

Alma Mater Studiorum Università di Bologna
Archivio istituzionale della ricerca

QoS-Aware Fog Node Placement for Intensive IoT Applications in SDN-Fog Scenarios

This is the final peer-reviewed author's accepted manuscript (postprint) of the following publication:

Published Version:

Herrera, J.L., Galan-Jimenez, J., Foschini, L., Bellavista, P., Berrocal, J., Murillo, J.M. (2022). QoS-Aware Fog Node Placement for Intensive IoT Applications in SDN-Fog Scenarios. IEEE INTERNET OF THINGS JOURNAL, 9(15), 13725-13739 [10.1109/JIOT.2022.3143948].

Availability:

This version is available at: <https://hdl.handle.net/11585/905040> since: 2024-05-07

Published:

DOI: <http://doi.org/10.1109/JIOT.2022.3143948>

Terms of use:

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

This item was downloaded from IRIS Università di Bologna (<https://cris.unibo.it/>).
When citing, please refer to the published version.

(Article begins on next page)

QoS-Aware Fog Node Placement for Intensive IoT Applications in SDN-Fog Scenarios

Juan Luis Herrera, Jaime Galán-Jiménez, Luca Foschini, Paolo Bellavista, Javier Berrocal, and Juan M. Murillo

Abstract—The advent of the Internet of Things (IoT) paradigm to intensive domains, such as industry, is a key enabler for the automation of critical, real-world processes. The strict Quality of Service (QoS) requirements of these domains make low-latency computing paradigms, such as fog computing, very attractive for meeting these requirements. Moreover, the requirements of scalability and flexibility in the underlying network communications motivate the use of Software-Defined Networking (SDN) in the infrastructure. To enable these fog-SDN environments, fog nodes that have both computing and SDN capabilities can be deployed, thus easing the deployment of fog in SDN networks. However, the exact placement of these fog nodes is key to the latency of the hosts that make use of them, and thus, must be carefully assessed to meet the stringent QoS requirements of critical, time-strict IoT applications. This paper focuses on this fog node placement problem by formalizing it and solving it through both optimal and approximated methods, including comparisons with state-of-the-art benchmarks. In particular, we analyze the performance of each of these methods in terms of latency and execution time in both SDN Internet topologies and Industrial IoT infrastructures. Our proposed heuristic provides placements with near-optimal latencies, with smaller optimality gaps than the benchmark, and computes them in tractable times.

Index Terms—Fog computing, Internet of Things (IoT), software-defined network (SDN), Quality of Service (QoS)

I. INTRODUCTION

THE POTENTIAL for real-world process automation brought by the Internet of Things (IoT) paradigm has caught the interest of intensive domains, such as industrial manufacturing, leading to the integration of IoT in industrial processes, termed the Industrial Internet of Things (IIoT) [1]. However, the transition of these domains towards IoT is not simple, as these applications have very high Quality of Service (QoS) requirements, such as low latency and short response

times [1]. These strict QoS requirements directly clash with some of the architectures applied on consumer-grade IoT. Cloud computing, the most popular deployment architecture for IoT applications [2], features the use of *cloud servers* in the core of the network. However, the large distance between the end IoT devices and the network core complicates the achievement of the low latencies required by these applications [3]. This issue has motivated the emergence of new paradigms, such as fog computing, that propose bringing some computing resources, the so-called *fog nodes*, closer to the edge [4]. Thus, fog computing closes the edge-cloud gap in the cloud continuum, lowering the latency between the IoT devices and the fog nodes running their services.

Nonetheless, the location of the servers is not the only factor that affects the QoS of IoT applications, the network infrastructure that connects them together is key, because the impact of server location on QoS heavily depends on the network QoS. When traffic moves through the network fabric, latency increases as longer links and more switches are traversed. While using the least latency path may seem as a good option, the constrained capacity of the network links enforces to apply traffic engineering techniques [5], allowing the network to use alternative paths, even whenever the least latency-influenced path is congested. For this reason, the interest on the use of Software-Defined Networking (SDN) is increasing, in particular in IoT fog infrastructures, where this increase has been experienced in the research community in the recent years [6]. SDN decouples the data plane, which is on charge of forwarding, from the control plane, which takes into more complex network tasks such as routing. SDN controllers, which embody the control plane, can be programmed, enabling for network programmability and, thus, for traffic engineering to be performed [5]. Furthermore, these intensive domains have requirements such as infrastructure scalability and flexibility [1], [6], which can also be met by using SDN.

Therefore, the integration of both fog computing and SDN allows to achieve the requirements of intensive IoT applications. In [7], some of the authors of this work proposed a fog node (FN), specifically designed as an enabler for IIoT applications: a single hardware box combining a SDN switch with fog computing resources. The interest of FNs in intensive IoT scenarios is the ease of migration: an already existing SDN network can be fog-enabled by replacing existing SDN switches with FN boxes, in a similar way than in the case of IP to SDN migration [8]. Nonetheless, FN placement affects the QoS of the applications deployed on the FNs [7], [9]–[12].

Manuscript received January 00, 0000; revised January 00, 0000; accepted January 00, 0000. Date of publication January 00, 0000; date of current version January 00, 0000. This work was partially funded by the project RTI2018-094591-B-I00 (MCI/AEI/FEDER,UE), by the 4IE+ Project (0499-4IE-PLUS-4-E) funded by the Interreg V-A España-Portugal (POCTEP) 2014-2020 program, by the Department of Economy, Science and Digital Agenda of the Government of Extremadura (GR18112, IB18030), by the Valhondo Calaff institution, and by the European Regional Development Fund. (Corresponding author: Juan Luis Herrera.)

J.L. Herrera, J. Galán-Jiménez, J. Berrocal and J.M. Murillo are with the Department of Computer Science and Communications Engineering, University of Extremadura, Spain (e-mail: [jlherrera, jaime, juanmam, jberolm]@unex.es).

L. Foschini and P. Bellavista are with the Dipartimento di Informatica-Scienza e Ingegneria, University of Bologna, Italy (e-mail: [paolo.bellavista, luca.foschini]@unibo.it)

Digital Object Identifier 00.000/JIOT.0000.00000000

Thus, the placement of a FN within the infrastructure is not to be chosen arbitrarily, and instead, it must be assessed in order to obtain the required QoS. Furthermore, the computing resources from FNs are finite, a phenomenon exhibited by FNs having a limited throughput [10]. Thus, it may not be possible to offload all the tasks from all the IoT devices directly into a single FN. This further complicates QoS optimization, as having multiple FNs requires not only assessing the optimal placement for each of the FNs, but also the assignment of which IoT devices offload their tasks to each of the FNs. Moreover, the routing of the traffic between each IoT device and the FN is also key for the QoS obtained, and hence, it must also be optimized in order to achieve the best QoS possible. This optimization can also have different objectives: in very constrained scenarios, it must be ensured that all IoT devices are able to meet the QoS requirement, and thus, maximum latency must be minimized. On the other hand, in less constrained scenarios in which the guarantee that all IoT devices meet the objective QoS is easier to obtain, the optimization can be aimed at minimizing the average latency instead, enhancing the overall QoS.

Some of the authors of the present work also defined in [10] the problem of placing a set of FNs in an infrastructure, assigning IoT devices to FNs and routing the traffic between them to achieve optimal QoS, that they call the Fog Node Placement Problem (FNPP). FNPP is a NP-hard problem [9], as it is a concrete case of a mathematical NP-hard problem, the Capacitated Facility Location Problem [13]. Different objectives for the FNPP can be considered. Nonetheless, [10] only proposes a formulation-based solution that scales poorly, and only considers average latency as an optimization objective. Thus, the main differences between [10] and this paper include: i) the implementation of a new formulation-based FNPP solution that minimizes the maximum latency among all traffic flows, ii) the design and implementation of a heuristic solution for the FNPP based on unsupervised machine learning algorithms, iii) a more extensive evaluation, including larger scenarios in both Internet SDN topologies and IIoT-like scenarios, and iv) a comparison of the defined solutions with a state-of-the-art benchmark.

Considering all the above challenges, the main contributions of this paper are:

- The formalization of the FNPP as a Mixed-Integer Linear Programming (MILP) optimization problem, including two FNPP solutions based on MILP solvers.
- A heuristic algorithm that assesses solutions to the FNPP based on unsupervised machine learning techniques and graph algorithms.
- A performance evaluation of these methods to solve the FNPP in Internet SDN networks, as well as IIoT scenarios, with topologies of varying sizes and considering multiple scenarios.
- A comparison of all the proposed methods by contrasting our solutions with other state-of-the-art ones.

The remainder of this paper is structured as follows. Sec. II presents the system model for the FNPP. Sec. III details the formulation of the FNPP's optimal solutions, while Sec. IV

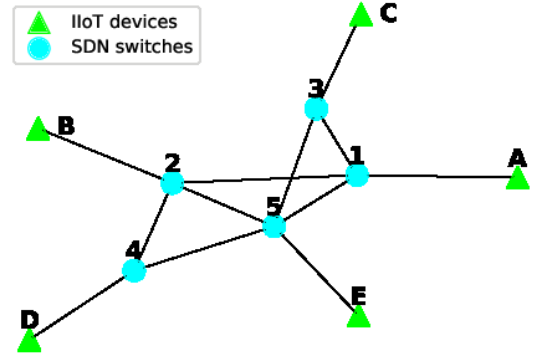


Figure 1: Topology for the example model: IIoT factory automation.

presents our proposed heuristic. An evaluation of FNPP solutions is presented in Sec. V, and Sec. VI compares the FNPP to alternative models proposed in related literature. Finally, Sec. VII concludes our work.

II. SYSTEM MODEL

To explain the FNPP model in detail, an example model of an IIoT application is leveraged in this section. In our IIoT example, a factory automation application is going to be deployed on a SDN network topology. For simplicity's sake, this topology consists of five IIoT devices and five SDN switches, arranged as depicted in Fig. 1. The factory automation application that will be deployed has very strict latency requirements [1], and thus, the factory owner has decided to transform the SDN topology into a SDN-fog infrastructure. To enable this transformation, the factory owner makes use of FNs that follow the model from [7]: hardware boxes that include a SDN switch and a computing device, that will substitute existing SDN switches. These hardware boxes support container-based virtualization, and thus enable for the execution of IIoT services (such as data analysis or computing services) along with performing the function of a SDN switch. To facilitate the understanding of the example, we assume that all links have the same latency, and thus, latencies can be transformed into a number of hops (i.e., traversed links). Therefore, the factory automation application imposes a specific QoS requirement: the maximum latency for the application is one hop (i.e., any path longer than one hop results in an invalid deployment). Based on this situation, we propose two example scenarios for the FNPP: the placement of a single FN (Fig. 2), which is the simplest case of the FNPP, and the more generic placement of multiple FNs (Fig. 3).

