Optimizing the Response Time in SDN-Fog Environments for Time-Strict IoT Applications

Optimizing the Response Time in SDN-Fog Environments for Time-Strict IoT Applications

Juan Luis Herrera, Jaime Galán-Jiménez, Javier Berrocal, *Member, IEEE*, and Juan Manuel Murillo, *Member, IEEE*

***Abstract*—This work has been published: https://doi.org/10.1109/JIOT.2021.3077992 The Internet of Things (IoT) paradigm offers applications the potential of automating real-world processes. Applying IoT to intensive domains comes with strict quality of service (QoS) requirements, such as very short response times. To achieve these goals, the first option is to distribute the computational workload throughout the infrastructure (edge, fog, cloud). In addition, integration of the infrastructure with enablers such as software-defined networks (SDNs) can further improve the QoS experience, thanks to the global network view of the SDN controller and the execution of optimization algorithms. Therefore, the best placement for both the computation elements and the SDN controllers must be identified to achieve the best QoS. While it is possible to optimize the computing and networking dimensions separately, this results in a suboptimal solution. Thus, it is crucial to solve the problem in a single effort. In this work, the influence of both dimensions on the response time is analyzed in fog computing environments powered by SDNs. DADO, a framework to identify the optimal deployment for distributed applications is proposed and implemented through the application of mixed integer linear programming. An evaluation of an IIoT case study shows that our proposed framework achieves scalable deployments over topologies of different sizes and growing user bases. In fact, the achieved response times are up to 37.89% lower than those of alternative solutions and up to 15.42% shorter than those of state-of-the-art benchmarks.**

***Index Terms*—Fog computing, edge computing, Internet of Things (IoT), software-defined network (SDN)**

I. Introduction

THE POPULARITY of IoT devices for the general public has made them ubiquitous. We are surrounded by everyday objects (*things*) that are connected to the Internet and run numerous IoT applications – programs that can interact with the real world through IoT devices, obtain inputs from their sensors and change the real world through their actuators. This scenario makes IoT applications interesting for different domains, both general-purpose domains, such as domotics, and intensive domains, such as industry or healthcare.

The cloud computing paradigm is the most common option for running these IoT applications [1]. In cloud computing, a set of powerful servers, generally far away from IoT devices, carry out the processing, while IoT devices need only to send their requests to these servers. While a cloud-centric architecture is enough for user-grade, general-purpose applications [2], a purely cloud-centric architecture may not be enough to meet the QoS requirements of complex and intensive applications. Industrial Internet of Things (IIoT) applications such as factory automation need very low response times [3], while Internet of Medical Things (IoMT) applications can be very time sensitive in executing artificial intelligence models [4]. Therefore, the objective of this paper is to minimize the response time of intensive IoT applications such as IIoT or IoMT applications. Cloud computing servers are often far away from IoT devices and thus may have latencies that may complicate the process of obtaining a sufficiently short response time.

For this reason, other paradigms such as fog computing or mist computing, which can be referred to under the umbrella term *fog computing* [2], are emerging to support applications with strict response-time requirements [2]. Fog computing takes advantage of the computational capabilities of nodes closer to end devices, as well as nodes of the devices themselves, by executing different parts of the IoT applications in them. This approach makes it is easier to achieve shorter response times than those possible by using pure cloud computing infrastructures. Therefore, in fog computing environments, which nodes to use and which node should host which parts of an application are decisions that must be made and affect the provided response time [5]. The problem of making optimal decisions, and thus distributing computations optimally, is known as the decentralized computation distribution problem (DCDP) [6].

A key element of the DCDP, and one of the main motivations behind fog computing paradigms, is network latency [5]: the latency from IoT devices to nearby fog devices is smaller than that to the cloud. Hence, the execution time can be shortened, which means that minimizing network latency, through techniques such as routing optimization [7], is key

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), and by the European Regional Development Fund. *(Corresponding author: Juan Luis Herrera.)*

The authors are with the Department of Computer Science and Communications Engineering, University of Extremadura, Spain (e-mail: jlherrerag@unex.es).

