	NoFail	100	85	60
		Recovered	0	0	0
		Failed	0	15	40
	Medium	NoFail	100	70	50
		Recovered	0	0	0
		Failed	0	30	50
	Large	NoFail	75	55	35
		Recovered	0	0	0
		Failed	25	45	65
vsReplicas	Tiny	NoFail	100	95	75
		Recovered	0	5	0
		Failed	0	0	25
	Small	NoFail	100	80	65
		Recovered	0	10	0
		Failed	0	10	35
	Medium	NoFail	100	65	50
		Recovered	0	10	0
		Failed	0	25	50
	Large	NoFail	85	45	35
		Recovered	0	10	0
		Failed	15	45	65
End2End	Tiny	NoFail	100	100	80
		Recovered	0	0	10
		Failed	0	0	10
	Small	NoFail	95	95	60
		Recovered	5	5	15
		Failed	0	0	25
	Medium	NoFail	85	75	40
		Recovered	15	15	20
		Failed	0	10	40
	Large	NoFail	70	55	25
		Recovered	25	25	35
		Failed	5	20	40

the results for the noReplicas and End2End methods. For each replication method, the top plot shows the number of nodes used, while the bottom plot depicts the amount of VFs embedded on the busiest node; this is, the node with more VFs. The trend observed in these two figures is also present for the vsReplicas method.

Since ILP seeks the optimal node, it will saturate it, thus using fewer nodes overall, in spite of the replication method. As the scenarios grow, the number of nodes used also grows, showing an increasing trend; and maintaining the fact that ILP uses the lowest number of nodes between all the mechanisms. However, for ILP the results regarding the VFs on the busiest nodes are the highest among all the mechanisms, suggesting a saturation on the nodes. This might lead to more failures, both in the node as in the communication links.

With regards to the nodes with the highest load, there are different trends among the mechanisms. In the case of FF, all the nodes are treated equally (i.e., no node is prioritized). The VFs are embedded in any case when there are enough resources for it. On average, for these simulation parameters, 8 VFs will fill a node; thus, this is the value observed for the busiest node in all the scenarios. On the contrary, ILP and FCT prioritize the nodes according to their availability factor and the tier to where they belong to. Furthermore, the acceptance rate is also considered in the embedding process. Thus, smaller VFs are prioritized, favoring them during the embedding, resulting in smaller VFs grouped in the same node, with the additional advantage of lower internal fragmentation in the nodes. This behaviour leads to an increasing trend for ILP, with more VFs in the busiest node.

FCT had a hybrid behavior by selecting a subset of candidates (i.e., communities) and balancing the load amongst them. Hence, it shows a stable amount of VFs on the busiest node in all the scenarios, getting to balance the load, but also using more nodes than ILP to achieve this. This also reinforces the hypothesis of using a mechanism for load balancing during the embedding process, for instance, using communities; particularly using communities similar in size, while also considering the intra-community and inter-community connectivity.

C. RESPONSE TIME

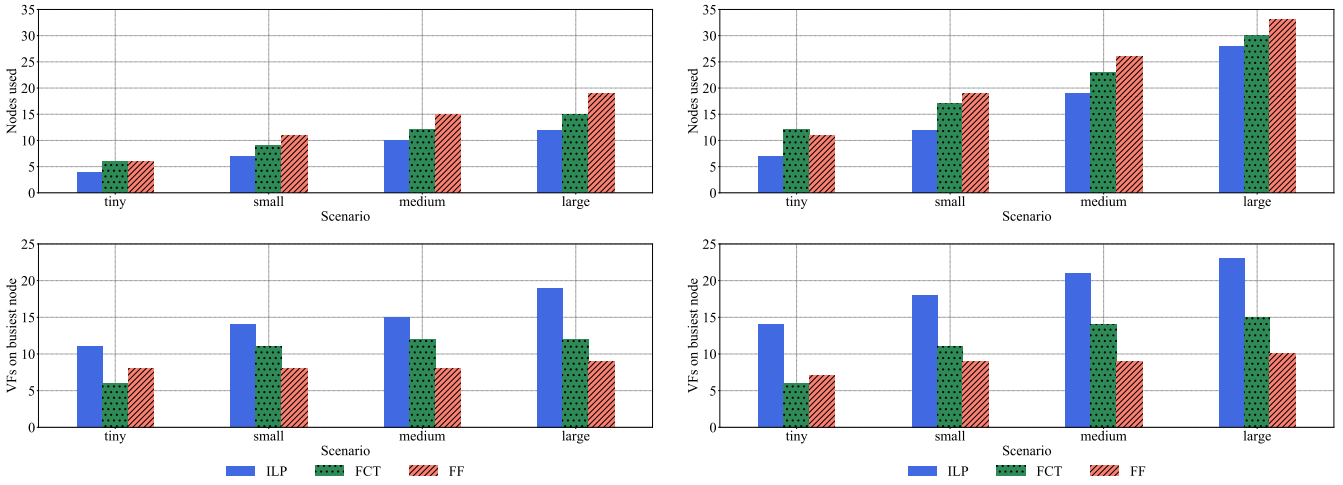
The following experiment was designed to evaluate the performance of the SCs from an end-user perspective. Figure 6 shows the results for this experiment for the medium scenario. Figure 6a depicts the results for the vsReplicas method, and Figure 6b for the End2End method. The plots show the average response time (in milliseconds) from successful end-to-end communications. The lack of a bar indicates that there was a node failure during the simulation that affected the corresponding Service Chain, preventing any successful end-to-end communication (i.e., no data was collected). The line on the top of the plot indicates the deadline for each Service Chain (also in milliseconds).