In the first scenario, the factory owner will replace a single SDN switch with a FN. The topology has five SDN switches, hence, there are five possible placements for the FN. However, not all placements are equally valid. For instance, let the FN be placed in switch 1, as depicted in Fig. 2a. Assuming a shortest path routing for all devices, we find that IIoT devices A, B, C and E are all able to reach the FN in one hop. However, it is impossible for IIoT device D to reach it in less than two hops. Similarly, if the FN is placed on switch 2, IIoT device C is unable to reach it in one hop. This pattern, in which one

IIoT device cannot reach the FN in an acceptable number of hops, appears in all placements except for switch 5. Thus, the solution to the FNPP is to place the FN in switch 5, which is shown in Fig. 2b. Moreover, to obtain a valid deployment, it is also key that traffic is routed in a specific manner. While it is simple to solve the FNPP in small topologies, such as the example one, manually testing all placements and routing possibilities in networks with hundreds of switches, different latencies in each link and constrained link capacities is not as simple. Therefore, there is a need for an automatic method that solves the FNPP.

In the second scenario, rather than a single FN, the factory owner is willing to replace two SDN switches with FNs. However, these FNs are less powerful than the FN from scenario one and, therefore, each of these FNs can only process the traffic of up to three IIoT devices. Thus, placing the FNs is slightly different from the previous scenario: placing a single FN in switch 5 only guarantees that up to three IIoT devices will be able to reach it in one hop or less. Furthermore, now there is an additional decision to be taken: which IIoT devices should be served by each FN, meeting the capacity constraints of the FNs. This is extremely important, since a bad decision can result in an invalid deployment. For instance, let one FN be placed in switch 5, and the other FN be placed in switch 2, such as represented in Fig. 3a. If IIoT devices A, B, and E are selected to be *assigned* to the FN in switch 5, that leaves IIoT devices C and D for the one in switch 2. However, while IIoT device D can reach switch 2 in one hop, IIoT device C cannot reach it in less than two. This deployment is, thus, invalid. Nonetheless, if IIoT devices A, C and E are assigned to the FN in switch 5, and therefore IIoT devices B and D are assigned to the FN in switch 2, the deployment becomes valid. This assignment option can be seen in Fig. 3b. The main conclusion to draw from this scenario is that placing multiple FNs adds FN-IIoT device assignments to the complexity of the problem. Even in this trivial scenario, there are 10 possible placement combinations for 2 FNs, each of them with 20 possible assignments, for a total of 200 possible solutions, not accounting for the additional combinations that differ in routing. In larger and more realistic scenarios, in which each IIoT device produces a different amount of traffic, each link has a different latency, link capacities are constrained, and there are hundreds of switches and IIoT devices, solving the FNPP manually is infeasible.

III. PROBLEM FORMULATION

The FNPP takes place in a network topology. We model this network topology as an undirected graph $G = \{V, L\}$, with V vertices¹ and L edges. Each of the edges represents a link, e.g., the link from vertex i to vertex j is modeled as $l_{ij} \in L$. Each link also has a capacity, i.e., C_{ij} ; as well as a transmission latency, β_{ij} . On the other hand, each vertex $v \in V$ can either be an IoT device (also called *host*), or a SDN switch. Thus, we can split V into two disjoint subsets:

¹We use the term vertex/vertices to avoid confusion between *nodes* and *fog nodes/FNs*.

Table I: List of formulation notations.

Parameter	Meaning
G	Graph that represents the network
L	Set of links of the network
V	Set of vertices of the network
H	Set of hosts (i.e. IoT devices) of the network
S	Set of SDN switches of the network
C_{ij}	Capacity of link l_{ij}
ϕ_h	Traffic generated by host h
α	Maximum traffic that can be processed by a FN per unit of time
β_{ij}	Propagation latency of link l_{ij}
β_S	Processing latency of a SDN switch
$L(h)$	Latency between host h and its mapped FN
θ	Number of FNs to be placed
Decision variable	Meaning
X_s	Boolean to determine if a FN is placed in switch s
Y_{hs}	Boolean to determine if host h is mapped to the FN located in switch s
f_{ij}^h	Boolean to determine if traffic generated by host h is routed through link l_{ij}

$V = H \cup S; H \cap S = \emptyset$: H . S contains all SDN switches, whereas H contains all the hosts.

Starting with S , the objective of the FNPP is to replace a given number of SDN switches with FNs. We call this number of FNs θ . Furthermore, each SDN switch $s \in S$ is a potential location for a FN. We assume that all FNs to be set in a network have a capacity α for processing traffic. Moreover, switches route the network's traffic, and thus, they have a processing latency of β_S . Continuing with H , hosts generate an amount of traffic that must be processed at a FN. We do not assume that this traffic distribution is uniform, i.e., each host can produce a different amount of traffic. Thus, the traffic generated by a host is labeled as ϕ_h . It is key to understand that both ϕ_h and α must be in the same unit (e.g., Gbps, Mbps, Kbps).

Starting with traffic routing, in the FNPP, we have a set of traffic flows, one per host, of volume ϕ_h . One key characteristic of the FNPP is that we know the source of the traffic flow (the host), but the destination (i.e., the FN assigned to said host) is part of the FNPP solution. Thus, flows need to be modeled based only on their source: let $f_{ij}^h \forall h \in H; l_{ij} \in L$ be a binary variable that takes a value of 1 if the traffic sent by host h traverses link l_{ij} and 0 otherwise. These variables allow the FNPP solution to route traffic, one of the required outputs. Moreover, FNs need to be placed, and therefore, let $X_s, s \in S$ be a binary variable that is 1 if a FN is placed on switch s and 0 otherwise. The final decision that must be taken is the assignment between hosts and FNs. Let $Y_{hs}, h \in H; s \in S$ be a binary variable that becomes 1 if host h is assigned to the FN placed in switch s and 0 otherwise.

The objective of the FNPP is to minimize the latency between hosts and FNs. To simplify further calculations, we define the latency from a host h as the sum of the latencies of the links that its traffic flow needs to traverse, plus the processing latencies of the intermediate switches, if any. Mathematically, $L(h) = (\sum_{l_{ij} \in L} f_{ij}^h (\beta_{ij} + \beta_S)) - \beta_S$. Nonetheless, there are two possible objectives for latency minimization. In our previous works, e.g., [9], [10], we only consider the average latency from all hosts to all switches. However, there

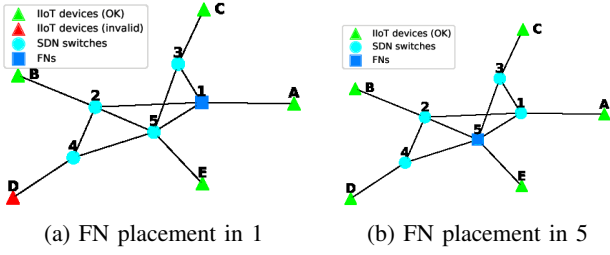


Figure 2: First scenario.

are cases in which, although the average latency is low enough to use the application, the hosts with the highest latencies do not meet the QoS objective [14]. This is exactly the example presented in Sec. II: despite the average latency in Fig. 2a meets the objective (one hop), there is an IIoT device (D) that is unable to use the application, because its latency is higher. In constrained cases in which minimizing the average latency is not enough to meet the QoS objective in every host, it is more desirable to minimize the maximum latency instead. Thus, one of the improvements of this work is the consideration of another objective for latency minimization: the maximum latency among all hosts.

Finally, we also provide a summary table of all the notations used throughout this section in Table I for easy reference.

Therefore, given the previous definitions, the FNPP can be formulated as either (1) or (2):

$$\min \frac{1}{|H|} \sum_{h \in H} L(h) \quad (1)$$

$$\min \max_{h \in H} L(h) \quad (2)$$

subject to:

$$i \in V, h \in H : \sum_{j \in V} f_{ij}^h - f_{ji}^h = \begin{cases} 1 & \text{if } i = h \\ -Y_{hi} & \text{otherwise.} \end{cases} \quad (3)$$

$$\forall l_{ij} \in L : \sum_{h \in H} f_{ij}^h \phi_h \leq C_{ij} \quad (4)$$

$$\sum_{s \in S} X_s \leq \theta \quad (5)$$

$$\forall s \in S : \sum_{h \in H} \phi_h Y_{hs} \leq \alpha X_s \quad (6)$$

$$\forall h \in H : \sum_{s \in S} Y_{hs} = 1 \quad (7)$$

$$\forall h \in H, s \in S, l_{ij} \in L : X_s, Y_{hs}, f_{ij}^h \in \{0, 1\} \quad (8)$$

If (1) is chosen as an objective, the formulation will minimize the average latency. On the other hand, if (2) is chosen instead, the objective will be minimizing the maximum latency. Furthermore, (3) represents the classic flow constraints, and allows traffic to behave as expected (i.e., each host is the source of a traffic flow, the assigned FN to said host is the destination of the flow, and all the SDN switches along the path are neither sources or destinations, only route the traffic).

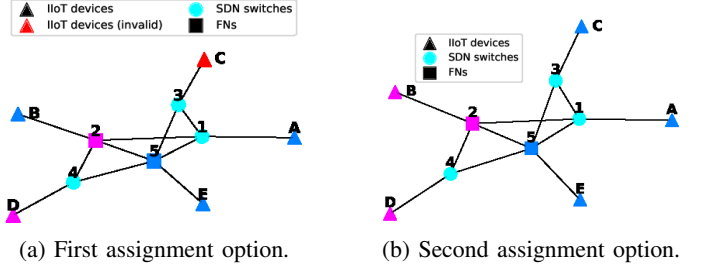


Figure 3: Second scenario.

Similarly, (4) enforces the capacity of each of the links in the infrastructure. No more FNs than θ , may be placed, as per (5), and (6) guarantees that the capacity of each FN is limited to α . Each host can only be assigned to a FN, a constraint that (7) guarantees. Finally, (8) ensures all the defined variables are binary.