Digital Object Identifier 00.000/JIOT.0000.0000000

for correctly solving the DCDP. Software-defined networking, a paradigm that allows networks to be programmed through SDN controllers, allows for programmable routing optimization [8], thus making this latency optimization scalable and flexible as new devices are added to the infrastructure. In fact, some proposals make use of multiple coordinated SDN controllers in fog IoT infrastructures to improve the QoS of the overall infrastructure [9]. Software-defined networking allows network controllers to be notified when new devices are added through discovery protocols [8], and it allows network controllers to monitor networking and computing devices [8], gathering performance information from the infrastructure, which can be used to solve the DCDP.

While the control that SDNs provide allows for the latency between computing devices to be optimized to a certain extent, the SDN control latency can also be optimized. SDNs rely on controllers for operation; therefore, the latency between SDN switches and controllers affects the latency between any two devices in the SDN [10]. This implies that if controllers are placed in a way that minimizes this latency, the latency between devices in the network will be minimized as well. The problem of placing the controller optimally is known in research works as the SDN controller placement problem (CPP) [10].

Several authors have studied both the DCDP and the CPP as separate problems [6], [10]–[18], providing partial solutions that consider only one dimension. The DCDP solutions assume that the network is a static entity that provides a certain latency, while the CPP solutions assume that the traffic flows do not change based on the latency achieved by the network. However, solving each of these problems separately may not result in sufficient response time optimization for IoT applications with strict response-time requirements. The decision of assigning a service to a node in a certain layer should take into account the optimal network latency, which depends on the controller placement; additionally, the decision of where to place the controller should consider the steering of traffic flows throughout the network, which depends on which nodes are requesting services and which nodes are providing them. Therefore, these dimensions affect each other. To fully optimize the response time of IIoT, IoMT and other intensive IoT applications, both problems should be solved together so that their mutual influences and trade-offs can be taken into account when merging them into a single new problem. We define the resulting combined problem as the Fog-SDN deployment problem. To the best of our knowledge, no prior work has addressed this problem. The main contributions of this work are as follows:

- A study of the relationship between the DCDP and the CPP in intensive IoT environments with the objective of better understanding the trade-offs between them. Moreover, further motivation regarding the use of SDNs for service discovery is provided.
- The formalization of optimal microservice deployment and controller placement (i.e., a combination of the DCDP and CPP) in a single effort, with the response time as the objective.
- The proposal of a distributed application deployment optimization (DADO) framework to contribute to the solving of the Fog-SDN deployment problem. The DADO framework includes an implementation based on mixed integer linear programming (MILP) that makes it suitable for design time optimization. Moreover, its combination of the DCDP and CPP in a single optimization solution is novel compared to state-of-the-art optimization solutions.
- The experimental evaluation of DADO in an IIoT scenario, focused on the scalability of the solution, the trade-off between latency and the response time, and the ability to limit the tolerable delay.

This paper is structured as follows: Sec. II offers the motivation for combining the CPP and the DCDP into the Fog-SDN deployment problem through an illustrative IIoT example. Sec. III proposes DADO, a solution to the Fog-SDN deployment problem. Sec. IV evaluates DADO using a setup based on the previously presented example. Sec. V presents related work. Finally, Sec. VI concludes the paper and highlights future challenges.

II. Background

To illustrate the importance of combining the CPP and the DCDP, a particular case study scenario is presented in this section.

A. Scenario: Fog IIoT factory

The scenario presented in this section is based on the environment proposed in [12] since this work provides not only an IIoT-based fog computing scenario but also enough details about the infrastructure with which to apply and evaluate our solution, DADO. In this scenario, a fog infrastructure based on an SDN is deployed in a factory to transform it into a cyber-physical smart factory by leveraging the IIoT [3]. An IIoT device is installed in each robot; initially, only 10 robots are part of the smart factory, but this system is expected to grow over time if the company decides to invest further into the transformation. Ten fog servers are placed in the same factory to provide the IIoT devices with services. Each fog server has a 800 MHz CPU and 1 GB of RAM [12] and is directly connected to an SDN switch.