The only mechanism that failed to fulfill a complete successful end-to-end communication for at least one SC was FF. This is due to FF suffering from more failures than ILP and FCT (node failures that prevented a successful communication, as seen in Figure 4); thus, there is a higher probability that the failure that affects the SCs occurs at the beginning of its workflow. Particularly, in Figure 6a is noticeable that 3 SCs were not able to complete any end-to-end communication (i.e., SCs 2, 3, 5), while in Figure 6b 2 SCs were affected (i.e., SCs 3, 10).

On average, all the chains were able to complete their end-to-end communications within their deadline. In most cases, FCT showed the best response times. FCT distributes the VFs among the different tiers of the landscape, and by bringing some VFs closer to the user, the response time is improved. On the other hand, since ILP seeks the node with the highest availability, and these nodes are on the highest tiers of the infrastructure, the VFs are embedded farther away from the user, impacting the response time.

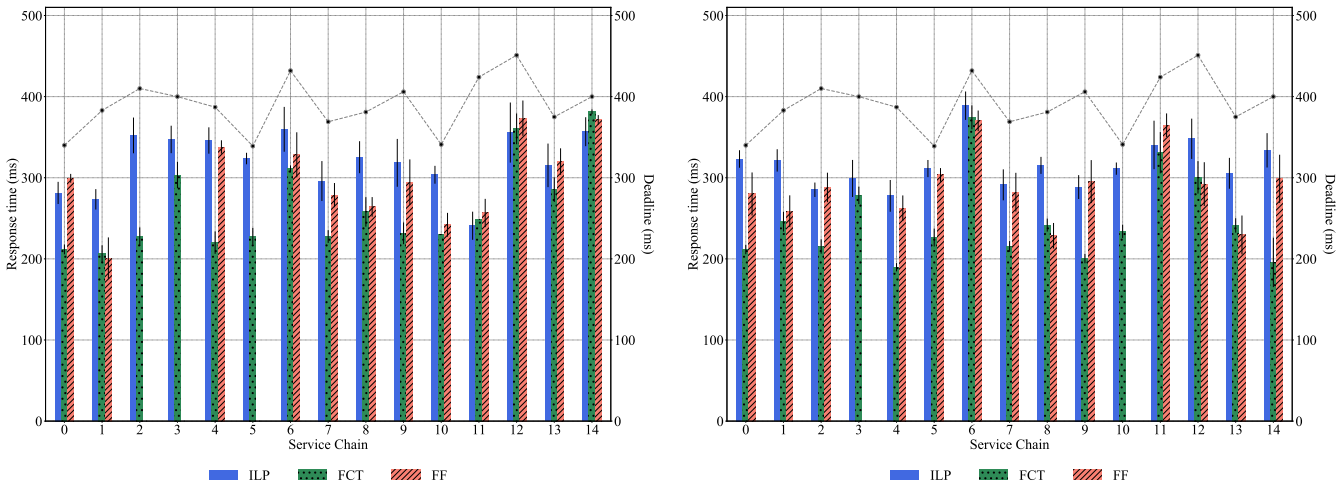
D. DISCUSSION

Overall, ILP showed the best results regarding the resilience and the number of nodes used. The outcomes on the resilience



(a) noReplicas (b) End2End

FIGURE 5. Node utilization.