As a mathematical problem, the FNPP is a concrete case of the *Capacitated Facility Location Problem* (CFLP) [13]. In the CFLP, a set of facilities with limited capacities must be placed in a graph to meet the demands of users, minimizing the accumulated link weight of the paths between users and facilities. The FNPP follows the same approach: placing capacitated FNs to meet traffic demands coming from IoT devices, in a manner that minimizes the latencies between IoT devices and FNs. The CFLP is a problem proven to be NP-hard [13]. Therefore, the FNPP, which can be reduced to the CFLP, is also NP-hard.

The FNPP formulation presented in this section allows for two solutions to appear: one that minimizes the average latency (labeled *MinMeanLat*, Minimize Mean Latency), and a completely novel one that minimizes the maximum latency (*MinMaxLat*, Minimize Maximum Latency). *MinMaxLat* is preferred in constrained scenarios (e.g., with a small number of FNs, or very limited capacities), as it will make every host meet the QoS objective if possible. However, in less limited scenarios in which the QoS objective can be met more easily, it is more desirable to find a better performing solution in average. For those cases, *MinMeanLat* should be used instead. Both *MinMeanLat* and *MinMaxLat* are able to be parameterized, and hence, they can be applied to different scenarios: multiple network topologies, number of FNs to be placed, link capacities, FN capacities, traffic distributions, etc. The application of MILP guarantees that the formulation yields results that are, in fact, optimal. However, while these solutions are valid methods to solve the FNPP, MILP solvers tend to require a very high amount of resources (i.e., RAM, execution time) [15]. Furthermore, these methods do not tend to scale well with the problem size, generally increasing their resource consumption in an exponential manner [15]. Thus, there is also a need for heuristic solutions that are able to scale better and solve the problem with fewer resources.

IV. HEURISTIC DESCRIPTION

In this section, we present an heuristic for the solution of the FNPP. This heuristic is motivated by the NP-hardness of the problem, which results in a high amount of time and resources required by the MILP-based solutions, along with their poor

Algorithm 1: Midpoint selection of initial centroids

```

1 Input:  $S$ : set of SDN switches;
2  $\theta$ : number of FNs to place;
3 Output: centroids: initial centroids;
4 begin
5   interval :=  $\frac{|S|}{\theta}$ ;
6   first := 0;
7   last := interval;
8   centroids :=  $\emptyset$ ;
9   for  $i := 0$  to  $\theta$  do
10    candidates :=  $S[\text{first}:\text{last}]$ ;
11    centroid := candidates[ $\frac{|candidates|}{2}$ ];
12    centroids := centroids  $\cup$  {centroid};
13    first := last;
14    last := last + interval;
15    if first = last then
16      last := last + 1;
17    end
18  end
19 end

```

scalability. Thus, the objective of the heuristic is to be a fast, lightweight, scalable, near-optimal method to solve the FNPP.

The presented heuristic can be structured as four main algorithms that are executed sequentially. Nonetheless, to understand the role of each of the algorithms, it is necessary to understand a part of its core basis first. The heuristic is based on an unsupervised machine learning algorithm named *k-medoids* [16]. While the specifics of *k-medoids* will be explained later, it can be understood as a method that, starting from some initial FN placements called *centroids*, will *move* FNs towards placements that have low latencies. Thus, the initial FN placements need to be assessed before being able to make use of *k-medoids*. It is crucial to understand that these FN placements are merely initial, and rarely FNs stay in their initial positions. Despite this behavior, the initial centroids fed to *k-medoids* do affect its outcome. Our heuristic supports three criteria for initial centroid assessment: midpoint selection, highest betweenness centrality, and random. While the two latter lack a description, since they are self-explanatory, midpoint selection is detailed in Algorithm 1. The computational complexity of Algorithm 1, similarly to the rest of the algorithms used for initial centroid selection, it simply needs to iterate over the number of FNs to place. Therefore, the worst-case complexity of centroid selection is $O(\theta)$. While random and midpoint selection have been used in related literature to place FNs [11], the application of graph metrics such as betweenness centrality to initial centroid assessment is novel.

Once initial centroids are assessed, *k-medoids* can be leveraged. This algorithm is an unsupervised learning clustering algorithm: given a data structure, *k-medoids* divides the structure into clusters of data points, so that points in the same cluster are as similar as possible to each other, and as dissimilar as possible to points in other clusters. We have implemented our

own version of *k-medoids*, which uses latency as the similarity metric and is described in Algorithm 2. Conceptually, *k-medoids* splits the network into θ partitions (i.e., subsets of vertices that are closest to each other), and places a FN in the vertex of each partition with lowest latencies compared to the rest. Internally, *k-medoids* assumes a set of initial centroids (I), and creates a set of vertices associated to each centroid, which is called the centroid's cluster (lines 5-19). To do so, *k-medoids* calculates, for each vertex in the infrastructure, that vertex's latency to each centroid (lines 9-12). Each vertex is then added to the cluster of the centroid with minimal latency to it (lines 13-19). Thus, *k-medoids* is able to generate a network partition per centroid, which contains all the vertices with minimal latency to the centroid, i.e., its cluster. Then, for each of these partitions, *k-medoids* calculates which vertex in the partition has the minimal average latency to the rest, i.e., which vertex is, considering latency as distance, the *center* of the partition (lines 20-38). This vertex will become the centroid of the partition. Two scenarios can appear at this step: either the newly-calculated centroids may be the same as the old centroids (line 35 is never executed), or at least one centroid has changed (line 35 is executed). In the first case, *k-medoids* is said to converge, and hence, the centroids are the final placement for FNs. However, any change in the centroids indicates that the placement can be enhanced: even assuming clusters tailored for each centroid, there is another vertex in the cluster that is a better centroid. Thus, *k-medoids* is executed again, and the newly-calculated centroids are the initial centroids for this new iteration (lines 39-41). The recursiveness of *k-medoids* guarantees that the process is repeated until convergence is reached. Regarding the complexity of the *k-medoids* algorithm, lines 20-38 in Algorithm 2 are the most significant. In line 27, the shortest path is calculated using Dijkstra's algorithm, which is known to have a worst-case complexity of $O(|V| \log |V|)$ [17]. This procedure call is performed inside a triple-nested loop, giving these lines a total complexity of $O(\theta |V|^3 \log |V|)$. However, since *k-medoids* is recursive, this is repeated until convergence. Thus, Algorithm 2 has a total complexity of $O(c_{it} \theta |V|^3 \log |V|)$, where c_{it} is the number of iterations required for convergence.

The third main algorithm of the heuristic is the assignment algorithm, which decides, for each host, which FN should it send its traffic to. To do so, the heuristic first sorts the hosts, in ascending order, using the size of their traffic flows as the criteria (line 9). Thus, smaller flows are assigned first to minimize the average latency: smaller flows are given priority, so more flows can have smaller latencies, and hence, average latency also shrinks. Then, for each of the hosts, the algorithm finds the FN with minimal latency that has enough remaining capacity to process its traffic flow (lines 10-18). This FN is then assigned to the host. The behavior is detailed in Algorithm 3. Complexity-wise, the most costly part of the algorithm is line 11. In this line, FNs are sorted using the shortest path's distance as criterion. If this sort was performed using TimSort, the default algorithm in languages such as Python, the worst-case complexity of the sorting algorithm would be $O(\theta \log \theta)$ [18]. Furthermore, using the shortest path as a criterion entails using Dijkstra's algorithm to calculate it,

Algorithm 2: Modified k-medoids

```

1 Input:  $G$ : topology graph;
2  $I$ : set of initial centroids;
3 Output:  $F$ : placement for FNs;
4 begin
5    $\mathcal{C} := \text{dictionary}()$ ;
6   for  $i \in I$  do
7      $\mathcal{C}_i := \emptyset$ ;
8   end
9   for  $v \in G.V$  do
10     $\text{minDist} := \infty$ ;
11    for  $i \in I$  do
12       $\text{centDist} := \text{shortestPathDistance}(v, i, G)$ ;
13      if  $\text{centDist} < \text{minDist}$  then
14         $\text{minDist} := \text{centDist}$ ;
15         $\text{minCentroid} := i$ ;
16      end
17    end
18     $\mathcal{C}_{\text{minCentroid}} := \mathcal{C}_{\text{minCentroid}} \cup v$ ;
19  end
20   $F := \emptyset$ ;
21   $\text{changes} := 0$ ;
22  for  $i \in I$  do
23     $\text{minDist} := \infty$ ;
24    for  $v_1 \in \mathcal{C}_i$  do
25       $\text{totalDist} := 0$ ;
26      for  $v_2 \in \mathcal{C}_i - \{v_1\}$  do
27         $\text{totalDist} := \text{totalDist} +$ 
28           $\text{shortestPathDistance}(v_1, v_2, G)$ ;
29      end
30      if  $\text{totalDist} < \text{minDist}$  then
31         $\text{totalDist} := \text{minDist}$ ;
32         $\text{newCentroid} := v_1$ ;
33      end
34      if  $\text{newCentroid} \neq i$  then
35         $\text{changes} := \text{changes} + 1$ ;
36      end
37     $F := F \cup \{\text{newCentroid}\}$ ;
38  end
39  if  $\text{changes} > 0$  then
40     $F := \text{modifiedKMedoids}(G, F)$ ;
41  end
42 end

```

with its own complexity of $O(|V| \log |V|)$ [17]. Therefore, the complexity of Algorithm 3 is of $O(\theta|V| \log \theta \log |V|)$.