In this factory, an SDN is leveraged to provide service discovery due to its properties as a modular, independent and transparent solution [8]. In fact, an SDN is the quickest mechanism for implementing service discovery by leveraging overlay networks [19]. Because SDNs need at least one controller, the factory uses the classic SDN control model and co-locates the controllers with SDN switches [10]. Fig. 1 shows this cyber-physical IIoT system [3], divided into three layers. The physical layer embodies the *physical* part of the system, while the *cyber* part is divided into two layers: the networking layer, which contains the SDN switches and controllers, and the computing layer, which contains the IIoT devices and fog servers.

In this infrastructure, an IIoT application is to be deployed to monitor and manage the smart factory continuously by gathering the statuses of the robots through the sensors connected to the IIoT devices and processing them in fog

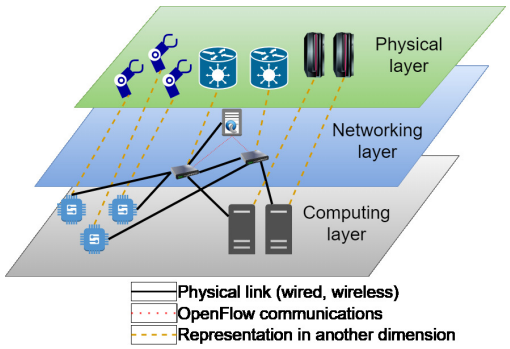


Figure 1: The presented scenario with the three involved layers.

servers to provide commands accordingly. This application was designed using a microservice architecture (MSA) and therefore comprises different independent services that perform a certain type of processing, the functions of which can be requested separately or can be combined through workflows. Each microservice takes 100, 500 or 1000 MCycles to run [12] depending on its computational complexity. Each workflow comprises the execution of a certain functionality, chaining between one and six microservices depending on the number required to perform the functionality. The response time is a combination of the execution time (i.e., the time it takes to execute the microservices of a functionality) and latency (i.e., the time it takes for one computer to communicate to another in a network).

Fig. 2 shows how the locations of the microservices and SDN controllers influence the response time: a sample workflow is shown, in which a command request functionality is performed by chaining a microservice that aggregates information from multiple sensors (M1) with another microservice that analyzes the aggregated information to issue a corresponding command (M5). In this figure, the box with the OpenFlow logo represents the SDN controller, the dashed lines represent SDN control messages and the solid lines represent application messages. In Fig. 2a, M1 is located in fog server S1, and M5 is located in fog server S2. When the leftmost IIoT requests a command, its message is sent to the SDN, with M1 as the destination address. The SDN switch is unaware of M1's location; thus, it asks the SDN controller where M1 can be found through an OpenFlow packet-in message to perform service discovery. The SDN controller answers with a packet-out message, and the switch routes the message to S1. Once the data are aggregated on S1, the message requesting M5 is sent to the network to be analyzed. Again, the SDN switch must request service discovery to find M5. After the packet-out message arrives, the message is routed to S2. Finally, S2 issues the command and sends it back to the IIoT device through the network. In this exchange, messages must be sent 23 times through the network, with each new transmission increasing the delay due to latency. Fig. 2b shows a different strategy, in which the IIoT device itself aggregates the information and the SDN controller is placed closer. Because of these changes, the IIoT device needs only to request M5. Because the SDN controller is placed on the same SDN switch as that to which the message is sent, service discovery is performed locally with minimal latency. S1 analyzes the aggregated information and sends back the command as previously. This new strategy changes the number of message transmissions to four, thus reducing the overall execution time. Therefore, two decisions must be made to deploy this application optimally: i) in which host to execute each microservice in the architecture and ii) in which switch or switches to place the SDN controllers.