(a) vsReplicas (b) End2End

FIGURE 6. Service Chains Response Time - Medium scenario.

experiment were expected, considering that the model finds an optimal solution for the embedding of the VFs and their replicas. However, this embedding mechanism requires more time to reach a result; thus, it is not suitable for more complex scenarios that demand quick response times for the embedding. As for the number of nodes, the fact that the ILP model concentrates the VFs in the same optimal nodes leads to a bottleneck for these nodes and their respective links, affecting the performance of the Service Chains that go through them.

On the other hand, FCT achieved a better load balancing for the nodes, by spreading the VFs along the different tier nodes of the infrastructure, which are grouped into communities. Furthermore, by implementing a vertical search inside the communities considering the amount of resources available by tier, and exploiting the possibility of embedding the VFs closer to the edge of the network, FCT obtained better response times, ultimately increasing the QoS for final users while offering a close to the optimal ratio of recovery from failures.

FF was the least effective mechanism of the group, showing the lowest recovery rate from failures. Additionally, it was the only mechanism where some of the Service Chains were not able to complete a successful end-to-end communication. This proves that even though this is a simple mechanism, it is not necessarily adequate to be applied in the Cloud-Fog-Mist-IoT service infrastructure.

Considering the results obtained in the experiments performed and the discussion presented above, we conclude that FCT exhibits a balance between recovery from

failures and response time of applications, making it suitable for the Cloud to IoT landscape modeled in this research.

VIII. CONCLUSIONS

The Cloud-Fog-Mist-IoT continuum offers the infrastructure for virtualized network services, called Virtual Functions. These VFs are grouped in a structure called Service Chains, which are a set of VFs interconnected in order to fulfill a set of specific service/application requirements. The network operator has to provide the proper management and orchestration for said SCs, offering certain availability even in the face of failures. Replicating the VFs among the network infrastructure allows the activation of replicas in case of failures to increase the availability of the SCs. The VFs and their replicas must be strategically embedded in order to enhance resilience.

This work presents a formal grammar that allows the customized specification of SC requests and the replicas of their constituent VFs. Furthermore, two embedding mechanisms for the VFs and their replicas are proposed, one based on ILP and a heuristic based on Fluid Communities. The bi-level ILP model proposed resulted in the embedding of VFs in mostly Fog and Mist nodes, and also in the saturation of nodes that were selected as optimal (based on their availability factor). This saturation is counterproductive, potentially causing more failures in both the nodes and their communication links.

Since the ILP-based solutions are not suited for more complex scenarios and trying to compensate for the drawbacks found in the results, a complementing heuristic based on Fluid Communities was also presented. The mechanism of Fluid Communities by Tiers allowed the creation of balanced communities with an equivalent number of nodes. This heuristic, called FCT, takes advantage of the communities to balance the load among the nodes.

Both mechanisms were evaluated using simulations. The well-known FF approach was used for comparison purposes. Three replication methods were used: no replicas (noReplica), some VFs had replicas (vsReplicas), and all the VFs had replicas (End2End). For all the analyzed scenarios, FF showed less capacity to recover from failures. ILP was the mechanism with less unrecovered failures, but also was the mechanism that saturated the nodes the most. FCT got to balance the load among the nodes, while also improving the response time of the SCs and experiencing fewer failures than FF.

Future work includes adding more dynamism to the scenarios to study the impact of the changes in the availability of the nodes for the embedding mechanisms. Additionally, proposing a learning mechanism that allows taking into consideration previous failures and updating the availability factor of each node, and designing a solution to automatically suggest which of the VFs in an SC should be replicated considering a set of user requirements. Furthermore, the adaptation and validation of the mechanisms for a Cloud-Fog-Mist-IoT federate environment will be considered for the following research.

ACKNOWLEDGMENT

The work presented in this paper was partially carried out in the scope of the project SNOB-5G: Scalable Network Backhauling for 5G (reference CENTRO-01-0247-FEDER-045929) leading to this work is co-financed by the ERDF - European Regional Development Fund through the Operational Program for Competitiveness and Internationalisation - COMPETE 2020, the Center Portugal Regional Operational Program - CENTRO 2020 and the Portuguese Foundation for Science and Technology - FCT under the MIT Portugal Program.