The final algorithm takes the only remaining decision: routing. The heuristic uses a common method for routing: k-shortest path. In the classic k-shortest path, k paths between each host and its assigned FN are calculated, and sorted in ascending order according to their latency. Then, for each of the paths, the capacity of the links traversed by the path is checked. If all the links have enough capacity to route the traffic, the path will be chosen as the definitive one between the host-FN pair, and the size of the traffic flow will be

Algorithm 3: Host-FN assignment

```

1 Input:  $G$ : topology graph;
2  $F$ : placement for FNs;
3  $\alpha$ : FN capacity;
4  $\phi$ : vector of traffic flows.  $\phi_h$  is the size of the traffic
   flow of host  $h$ ;
5 Output:  $A$ : assignments, in dictionary form.  $A_h$  is the
   FN assigned to host  $h$ ;
6 begin
7    $A := \text{dictionary}()$ ;
8    $\text{remainingCap} = \text{dictionary}(\text{keys}=F, \text{values}=\alpha)$ ;
9    $\text{sortedH} := \text{ascendingSort}(H, \text{criteria}=\phi)$ ;
10  for  $h \in \text{sortedH}$  do
11     $\text{FNCandidates} := \text{ascendingSort}(F,$ 
12       $\text{criteria}=\text{shortestPathDistance}(h, G))$ ;
13    for  $f \in \text{FNCandidates}$  do
14      if  $\phi_h \leq \text{remainingCap}_f$  then
15         $A_h := f$ ;
16         $\text{remainingCap}_f :=$ 
17           $\text{remainingCap}_f - \phi_h$ ;
18        break;
19      end
20    end
21  end

```

deducted from the remaining capacity of the links. Otherwise, the next path is selected. The heuristic computes these paths *lazily*: it copies the original graph (line 10), and calculates the shortest path between the host and the FN (line 12). If link capacity holds (lines 14-19), it is decided to be the path for the host-FN pair (line 21). If link capacity does not hold, all the links without enough remaining capacity are removed from the graph's copy (line 18), and the shortest path is calculated again (lines 11-21). This process is repeated until either there are no paths between the FN and the host, or a suitable path is found. The details of the routing algorithm can be found in Algorithm 4. The complexity of this routing algorithm is mainly line 12's, which is $O(|V| \log |V|)$ [17]. However, the shortest path is recalculated every time the capacity constraints are not met. At worst, a single link will be removed from the graph on every iteration, and therefore, it will be re-calculated $|L|$ times. As the paths need to be calculated for every host, the worst-case complexity of Algorithm 4 is $O(|H||L||V| \log |V|)$.

The heuristic makes use of all the algorithms described in this section, in the same order they have been presented: first, it generates an initial set of centroids, using either midpoint, HBC or random selection. That initial set of centroids is fed to the modified k-medoids, which yields the placement for the FNs. With this placement, the information about the size of the traffic flows and the capacity of the FNs, the heuristic assigns hosts to FNs. And finally, based on these assignments, it routes each of the traffic flows. However, simply pipelining the algorithms in this manner may lead to feasibility problems, as each step cannot undo the decisions taken by previous steps. This is something common in greedy

Algorithm 4: Routing algorithm

```

1 Input:  $G$ : topology graph;
2  $\phi$ : vector of traffic flows.  $\phi_h$  is the size of the traffic
   flow of host  $h$ ;
3  $A$ : assignments, in dictionary form.  $A_h$  is the FN
   assigned to host  $h$ ;
4 Output:  $R$ : routes, in dictionary form.  $R_h$  is the path
   from  $h$  to its assigned FN;
5 begin
6    $sortedH := \text{descendingSort}(H, \text{criteria}=\phi)$ ;
7    $R := \text{dictionary}()$ ;
8   for  $h \in sortedH$  do
9      $finalPath := 0$ ;
10     $G' := \text{copy}(G)$ ;
11    while  $finalPath \leq 0$  do
12       $route := \text{shortestPath}(h, A_h, G')$ ;
13       $finalPath := 1$ ;
14      for  $l_{ij} \in route$  do
15         $cap = \text{capacity}(l_{ij}, G')$ ;
16        if  $cap < \phi_h$  then
17           $finalPath := 0$ ;
18           $\text{removeLink}(l_{ij}, G')$ ;
19        end
20      end
21    end
22     $R_h := route$ ;
23  end
24 end

```

algorithms [11], such as the ones used in this heuristic, but leads to possible feasibility problems, as each decision affects all the following ones. Although the sorting in Algorithms 3 and 4 try to avoid these situations, sometimes the heuristic may not find a solution. Nonetheless, it is key to understand that such case does not mean that there is no solution: rather, it means some traffic flows must be offloaded to the cloud, rather than to the fog, as [11] explains. However, the QoS-strict IoT applications treated in the FNPP may not properly work with cloud offloading. Hence, this heuristic adds a *retry system* to minimize these situations, up to a given number of retries, which is a parameter for the heuristic on its own. To guarantee that the solutions are different, and thus, that each retry will find a different solution, the heuristic makes use of all three criteria for finding the initial centroids. First, it tries to use midpoint selection, as it has shown the best results. On the next retry, it uses highest betweenness centrality, which normally allows the heuristic to find feasible results, although with higher latencies than midpoint. If this criteria fails, random selection is used for the rest of the retries, as it guarantees different initial centroids on each retry. The end user can also select the initial criteria if they want to skip to highest betweenness centrality or random directly on the first retry. This behavior and pipelining is represented in Fig. 4. Out of all the modules of the heuristic, the most complex one is Algorithm 2, k-medoids. Within the final heuristic, k-medoids needs to be re-executed on every retry,

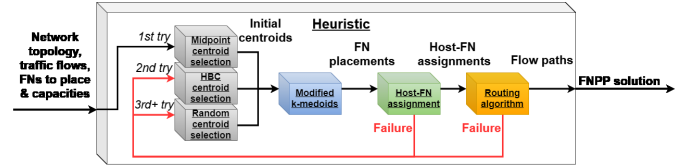


Figure 4: Heuristic behavior diagram.

and therefore, the worst-case complexity of the heuristic is $O(Rc_{it}\theta|V|^3 \log|V|)$, where R is the number of retries. In conclusion, the heuristic's complexity depends the most on the topology size in terms of vertices, the number of FNs to place, the number of iterations k-medoids requires to converge and the number of retries selected.

An interesting feature that stems from the modular nature of the heuristic is its ability to adapt itself to dynamic environments. In the FNPP, each FN is a hardware box, and therefore, changing the placement of FNs in real time is not feasible. Nonetheless, the host-FN assignments and the paths followed by traffic can be changed in real time. To do so, whenever a change is detected, such as an IoT node moving from one part of the network to another, new IoT devices being added to the network, or a change in the traffic demands, the heuristic can be triggered to recalculate host-FN assignments and routing. This recalculation is performed by feeding the current FN placements as F to Algorithm 3, and using the output of the recalculation for Algorithm 4. This sort of execution guarantees that the FNs will not change their placement, and lowers the complexity to $O(|H||L||V| \log|V|)$, as the first two algorithms are skipped and there is no need to perform retries because none of the remaining algorithms have random components. However, the MILP-based solutions do not support this partial execution, only the proposed heuristic supports it.

V. PERFORMANCE EVALUATION

In this section, we present the evaluation of the FNPP solution methods. Scenarios in 5 topologies have been tested with 6 methods for solving the FNPP each: placement through highest betweenness centrality (HBC), placement through highest closeness centrality (HCC), our proposed heuristic, the proposed heuristic of Maiti *et al.* [11] (which is used as a benchmark), the MILP-based optimal solution that minimizes the mean latency of all hosts (MinMeanLat), and the MILP-based optimal solution that minimizes the maximum latency among all hosts (MinMaxLat). The objective of comparing our solutions with HBC and HCC is to evaluate the difference between specifically designed FNPP solutions and simple placement criteria. We assume that, without knowing about the FNPP, a network administrator would place FNs following either HBC or HCC, and thus, our aim is obtaining better results than both criteria. The benchmark, which is thoroughly described in [11], is another heuristic for solving the FNPP proposed in related literature. The benchmark also uses a version of k-medoids to place FNs in the network, although it lacks assignment and routing algorithms compared to the heuristic proposed in this paper.

It is important to note that HBC, HCC and the benchmark only feature a node placement algorithm. Therefore, assignment is made in a similar manner to the heuristic, in order to have a fair comparison between methods. Routing is performed using k-shortest path, although the weight of links is not their latency, but the inverse of their remaining capacity. This choice for the links' weight setting is used because, in some cases, some of the methods are unable to find a fog-only deployment (e.g., their strategy assigns host to FNs in such a manner that they run out of free capacity to satisfy all hosts). In these cases, the unsatisfied hosts would have to offload their demands to the cloud instead, similar to the approach presented in [11]. Using their remaining capacity for routing minimizes the amount of cases in which these methods require sending data to the cloud, hence enabling for a more clear and fair comparison between FN placement methods. Moreover, to maintain a fair comparison between all methods, and to ease on the visualization of the results, these cases are reported as if the method was unable to find a solution: if the method requires to make use of the cloud, its results are not depicted in the graphics. Despite this lack of visualization, it is important to note that the methods never fail to find solutions, rather, they fail to find a solution that does not make use of the cloud.

A. Evaluation setup

The scenarios used for testing these methods can be divided into two categories of experiments: i) *SDN deployment*, and ii) *IIoT deployment scenarios*. The first category includes *SDN deployment* scenarios, which have tested the FNPP under four topologies: Abilene, GEANT, Germany-50 and Brain. Their information, including topology details, link capacities, latencies and traffic matrices have been obtained from [19]. The objective of SDN scenarios is to evaluate the performance of the FNPP over real, SDN networking scenarios. A host is considered to be connected to each switch in order to be the source of the information to be processed at a FN.

The second category, instead, is the *IIoT deployment* category, which contains a single fog topology. The objective of IIoT deployment scenarios is to validate the FNPP solution in a large IIoT scenario based on a topology with a dense edge. Concretely, we aim to use a real, pre-existing topology, so the evaluation is performed in a topology that has been designed using a real rationale. Hence, this topology is a modified version of Brain, a real topology with a dense edge. In this version, the 152 switches at Brain's edge, concretely those with numbers in their labels, are treated as hosts, while the remaining 9 are left as SDN switches. However, Brain is a Germany-wide topology, rather than an industrial facility-wide one. Thus, Brain has been resized (i.e., the length of the links has been shortened) through linear interpolation, so that it is the same size as the Boeing Everett Factory, because it is the largest industrial factory in the world [20]. This resizing affects the latency of the links, as information is sent within the industrial facility rather than across a country. Furthermore, IIoT scenarios have their link capacity limited to analyze the effect of link capacity limits. These scenarios

are labeled Light Capacity Limit (LCL) and Heavy Capacity Limit (HCL). In LCL, the link capacities are set to a maximum of an 125% of the heaviest demand using linear interpolation: after routing the heaviest demand, the links still have capacity left (concretely, a 25% of said demand) to route other traffic flows. In HCL, link capacity is set to a maximum of the heaviest demand instead: links used to route the heaviest demand cannot be used to route any other traffic. The LCL scenarios are also divided into *scaled* (i.e., α varies with θ so that the aggregate α of all FNs stays constant) and *unscaled* (i.e., α is a fixed, constant value) to assess which one, link or FN capacity, is more restrictive, while HCL does not have this objective and is always unscaled. Finally, an additional IIoT deployment scenario featuring all 161 switches and scaled FN capacities has also been used to evaluate the effect of topology size and complex placement decisions on the average latency.