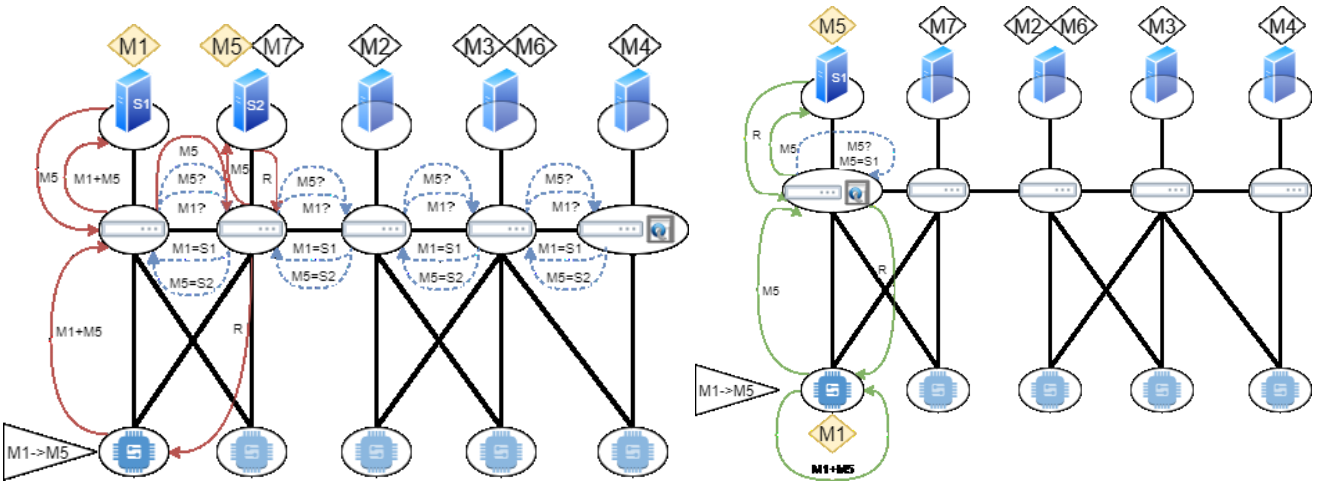
B. DCDP: Placing the microservices

The solution to the DCDP involves making the first decision: selecting which microservices should run in the IIoT devices themselves and which microservices should run in the fog server. As shown in Fig. 2, while IIoT devices are not as powerful as fog servers, executing some microservices in them – especially those that are lightweight and requested very often – allows parts of the workflows to be executed locally, thus completely ignoring the network latency. The difference is in how much faster these services can be executed when they are placed in the fog and how large the latency between the IIoT devices and fog servers is. If the latency to the fog servers is larger than the difference in execution time, then it is worthwhile to execute the microservices locally; however, if the latency is smaller than this difference, then the response time will be shorter if they run on fog servers. Therefore, the choices for solving the DCDP are inherently related to network latency. Thus, to optimize the response time through the DCDP, the network latency should be optimized first.

C. CPP: Placing the SDN controllers

Solving the CPP is related to the other decision in the scenario: where to place the SDN controllers. The placement of the SDN controller plays a key role in control latency and, by extension, is related to the overall network latency [10]. Controller placement is strongly related to the network topology but also to how traffic flows are steered through the network [18]: while topology-wise, a node may seem optimal for controller placement, it may not be optimal if that network zone is not frequently used. This case is shown in Fig. 2b, in which the controller is placed on the leftmost part of the network because traffic flows are steered through that zone.

In this scenario, traffic is generated by the IIoT application when a workflow cannot be executed fully locally; in these cases, one host sends a message with the input data of the microservice to the host that executes it. Once the microservice is executed, the output data of the microservice are sent back. The execution of M5 in Fig. 2b is a graphical example of this. These messages generate two traffic flows: one to send the input data and another one to send the response. Thus, the decisions made while solving the DCDP may affect the CPP: the choice to execute a microservice locally removes traffic flows, which may make its area less suitable for controller placement, while the choice to execute a microservice remotely adds traffic flows, which may make that area more suitable for controller placement. Suboptimal decisions in solving the DCDP may lead to suboptimal decisions in solving the CPP. Thus, to optimize the response time through the CPP,



(a) Fog deployment, suboptimal controller placement.