REFERENCES

[1] S. Marston, Z. Li, S. Bandyopadhyay, J. Zhang, and A. Ghalsasi, "Cloud computing—The business perspective," *Decis. Support Syst.*, vol. 51, no. 1, pp. 176–189, Apr. 2011.
[2] K. Velasquez, D. P. Abreu, M. R. M. Assis, C. Senna, D. F. Aranha, L. F. Bittencourt, N. Laranjeiro, M. Curado, M. Vieira, E. Monteiro, and E. Madeira, "Fog orchestration for the Internet of everything: State-of-the-art and research challenges," *J. Internet Services Appl.*, vol. 9, no. 1, pp. 1–23, Jul. 2018.
[3] M. Iorga, L. Feldman, R. Barton, M. J. Martin, N. S. Goren, and C. Mahmoudi, "Fog computing conceptual model," Nat. Inst. Standard Technol. (NIST), Gaithersburg, MD, USA, Tech. Rep. NIST Special Publication 500-325, Mar. 2018.
[4] F. De Turck, J.-M. Kang, H. Choo, M.-S. Kim, B.-Y. Choi, R. Badonnel, and J. W.-K. Hong, "Softwarization of networks, clouds, and Internet of Things," *Int. J. Netw. Manage.*, vol. 27, no. 2, Mar. 2017, Art. no. e1967.
[5] S. Wright, Y. C. Hu, and A. Reid, "Network functions virtualisation (NFV); Infrastructure overview," ETSI Ind. Specification Group (ISG), Stamford, CT, USA, Tech. Rep. DGS/NFV-INF001, Jan. 2015.
[6] J. Quittek, "Network functions virtualisation (NFV); Management and orchestration," ETSI Ind. Specification Group (ISG), Stamford, CT, USA, Tech. Rep. DGS/NFV-MAN001, Dec. 2014.
[7] B. Yi, X. Wang, K. Li, S. K. Das, and M. Huang, "A comprehensive survey of network function virtualization," *Comput. Netw.*, vol. 133, pp. 212–262, Mar. 2018.
[8] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.
[9] A. Gupta, M. F. Habib, U. Mandal, P. Chowdhury, M. Tornatore, and B. Mukherjee, "On service-chaining strategies using virtual network functions in operator networks," *Comput. Netw.*, vol. 133, pp. 1–16, Mar. 2018.
[10] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 4, pp. 1888–1906, Feb. 2013.
[11] M. Schöller and N. Khan, "Network functions virtualisation (NFV); Resiliency requirements," ETSI Ind. Specification Group (ISG), Stamford, CT, USA, Tech. Rep. DGS/NFV-REL001, Jan. 2015.
[12] H. Nakamura, "Network functions virtualisation (NFV); Reliability; Report on models and features for end-to-end reliability," ETSI Ind. Specification Group (ISG), Stamford, CT, USA, Tech. Rep. DGS/NFV-REL003, Apr. 2016.
[13] N. M. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," in *Proc. IEEE INFOCOM 28th Conf. Comput. Commun.*, Apr. 2009, pp. 783–791.
[14] M. T. Beck and J. F. Botero, "Scalable and coordinated allocation of service function chains," *Comput. Commun.*, vol. 102, pp. 78–88, Apr. 2017.
[15] C. Guerrero, I. Lera, and C. Juiz, "Evaluation and efficiency comparison of evolutionary algorithms for service placement optimization in fog architectures," *Future Gener. Comput. Syst.*, vol. 97, no. 1, pp. 44–131, Aug. 2019.
[16] D. Thierens, "Scalability problems of simple genetic algorithms," *Evol. Comput.*, vol. 7, no. 4, pp. 331–352, Dec. 1999.

[17] L. R. Bays, L. P. Gaspary, R. Ahmed, and R. Boutaba, "Virtual network embedding in software-defined networks," in *Proc. NOMS - IEEE/IFIP Netw. Oper. Manage. Symp.*, Apr. 2016, pp. 10–18.

[18] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *Proc. IEEE 3rd Int. Conf. Cloud Netw. (CloudNet)*, Oct. 2014, pp. 7–13.

[19] M. M. Alam Khan, N. Shahriar, R. Ahmed, and R. Boutaba, "Multi-path link embedding for survivability in virtual networks," *IEEE Trans. Netw. Service Manage.*, vol. 13, no. 2, pp. 253–266, Jun. 2016.

[20] M. R. Rahman and R. Boutaba, "SVNE: Survivable virtual network embedding algorithms for network virtualization," *IEEE Trans. Netw. Service Manage.*, vol. 10, no. 2, pp. 105–118, Jun. 2013.