To assess traffic, we obtained the traffic matrices from [19], and obtained the peak matrix. We label this as *traffic matrix 5*. Then, we scale traffic matrix 5 by multiplying it by 0.4, 0.5, 0.7 and 0.9, generating traffic matrices 1, 2, 3 and 4, respectively [5]. These matrices are used to simulate an increasing amount of traffic in the network. In the following subsections, we show the results of the evaluation, first on SDN deployment scenarios, and then on IIoT deployment ones. Moreover, we have performed emulations on Mininet for some of the SDN deployment scenarios. In these emulations, we have created a series of hosts that send their traffic to their assigned FNs using *iperf*, while they assess their latency as half of the round trip time obtained with *ping*. The Mininet emulations have been performed on an Amazon Web Services t2.large instance.

The objectives of this evaluation are to assess the impact of the number of placed FNs (i.e., θ), FN capacity (i.e., α), traffic and link capacity in the latencies experienced by flows in the network, as well as in the time required to obtain a solution. Thus, four metrics are used to evaluate the solutions: the mean latency from all hosts to their assigned FNs, the maximum FN-host latency throughout all hosts, the statistical distribution of the latencies and the time required to find a solution. The first three metrics are QoS-related, and thus, allow for a user to make a choice on a solution based on the QoS requirements of their concrete use case (e.g., picking a method with lower maximum latency, even if it has higher average latency, because the scenario complicates guaranteeing that all devices can meet the QoS requirements). The latter, although not directly related to the QoS of the application, allow users to also consider scalability and resource consumption to choose a method. Moreover, we aim at comparing the effects of all these parameters in each of the six FN placement methods previously mentioned, in experiments involving both Internet and IIoT-oriented fog SDN topologies.

B. Performance analysis - SDN deployment

The first analysis consists on assessing the impact of the number of FNs to be placed in the network (θ), and each placement method's performance, w.r.t. average latency. Fig. 5 depicts the results of the analysis in the Abilene topology.

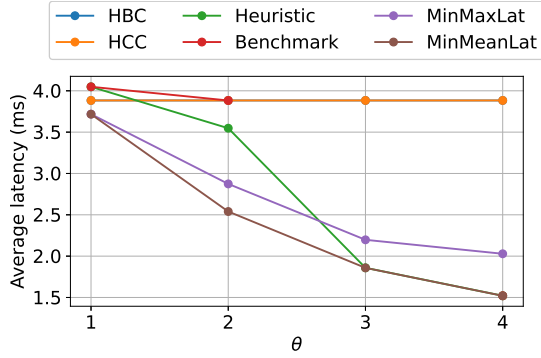


Figure 5: Average latencies in Abilene.

The main conclusion is that the correlation between θ and average latency is inverse. In the case of optimal placement, the correlation is similar to a harmonic progression. This is caused by the fact that, in the best case, the latency achieved with a single FN is divided by the number of FNs placed (i.e., with 2 FNs, the latency will be at best $\frac{1}{2}$ of the original, with 3 FNs it will be $\frac{1}{3}$, etc.). The decreasing trend is mimicked by every method except HBC and HCC, which have an slightly increasing trend instead. Method-wise, the optimal solution is always MinMeanLat, generally followed by MinMaxLat. The heuristic is the third best method, after which comes the benchmark, and finally HBC and HCC yielding the worst results. It is also important to note that the benchmark requires a cloud deployment in $\theta = 3$. Finally, the performance gap between the heuristic and the optimal solution tends to decrease in higher θ values, with a 1 ms gap with $\theta = 2$ and merely 0.0005 ms of difference with $\theta = 4$. The difference between the heuristic and the benchmark depends on the value of θ : with $\theta = 1$, they yield the same solution, as they use the same method to place the single FN. Nonetheless, as more FNs need to be placed, the different assignment methods show a difference of up to 0.34 ms in $\theta = 2$. In $\theta = 3$, since the benchmark requires cloud usage, this gap increases to approximately 45 ms.

Fig. 6 shows the average latencies in larger topologies: GEANT (Fig. 6a), Germany-50 (Fig. 6b) and Brain (Fig. 6c). Performance-wise, we can see in general that HBC and HCC yield the highest latencies, followed by MinMaxLat, which highly varies in this performance metric, raising from the 4th place to the 2nd in most high θ scenarios. The benchmark and the heuristic yield similar results, but the benchmark tends to require the cloud with higher values of θ such as 3 and 4, whereas the heuristic can successfully make use of fog-only deployments in all cases. Finally, MinMeanLat is the optimal method, and hence, always yields the best average latencies. In more detail, in GEANT, we find that all methods except HBC and HCC follow a decreasing trend, similar to the results in Fig. 5. Nonetheless, this trend is not as quick to decrease as it was in Abilene. This is mainly related to the shape and distribution of vertices in the infrastructure: while Abilene is a more sparse topology, with all vertices separated by similar distances within the USA, GEANT has a dense core in central Europe, along with some sparse vertices

separated by much longer distances (e.g., London-New York). Hence, to have a quickly decreasing trend, FNs must be placed in such a manner that the dense core has very low latencies, and the vertices far away from the core have FNs placed in them. As MinMeanLat shows, this does not yield optimal average latencies, hence, the trend does not decrease as quickly as before. This also applies to the rest of the methods, which exhibit similar trends. Continuing with Germany-50, the trends of MinMeanLat, MeanMaxLat and the heuristic are all decreasing. The heuristic decreases less sharply, while MinMaxLat starts higher than the heuristic. On the other hand, HBC and HCC exhibit a large difference in this topology: HCC yields better latencies (approximately 1 ms lower) than HBC. Moreover, the benchmark exhibits the opposite behaviour, with its latency raising as θ increases. Finally, in the Brain topology, we find a trend with sharper decreases than in the previous topologies, due to the density of the topology. All the methods, except HBC and HCC, exhibit a decreasing trend. Most interestingly, the heuristic and MinMeanLat are almost parallel, with an optimality gap of approximately 0.27 ms. The benchmark could fall into this category as well if it did not depend on the cloud for $\theta = 3$ and 4. Overall, we conclude that the trends followed by average latencies as the number of FNs increases depend on topology density, and that the heuristic tends to perform in a similar manner to MinMeanLat, and does not require for the cloud as often as the benchmark.

Fig. 7 depicts the cumulative distribution function (CDF) of the latencies experimented by each of the flows in the emulated Mininet environment, in a situation with $\theta = 4$. MinMeanLat rises quickly, with a 25% of the flows staying under 64 ms of latency. Nonetheless, there is a 20 ms gap to the remaining 75%, which steadily rises from 81 to 97 ms. Overall, we find a behaviour in which there is a clear separation between low-latency flows and high-latency flows. MinMaxLat and the heuristic exhibits a very similar behaviour, the heuristic rising faster, but MinMaxLat having lower latencies in the highest latency flows (106 ms with MinMaxLat, 109 ms with the heuristic). Nonetheless, both rise in a very steady manner, only exhibiting a gap on the higher quarter. The benchmark exhibits higher latencies at both the lowest latency flows and the highest latency flows, while those in the middle are similar to the heuristic and the MILP-based solutions. Finally, both HBC and HCC yield very bad results, with a clear gap of 20 ms after the lower 10% of the flows, and with a latency of approximately 1.5 times the one achieved by the heuristic and MILP-based solutions.

The execution time analysis is depicted in Fig. 8. This analysis makes use of the largest θ value to maintain fairness and compare all solutions in a worst-case scenario for execution time, since their execution time is directly related to the value of θ . This analysis allows for a study of the scalability of each of the methods, which may be key for their usage in larger topologies. The main conclusion that is drawn from this figure is that MinMaxLat does not scale well: it is the slowest method on every case, and the gap between MinMaxLat and the rest of the methods becomes extremely large in topologies with 50 nodes or more (e.g., Germany-50, Brain). In the worst-case scenario, MinMaxLat takes about 12