[21] V. B. Souza, X. Masip-Bruin, E. Marin-Tordera, W. Ramirez, and S. Sanchez-Lopez, "Proactive vs reactive failure recovery assessment in combined fog-to-cloud (F2C) systems," in *Proc. IEEE 22nd Int. Workshop Comput. Aided Modeling Design Commun. Links Netw. (CAMAD)*, Jun. 2017, pp. 1–5.

[22] S. Aidi, M. F. Zhani, and Y. Elkhatib, "On improving service chains survivability through efficient backup provisioning," in *Proc. 14th Int. Conf. Netw. Service Manage. (CNSM)*. Rome, Italy, Dec. 2018, pp. 108–115.

[23] I. Lera, C. Guerrero, and C. Juiz, "Availability-aware service placement policy in fog computing based on graph partitions," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 3641–3651, Apr. 2019.

[24] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 69, no. 2, pp. 1–15, Feb. 2004.

[25] V. B. C. Souza, W. Ramirez, X. Masip-Bruin, E. Marin-Tordera, G. Ren, and G. Tashakor, "Handling service allocation in combined fog-cloud scenarios," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–5.

[26] R. Mahmud, K. Ramamohanarao, and R. Buyya, "Latency-aware application module management for fog computing environments," *ACM Trans. Internet Technol.*, vol. 19, no. 1, pp. 1–21, Mar. 2019.

[27] N. Deo, *Graph Theory With Applications to Engineering and Computer Science*, 1st ed. New York, NY, USA: Dover, 2017.

[28] C.-E. Bichot and P. Siarry, *Graph Partitioning*, 1st ed. Hoboken, NJ, USA: Wiley, 2011.

[29] K. Velasquez, D. P. Abreu, L. Paquete, M. Curado, and E. Monteiro, "A rank-based mechanism for service placement in the fog," in *Proc. IFIP Netw.*, Paris, France, Jun. 2020, pp. 64–72.

[30] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner, "Optimized IoT service placement in the fog," *Service Oriented Comput. Appl.*, vol. 11, no. 4, pp. 427–443, Dec. 2017.

[31] F. Parés, D. G. Gasulla, A. Vilalta, J. Moreno, E. Ayguadé, J. Labarta, U. Cortés, and T. Suzumura, "Fluid communities: A competitive, scalable and diverse community detection algorithm," in *Complex Networks & Their Applications VI*. Lyon, France: Springer, Nov. 2018, pp. 229–240.

[32] S. Fortunato, "Community detection in graphs," *Phys. Rep.*, vol. 486, nos. 3–5, pp. 75–174, Feb. 2010.

[33] I. Lera, C. Guerrero, and C. Juiz, "YAFS: A simulator for IoT scenarios in fog computing," *IEEE Access*, vol. 7, pp. 91745–91758, 2019.

[34] D. Perez Abreu, K. Velasquez, M. Curado, and E. Monteiro, "A comparative analysis of simulators for the cloud to fog continuum," *Simul. Model. Pract. Theory*, vol. 101, May 2020, Art. no. 102029.

[35] (2019). *CPLEX Optimizer*. Accessed: Dec. 20, 2019. [Online]. Available: https://www.ibm.com/analytics/cplex-optimizer

[36] M. Jalili and M. Perc, "Information cascades in complex networks," *J. Complex Netw.*, pp. 665–693, Jul. 2017.

[37] E. Bauer and R. Adams, *Reliability and Availability of Cloud Computing*, 1st ed. Hoboken, NJ, USA: Wiley, 2012.

[38] E. G. Nfv, "Network functions virtualisation (NFV); Use cases," Eur. Telecommun. Standards Inst., Sophia Antipolis, France, Tech. Rep. ETSI GS NFV 001 v1.1.1, Oct. 2013.

[39] A. Hmaity, M. Savi, F. Musumeci, M. Tornatore, and A. Pattavina, "Protection strategies for virtual network functions placement and service chains provisioning," *Networks*, vol. 70, no. 4, pp. 373–387, Sep. 2017.

[40] T. Ahmed, A. Alleg, and N. Marie-Magdelaine, "An architecture framework for virtualization of IoT network," in *Proc. IEEE Conf. Netw. Softwarization (NetSoft)*, Jun. 2019, pp. 183–187.

[41] O. Skarlat, M. Nardelli, S. Schulte, and S. Dustdar, "Towards QoS-aware fog service placement," in *Proc. IEEE 1st Int. Conf. Fog Edge Comput. (ICFEC)*, May 2017, pp. 89–96.