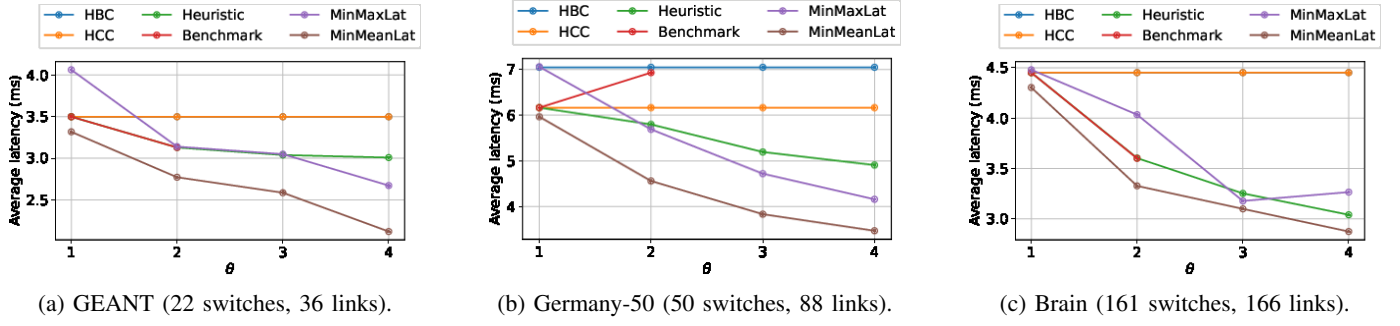


Figure 6: Average latencies in other topologies

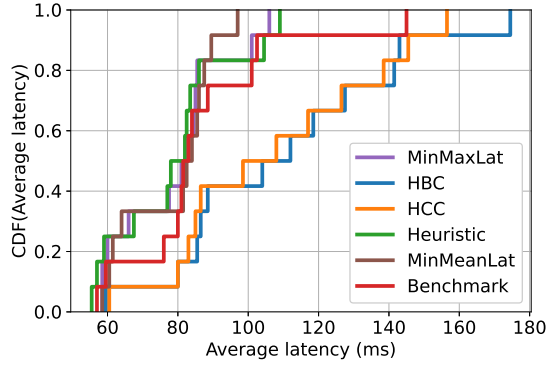
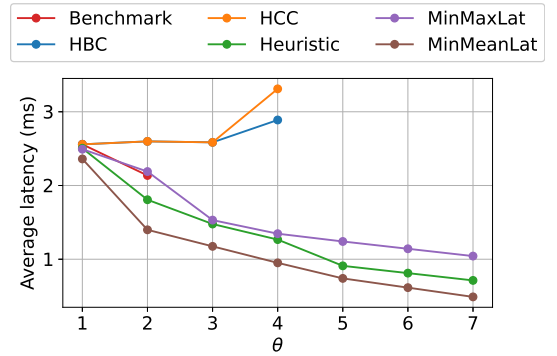
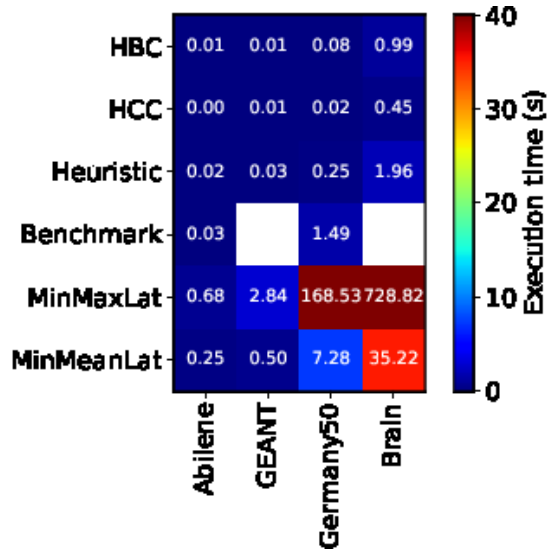
Figure 7: CDF of emulated latency in Abilene, $\theta = 4$ 

Figure 9: Average latencies in LCL-scaled.

Figure 8: Optimization times for all topologies, $\theta = 4$.

minutes to optimize, a value extremely large compared to the rest of the methods, that take less than a minute. In order not to occlude the rest of the methods because of the results of MinMaxLat, the color palette has been adjusted for the rest of the methods (i.e., between 0 and 40 seconds). Continuing with MinMeanLat, its scalability is not good either: despite being a fairly competitive method in smaller topologies such as Abilene or GEANT, solving the FNPP in 0.24 to 0.51 seconds, the times in Germany-50 (7.28 seconds) and Brain (35.22 seconds) rise extremely fast. Concretely, we find that

duplicating the topology size brings with it an increase of approximately 14 times the execution time. This is common in MILP solving, which normally has exponential temporal complexity [15]. Thus, although 35 seconds is still a short time, it may increase heavily on larger topologies. All the remaining methods have very good scalabilities: HCC is the fastest method ranging from 0.003 to 0.45 seconds, with HCC following it (0.005 seconds to 1 second). The benchmark also scales well (0.03 to 1.49 seconds), and the heuristic is also very close (0.01 to 1.96 seconds). Considering the previously analyzed results, we find that, despite the MILP optimization is able to solve the FNPP in tractable time in topologies with tenths or hundreds of nodes, it may be prohibitive in topologies with thousands of nodes. Therefore, we recommend the usage of the heuristic in such topologies.

C. Performance analysis - IIoT deployment

The first analysis to be performed in the IIoT deployment scenarios is the average latency analysis. Very similar to the average latency analyses performed in the SDN deployment scenarios (Figs. 5, 6), these allow for the assessment of the average performance of a flow.

The first IIoT analysis, performed in the LCL-scaled scenario and depicted in Fig. 9, aims at assessing the effect of θ in average latency. While LCL-scaled does feature a more stringent link capacity limit, we find out that FN capacity (α) is much more impacting. This is mainly because stringent link capacity limits give a larger window for algorithms to maneuver after setting a path, by using alternative paths for newer flows. However, stringent FN capacity limits make each

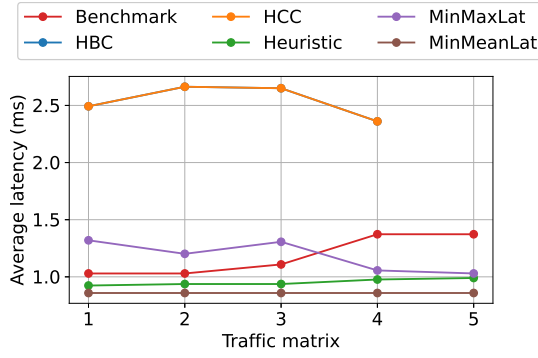
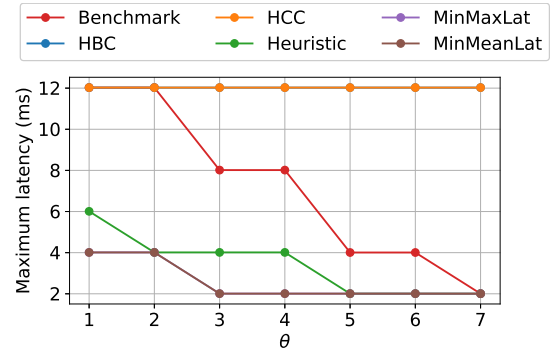
Figure 10: Average latencies in HCL, $\theta = 4$.

Figure 11: Maximum latencies in HCL.

assignment decision vital, especially for algorithms without a retrying system, since finding an alternative FN-host assignment after making an inefficient decision can be impossible. Despite this fact, we still find the effect of traffic and link capacity in Fig. 9. First, HBC and HCC exhibit the worst results, followed by the benchmark, which is unable to yield fog-only deployments any further than $\theta = 2$, MinMaxLat, the heuristic, and MinMeanLat being the best method. Moreover, we also find that HBC and HCC fail to find a fog-only deployment after $\theta = 4$. This result comes from FN capacity limits: LCL-scaled reduces the FN capacity with higher θ , so that the aggregated capacity from all FNs remains constant. Hence, since neither HBC or HCC have a retry system either, a series of inefficient decisions lead them to requiring cloud usage. Furthermore, latencies are more differentiated in $\theta = 1$ than in other scenarios: MinMeanLat yields an average latency 0.2 ms lower than the rest of the methods, which also exhibit a gap, although smaller (0.05 ms). This difference comes from routing algorithms: the benchmark, HBC and HCC share very similar latencies because they share a routing algorithm. The heuristic makes use of a different algorithm, that allows it to reduce the average in 0.05 ms, and MinMaxLat exhibits slightly different (0.01 ms) results as well. Finally, the heuristic has an average optimality gap of 0.26 ms.

The final average latency analysis is performed in the HCL scenario, and its results are shown in Fig. 10. The scenario with $\theta = 4$ was chosen because it shows most clearly the effects of traffic over average latency. The methods still follow a similar order: HBC and HCC have the same results, yielding the worst latencies among all methods (between 2.4 and 2.7 ms). Next, the benchmark yields much better results (between 1 and 1.4 ms), being the fourth best method. The third best, very close to the benchmark, is the MinMaxLat method (between 1 and 1.3 ms). The second best method is the heuristic (between 1 and 0.9 ms), with a small optimality gap (an average of 0.07 ms w.r.t. MinMeanLat). The optimal method is the MinMeanLat formulation (0.85 ms). The general trend in most methods, most notably in the benchmark and the heuristic, is to have average latency rise with higher traffic. This is a consequence of the link capacity limits: since HCL imposes very strict limits on link capacity, higher demands quickly fill up the links of the infrastructure. Therefore, each routing algorithm is forced to find alternative paths, which normally have a higher

latency, and thus, overall latency rises as a consequence. This behaviour is not exhibited by MinMeanLat or MinMaxLat, precisely because these methods do not have a conventional routing algorithm, and instead, route traffic based on MILP. Thus, it consistently chooses routes that minimize the overall or maximum latency, as opposed to using routing algorithms based on k-shortest path. Moreover, we find that HBC and HCC are unable to find a solution for traffic matrix 5, precisely because their routing decisions in the initial steps leave no room for later flows to find alternative routes.

In order to analyze maximum latency, Fig. 11 depicts the maximum latencies among all flows the HCL scenario, with the objective to analyze the effects of θ . As it can be seen, the trend is the exact opposite: a higher θ is directly related with a lower maximum latency. Dissecting this analysis by methods, we find again HBC and HCC consistently yielding the worst results. Moreover, the trend for both methods is completely flat: adding more FNs does not imply a better maximum latency if they are used. This phenomenon, which appears also in LCL-unscaled, is related to how they manage FN capacity: since they have a FN with enough capacity to meet all traffic demands, they direct all the traffic towards said FN. Thus, it is irrelevant whether more FNs are placed in the infrastructure. The next method is the benchmark, which yields the same results as HBC/HCC for under 3 FNs. However, as more FNs are added, the maximum latencies achieved by the benchmark decrease, until reaching an optimal result in $\theta = 7$. Overall, the gap between the benchmark and the methods with smaller latencies is very large unless a very high number of FNs is placed, and thus, obtaining good maximum latencies with it can be very costly. The heuristic is next, with an average optimality gap of only 1 ms. In general, the heuristic tends to closely follow the optimal solutions, and, despite needing a high number of FNs to become optimal, this number is significantly smaller than the benchmark. Finally, once again, both formulations yield optimal maximum latencies. Furthermore, there is an interesting phenomenon in these methods: there is a cap at 2 ms. After reaching this cap with 3 FNs, not even duplicating their number will decrease the maximum latency. This appears because FNs cannot be placed infinitely closer to hosts, and thus, once all FNs are placed very close to their assigned hosts, maximum latencies are minimal, and more FNs will not decrease it further. In

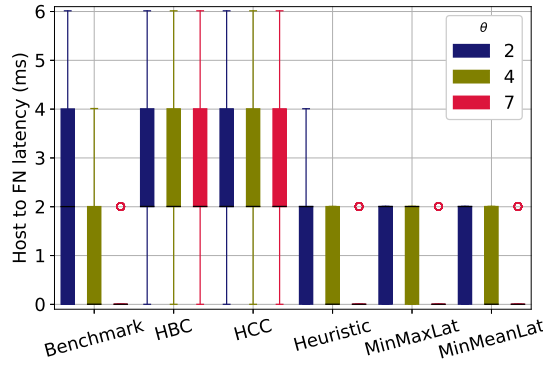


Figure 12: Latency box plot for LCL-unscaled.

conclusion, placing more FNs tends to lead to lower maximum latencies, until a certain limit is achieved.