[42] D. Perez Abreu, K. Velasquez, L. Paquete, M. Curado, and E. Monteiro. (2020). *Resilient Embedding—GitLab Repository*. Accessed: Jul. 29, 2020. [Online]. Available: https://git.dei.uc.pt/dabreu/ResilientServiceChains.git

DAVID PEREZ ABREU received the B.S. and M.S. degrees in computer science from Central University of Venezuela, in 2005 and 2013, respectively. He is currently pursuing the Ph.D. degree with the University of Coimbra, Portugal.

He worked as a Researcher with the Laboratory for Mobile and Wireless Networks, Central University of Venezuela, from 2006 to 2014. He is also working with the Laboratory of Communications and Telematics, University of Coimbra. He has published several conference and journal papers. His research interests include operating systems, virtualization, cloud computing, resilience, and the Internet of Things.

KARIMA VELASQUEZ received the B.S. and M.S. degrees in computer science from the Central University of Venezuela, in 2005 and 2013, respectively. She is currently pursuing the Ph.D. degree with the University of Coimbra, Portugal.

She worked as a Researcher with the Laboratory of Computer Networks, Central University of Venezuela, from 2006 to 2014. She is also working with the Centre for Informatics and Systems, Laboratory of Communications and Telematics, University of Coimbra. She has published several conference and journal papers. Her current research interests include fog and cloud computing, network performance, and quality of service.

LUÍS PAQUETE received the Ph.D. degree in computer science from TU Darmstadt, in 2005, and the M.Sc. degree in systems engineering and computer science from the University of Algarve, in 2001.

He is currently an Associate Professor with the Department of Informatics Engineering, University of Coimbra. His research interests are within the algorithms for combinatorial optimization problems. In particular, he has been working on basic questions that arise on solution techniques to optimization problems with multiple objectives.

Dr. Paquete is in the editorial board of *Operations Research Perspectives* and is the Area Editor of *ACM Transactions on Evolutionary Learning and Optimization*.

MARILIA CURADO received the Ph.D. degree in computer engineering from the University of Coimbra, Portugal, in 2005.

She is a Full Professor with the Department of Informatics Engineering (DEI), University of Coimbra. She is currently the Director of the Laboratory for Informatics and Systems, Pedro Nunes Institute. She has participated in a large number of national and international projects, in particular, the European FP6 EuQoS, CONTENT, and WEIRD projects, and in the European projects FP7 MICIE, GINSENG, and COCKPIT-CI. She is currently involved in the H2020 ATENA and POSEIDON projects, being the Principal Investigator of the EUREKA Eurostars OUTERMOST and FCT DenseNet projects, and Coordinator of the UC team in the PT2020 MobiWise and Mobilizer 5G projects. She has been involved in the scientific organization and coordination of several international conferences, i.e., WoWMoM, IM, ACM SAC, IoT-SoS, WMNC, and CloudNet. Her research interests include resilience and quality of service in 5G networks, the Internet of Things, and communications in the cloud.

Dr. Curado is a member of the Editorial Board of *Elsevier Computer Networks*, *Elsevier Computer Communications*, *Wiley Transactions on Emerging Telecommunications Technologies*, and *Wiley Internet Technology Letters*.

EDMUNDO MONTEIRO (Senior Member, IEEE) graduated in electrical engineering (informatics specialty) from the University of Coimbra (UC), Portugal, in 1984, and the Ph.D. degree in informatics engineering (computer communications) and the Habilitation degree in informatics engineering from UC, in 1996 and 2007, respectively.

He has more than 25 years of research and industry experience in the field of computer communications, wireless technologies, quality of service and experience, network management, and computer security. He is currently a Full Professor with the Department of Informatics Engineering (DEI), UC. He is also a Senior Member of the Research Centre for Informatics and Systems, University of Coimbra (CISUC). He has participated in many Portuguese and European research projects and initiatives. His publications include six books (authored and edited), several book chapters and journal publications, and over 200 papers in national and international refereed conferences. He is also a coauthor of nine international patents, and is involved in the organization of many national and international conferences and workshops.

Dr. Monteiro is a member of the Ordem dos Engenheiros (the Portuguese Engineering Association), and a Senior Member of the IEEE Communication and Computer Societies and ACM SIGCOMM. He is also a member of the Editorial Board of *Springer Wireless Networks* journals.