To add to these analyses, a box plot of latency for the LCL-unscaled scenario is depicted in Fig. 12. With this box plot, it is possible to have additional information about each method's distribution on latency. The first analysis considers $\theta = 2$. Starting with the benchmark, the box plot shows that, although the benchmark achieves good results, they have a large spread. The Q1 is very low, starting nearly at 0 ms, which implies that 25% of the hosts or more have near-zero latency. However, its Q3 is nearly 4 ms, which implies its IQR is 4 ms wide. Thus, a 75% of the hosts have latencies that range from almost zero latency to 4 ms, which is a very high spread considering how it compares to other methods. Next, HBC and HCC have very similar results and spreads: their Q1 is close to 2 ms while their Q3 is close to 4 ms, hence having a smaller spread at the cost of overall higher latencies. The heuristic shares its Q1 with the benchmark, and its spread with HBC and HCC: its Q3 is at 2 ms, and thus, 75% of the hosts have under 2 ms latencies. However, its main difference with the MILP solutions, which the heuristic shares its IQR with, are the whiskers. Approximately, the remaining 25% of the hosts have latencies ranging between 2 and 4 ms with the heuristic, hence the upper whisker. In MinMaxLat and MinMeanLat, however, all hosts have under 2 ms of latency. A final remark that should be considered are medians: all of the methods share a 2 ms median, meaning that 50% of the hosts will always have under 2 ms of latency, regardless of the method. Nonetheless, depending on the method, the other 50% will experience higher or lower latencies: with the heuristic and MILP solutions, the other 50% will experience similar latencies (i.e., the median and Q3 are very close), with HBC and HCC, they will be significantly higher (i.e., the median and Q1 are very close), and with the benchmark, they will be higher, although lower than with HBC and HCC. Moving on to $\theta = 4$, we see that most methods simply enhance their behaviour: the benchmark has a very similar behaviour to the heuristic's in $\theta = 2$, although with a lower median. Similarly, the heuristic behaves like MinMaxLat and MinMeanLat, with a lower median as well. HBC and HCC do not improve, and behave equally throughout all θ values. Finally, while MinMaxLat behaves similarly with $\theta = 2$ and $\theta = 4$, we find that MinMeanLat is able to have a lower median. This

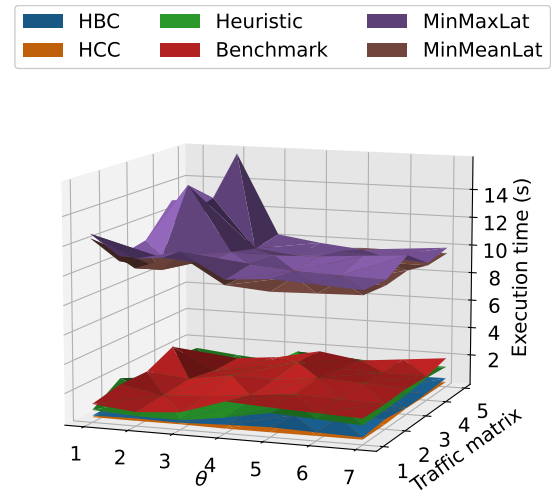


Figure 13: Execution times for LCL-unscaled.

behaviour responds to the optimization objective: MinMaxLat does not care about how many flows have a higher latency, as long as the overall highest one is minimal. MinMeanLat, on the other hand, prefers to have a higher number of flows with minimal latency. Finally, in $\theta = 7$, we find that all methods except HBC and HCC behave very similarly: latency is minimized, and both the box and whiskers are near 0. Nonetheless, there is a clear outlier at 2 ms in all cases, which represents the limit in which maximum latency cannot be further minimized that Fig. 11 also shows.

The next analysis is related to the execution times needed to place FNs, assign them to hosts and route the traffic, both for comparing all of the methods and for showing the effects of θ and traffic over them. Since these trends are followed in all scenarios, LCL-unscaled is used as a benchmark, with its results being depicted in Fig. 13. Comparing methods, we find that HCC is the fastest method, followed by HBC. The heuristic is next, significantly slower than both of them, but not exceedingly so. The benchmark is the third slowest method, although it can still be considered very fast. Finally, both MILP formulations take the most time to optimize the FN deployment, with a gap of approximately 500% w.r.t. the previous four methods. Out of the two, MinMeanLat is faster than MinMaxLat in almost every case. The effects of traffic, as well as θ , depend on the exact method: HBC and HCC are almost unaffected by these two parameters. The heuristic and benchmark, however, are slightly affected by traffic: more loaded matrices, such as 4 or 5, take more time to optimize than lightweight matrices, such as 1. θ has a more significant effect, as placing more FNs is more time-consuming in both methods. Finally, the MILP formulations do not show a general trend w.r.t. each of the parameters, and instead, have difficulties on a case-by-case basis. MinMeanLat only has a significant peak on the case with $\theta = 1$ and traffic matrix 1, staying stable throughout the rest of the cases. MinMaxLat shares this peak, while also showing significantly higher execution times in $\theta = 1$ with traffic matrix 2, as well

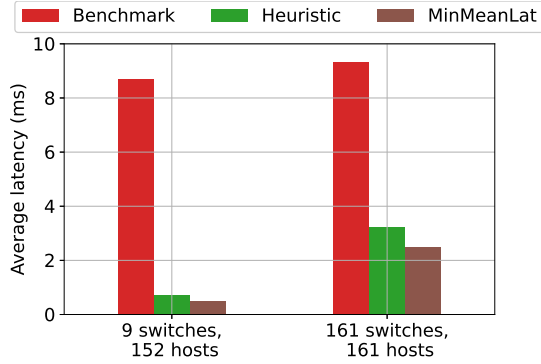


Figure 14: Average latencies in different topology sizes, $\theta = 7$.

as $\theta = 2$ with traffic matrices 3, 4 and 5. The conclusion from this analysis is that, out of all the methods, the most efficient one is the heuristic: it is consistently the second to third best placement method, as well as the third fastest one. The benchmark, however, obtains slightly worse solutions in a slightly higher time. Finally, HBC and HCC, despite fast, achieve the worst results overall. The opposite can be said about MinMaxLat and MinMeanLat, which, although being the best placement methods, they are also the slowest. Nonetheless, the FNPP is a design-time problem and, as such, even the highest times, such as the 14 seconds required by MinMeanLat and MinMaxLat, can be considered tractable times. However, in larger topologies, it is important to consider the scalability of the solution, as seen in the SDN deployment analysis and Fig. 8.

The final analysis assesses the impact of topology size and placement complexity in the average IoT to FN latency. In order to do so, the IIoT topology has been changed by not converting any of the original switches into hosts, and rather connecting a host to each switch. The result is a topology with 161 switches and 161 hosts, which is compared with the original IIoT deployment topology, featuring 9 switches and 152 hosts, by placing 7 FNs. Furthermore, in order to quantitatively compare our solution to the benchmark, cloud latency is shown in the cases the benchmark requires for cloud deployments. The results of this analysis are depicted in Fig. 14. It is important to note that the change in size makes the FN placement decisions much more complex, since $\binom{161}{7} = 487,444,845,680$ possible combinations for FN placement exist, rather than the original $\binom{9}{7} = 36$. Furthermore, the IoT to FN mapping maintains a very high complexity, with 7^{161} possibilities instead of 7^{152} . Only three methods are shown: MinMeanLat, the heuristic, and the benchmark, because the objective metric is average latency. As previously seen, MinMeanLat yields optimal latencies, followed by the heuristic, and with the benchmark in third place. The heuristic scales well, increasing its latency by a factor of 3.52 while the topology itself is 16.8 times larger, with 10^{10} times more possible placements and 10^8 times more possible assignments. In the case of the benchmark, we find that some of the IoT devices had to be assigned to the cloud, increasing the average latency to 8.7 ms in the original topology and 9.33 ms in the larger one. This makes the optimality gaps become very

large, at 8.21 ms (1,677.7%) in the original topology and 6.84 ms (374.8%) in the larger one. Nonetheless, the cloud latencies remain mostly stable in both topologies, which results in stable, although high, average latencies. In conclusion, larger topologies have a clear effect on the average latency experienced. In the present experiment, the average latency rises by between 2 and 3 ms, an amount relevant for very time-strict services such as IIoT factory automation [1]. Moreover, the complexity of FN placement and assignment can heavily affect latency, as the benchmark shows, requiring for cloud assignments and, thus, up to 16 times higher latencies.

VI. RELATED WORK

In order to meet the requirements of intensive IoT applications, latency is a crucial QoS dimension [1], and thus, it is key to optimize it. The optimization of QoS through the placement of equipment, especially in SDN and fog infrastructures, is a research topic that is still currently active and tackled from multiple points of view [11], [12], [14], [21], [22]. In this section, we review some of the related endeavors for latency and QoS optimization in SDN and fog environments.

On the one hand, we find that a component of latency in SDN networks is *control latency*: since the control plane is centralized in the figure of the SDN controller, SDN equipment must communicate with the controller in order to perform their tasks accordingly, and said communications have a certain latency. Control latency depends on the placement of the SDN controller or controllers relative to the SDN switches. Hence, the problem for the optimization of control latency by placing the SDN controller accordingly is known in research as the SDN Controller Placement Problem [14] (CPP). The CPP focuses on finding which SDN switches in the network are the best to host a SDN controller as well, and therefore, the CPP has a similar structure to the FNPP. Nonetheless, their focus is different: the CPP optimizes control latency and affects all the traffic of the SDN network [14], while the FNPP optimizes the latency of the IoT application by also allowing the offloading of computing tasks, only affecting the traffic directed towards the application. Furthermore, the CPP is a generalized problem in SDN networks, while the FNPP is a specific problem of SDN-fog scenarios. Another placement problem in IoT networks, and more concretely in wireless sensor networks, is the placement of base stations [21]. This problem consists on optimally placing a set of wireless base stations in a given area, maximizing metrics such as the coverage area of the wireless network or the network lifetime. Nonetheless, the station placement problem and the FNPP are different. To summarize the differences, the objective of wireless station placement is to optimally locate a set of wireless base stations at arbitrary points in a specified area to carry the information from the sensors to an *information sink* [21], while the role of the FNPP would precisely be to locate the information sinks in specific points (i.e., SDN switches).

On the other hand, the optimization of latency in IoT applications deployed on fog scenarios is often approached from the application point of view. Modern IoT applications are often divided into multiple services that may be deployed

on different machines, including fog and cloud nodes [22]. Since these services often need to interact with each other, and can be requested by different IoT devices in different locations, it is important to deploy the application services in a manner that minimizes the overall IoT application latency. As Brogi *et al.* surveyed in [22], the endeavor for finding optimal deployments is an active research topic on its own. While this line of research has the same objective, minimizing the latency of intensive IoT applications, the optimization efforts are meant to distribute software components in computing devices, a fundamentally different approach from the FNPP, which is meant to place the computing devices themselves.

Finally, other authors have also tackled latency optimization from the same perspective. These works share the core idea with the FNPP: to minimize the application latency by placing FNs optimally within the scenario. One of the most recent proposals is the *fog network planning problem* [12], which is also based on the idea of strategically placing FNs to reduce latency. The fog network planning problem differs from the FNPP's premises: rather than fog-enabling a SDN network, Gilbert *et al.* try and place a set of FNs in different *areas*, i.e., geographical clusters of IoT devices. The fog network planning problem includes FN placement and FN-IoT device assignment, but lacks routing considerations, including the selection of communication technologies between FNs instead. Another interesting research line is the one of Maiti *et al.*, who present FN placement as a relevant problem for IoT applications [11], [23], [24]. This set of works solves the problem of placing a given number of FNs in a multi-tiered SDN-fog network infrastructure. Nonetheless, there are important differences between the FNPP and this research line: these works focus on optimal FN placement in tree-shaped topologies, unlike the FNPP, which considers arbitrarily-shaped topologies. Moreover, the proposals within this research line do not consider IoT device to FN assignment or traffic routing because they are not required in tree-shaped network topologies. Thus, the main difference between these works and the FNPP is that they focus exclusively in FN placement, whereas the FNPP includes assignment and routing considerations.

In conclusion, the FNPP is closely related to the research topic of latency optimization in networks, although its focus on the placement of computing equipment in a network topology makes it conceptually different from the CPP, wireless base station placement, or service placement. Furthermore, unlike the most similar works, the FNPP supports arbitrarily-shaped network topologies, and considers the IoT devices to FNs assignment, as well as the routing of the traffic between them.

VII. CONCLUSIONS AND FUTURE WORK

The growth and development of the IoT paradigm has generated new possibilities of real world process automation and management in intensive domains. Nonetheless, integrating time-strict real world processes with IoT technology calls for time-strict IoT applications, which complicate meeting their QoS requirements in a cloud environment. While a combined fog-SDN infrastructure eases the achievement of the QoS

objectives, the placement of FNs in the infrastructure plays a key role on QoS. In this paper, we presented the FNPP: the problem of placing FNs optimally in a fog-SDN infrastructure. We have also developed three solutions for the FNPP, two of which (heuristic and MinMaxLat) are completely novel. Moreover, we have evaluated these solutions in both Internet topologies and IIoT fog environments, comparing them to alternatives such as HBC or HCC, as well as state-of-the-art benchmarks [11]. Our heuristic solution provides near-optimal results, with optimality gaps under 1 ms and nearly the same latency distributions, and finds fog-only deployments with more ease than the benchmark and similar scalability. On the other hand, our MILP-based solutions provide optimal results in tractable time for small topologies.

In the future, we expect to address other QoS objectives, such as reliability or resilience, in the FNPP, allowing for optimal placements in terms of their fault tolerance. Moreover, we also expect to create multi-objective solutions for the FNPP, able to target multiple QoS objectives at the same time. In this field, we expect to create both MILP-based, multi-objective solutions, as well as genetic algorithms able to yield a Pareto front. Furthermore, the FN scheduling problem, based around the migration of the containers or virtual machines used by FNs at execution time to optimally place them, is also a key future work. We expect to develop solutions for this problem, enabling for FN migration in real time.

REFERENCES

- [1] H. Xu, W. Yu, D. Griffith, and N. Golmie, "A Survey on Industrial Internet of Things: A Cyber-Physical Systems Perspective," *IEEE Access*, vol. 6, pp. 78 238–78 259, 2018.
- [2] J. Singh, T. Pasquier, J. Bacon, H. Ko, and D. Eysers, "Twenty security considerations for cloud-supported internet of things," *IEEE Internet of things Journal*, vol. 3, no. 3, pp. 269–284, 2015.
- [3] P. Bellavista, J. Berrocal, A. Corradi, S. K. Das, L. Foschini, and A. Zanni, "A survey on fog computing for the Internet of Things," *Pervasive and Mobile Computing*, vol. 52, pp. 71 – 99, 2019.
- [4] M. Aazam, S. Zeadally, and K. A. Harras, "Deploying fog computing in industrial internet of things and industry 4.0," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4674–4682, 2018.
- [5] J. Galán-Jiménez, J. Berrocal, J. L. Herrera, and M. Polverini, "Multi-objective genetic algorithm for the joint optimization of energy efficiency and rule reduction in software-defined networks," in *2020 11th International Conference on Network of the Future (NoF)*, 2020, pp. 33–37.
- [6] J. Wan, S. Tang, Z. Shu, D. Li, S. Wang, M. Imran, and A. V. Vasilakos, "Software-defined industrial internet of things in the context of industry 4.0," *IEEE Sensors Journal*, vol. 16, no. 20, pp. 7373–7380, 2016.
- [7] I. Bedhief, L. Foschini, P. Bellavista, M. Kassab, and T. Aguilu, "Toward self-adaptive software defined fog networking architecture for IIoT and industry 4.0," in *Proceedings of IEEE CAMAD 2019*, 2019.
- [8] J. Galán-Jiménez, "Legacy ip-upgraded sdn nodes tradeoff in energy-efficient hybrid ip/sdn networks," *Computer Communications*, vol. 114, pp. 106–123, 2017.
- [9] J. L. Herrera, L. Foschini, J. Galán-Jiménez, and J. Berrocal, "The service node placement problem in software-defined fog networks," in *IEEE Symposium on Computers and Communications*, 2020, pp. 1–6.
- [10] J. L. Herrera, P. Bellavista, L. Foschini, J. Galán-Jiménez, J. M. Morillo, and J. Berrocal, "Meeting stringent qos requirements in iiot-based scenarios," in *IEEE Global Communications Conference*, 2020, pp. 1–6.
- [11] P. Maiti, J. Shukla, B. Sahoo, and A. K. Turuk, "QoS-aware fog nodes placement," in *Proc. 4th IEEE Int. Conf. Recent Adv. Inf. Technol. RAIT 2018*, jun 2018, pp. 1–6.
- [12] G. M. Gilbert, N. Shililiandumi, and H. Kimaro, "Evolutionary approaches to fog node placement in lv distribution networks," *International Journal of Smart Grid*, vol. 5, no. 1, pp. 1–14, 2021.
- [13] S. Melkote and M. S. Daskin, "Capacitated facility location/network design problems," *European journal of operational research*, vol. 129, no. 3, pp. 481–495, 2001.

- [14] T. Das, V. Sridharan, and M. Gurusamy, "A Survey on Controller Placement in SDN," *IEEE Communications Surveys & Tutorials*, pp. 472–503, 2019.
- [15] J. T. Linderöth and A. Lodi, "Milp software," *Wiley encyclopedia of operations research and management science*, vol. 5, pp. 3239–3248, 2010.
- [16] H.-S. Park and C.-H. Jun, "A simple and fast algorithm for k-medoids clustering," *Expert systems with applications*, vol. 36, no. 2, pp. 3336–3341, 2009.
- [17] M. Barbehenn, "A note on the complexity of dijkstra's algorithm for graphs with weighted vertices," *IEEE transactions on computers*, vol. 47, no. 2, p. 263, 1998.
- [18] N. Auger, V. Jugé, C. Nicaud, and C. Pivoteau, "On the worst-case complexity of timsort," *arXiv preprint arXiv:1805.08612*, 2018.
- [19] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly, "SNDlib 1.0—Survivable Network Design Library," in *Proceedings of INOC 2007*, apr 2007.
- [20] Boeing, "Boeing: Future of Flight Aviation Center & Boeing Tour - Background Information," 2013. [Online]. Available: <https://web.archive.org/web/20130313010544/http://www.boeing.com/commercial/tours/background.html>
- [21] Y. Shi and Y. T. Hou, "Optimal base station placement in wireless sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 5, no. 4, pp. 1–24, 2009.
- [22] A. Brogi, S. Forti, C. Guerrero, and I. Lera, "How to place your apps in the fog: State of the art and open challenges," *Software: Practice and Experience*, vol. 50, no. 5, pp. 719–740, 2020.
- [23] P. Maiti, H. K. Apat, B. Sahoo, and A. K. Turuk, "An effective approach of latency-aware fog smart gateways deployment for iot services," *Internet of Things*, vol. 8, p. 100091, 2019.
- [24] P. Maiti, H. K. Apat, A. Kumar, B. Sahoo, and A. K. Turuk, "Deployment of multi-tier fog computing system for iot services in smart city," in *2019 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*. IEEE, 2019, pp. 1–6.



Luca Foschini (Senior Member, IEEE) received a Ph.D. degree in computer science engineering from the University of Bologna in 2007, where he is an Associate Professor of computer engineering. His interests span from integrated management of distributed systems and services to wireless pervasive computing and scalable context data distribution infrastructures and context-aware services.



Paolo Bellavista (Senior Member, IEEE) received a Ph.D. in computer science engineering from the University of Bologna, Italy, where he is now a full professor of distributed and mobile systems. His research activities span from pervasive wireless computing to edge cloud computing or middleware for Industry 4.0 applications.



Juan Luis Herrera received a Bachelor's degree in software engineering from the University of Extremadura in 2019. He is a researcher in the Computer Science and Communications Engineering Department of the University of Extremadura. His main research interests include the IoT, fog computing and SDNs.



Javier Berrocal (IEEE Member) is a co-founder of Gloin. His main research interests are software architectures, mobile computing, and edge and fog computing. Berrocal has a Ph.D. in computer science from the University of Extremadura, where he is currently an Associate Professor.



Jaime Galán-Jiménez received a Ph.D. in computer science and communications from the University of Extremadura in 2014, where he is now an Assistant Professor. His main research interests are SDNs, 5G network planning and design, and mobile ad hoc networks.



Juan M. Murillo (IEEE Member) is a co-founder of Gloin and a Full Professor at the University of Extremadura. His research interests include software architectures, mobile computing, and cloud computing.