(b) Mist and fog deployment, optimal controller placement.

Figure 2: Workflow execution with two different deployment strategies.

microservices should be placed in a way that optimizes the response time.

D. Computing and networking trade-off

Considering the response time as the QoS metric to be optimized in a deployment, such as in the scenario presented in Sec. II-A, involves considering the inherent relationship and trade-off between computing and networking. This relationship partially comes from the characteristics of scenarios in which the network is used to transmit computing-related data. To optimize the performance, the network can be optimized through the CPP, i.e., transmitting information with minimal latency guarantees that computing can occur as soon as possible. However, if the destinations of this information are the slower hosts of the infrastructure, then the response time will be slowed down by the computation being performed on those hosts and, thus, still will not be optimal. Another approach for response time optimization could be to optimize how computing is divided and carried out through the DCDP. However, the higher latency of an unoptimized network would again slow down the response time. Therefore, treating each of these layers separately can lead to suboptimal solutions due to the lack of a complete view of the infrastructure environment and context. As Sec. II-B and Sec. II-C show, the solution of the CPP is required to solve the DCDP, and vice versa.

Thus, the relationship and trade-off between computing and networking generate a bootstrapping problem: to optimize the network layer through the CPP, the computing layer should be optimized first through the DCDP, but to optimize the computing layer through the DCDP, the network should be optimized first through the CPP. Considering them separately implies possibly making suboptimal decisions because of these behavior differences, such as those shown in Fig. 2a. To avoid potentially obtaining a suboptimal response time for the IIoT application, both computing and networking should be optimized at the same time to avoid this bootstrapping problem.

III. DADO FRAMEWORK

In this section, the DADO framework is described. It is aimed at mitigating the issues raised by the Fog-SDN deployment problem. Taking as input statistics related to the computational load of microservices, the computational power of fog or cloud servers, the latency and capacity of links, the workflows used by the requested functionalities, and the network topology, DADO is able to provide as output information on the placement of SDN controllers, the placement of microservices at hosts and the paths taken by the traffic flows. Moreover, this infrastructure description can be obtained by leveraging the SDN [8]. Through a combination of all these metrics, the optimal response time is provided. DADO uses at its core a mixed integer linear programming (MILP) formulation, as presented in the following.

DADO is aimed at optimizing the response time in fog architectures, as well as in hybrid ones, for IoT applications based on an MSA. Thus, the infrastructure will contain SDN switches, IoT devices, fog servers and cloud nodes. IoT devices, as well as fog and cloud servers, can execute parts of the logic of the application, generate traffic and consume this traffic – they are *hosts*. SDN switches have a completely different behavior, forwarding traffic by applying routes calculated by their assigned SDN controller. Therefore, let the infrastructure be represented as a graph $G = \{V, L\}$. Let H be a set of hosts and S be a set of SDN switches so that the set of vertices $V = H \cup S$. Let L be the links that connect the different elements of the infrastructure.

Not all of these hosts have the same capabilities. Generally, fog servers are more powerful than IoT devices, with cloud servers being the most powerful. This power can be represented by the host's speed in executing microservices, as well as by the maximum amount of services the host can execute. Thus, let a host $h \in H$ be a tuple $h =< P_h, r_h >$, with P_h being the computational power of the host (measured as its clock speed in Hz) and r_h being the host's total RAM, measured in bytes. If other applications or services are running on the host, then P_h is the computational power of the host

not being used for other applications or services (i.e., it can be used by the IoT application) and r_h is the host's remaining free RAM.

In this infrastructure, the links have two essential limitations. First, the links affect latency since the transmission of data over them is not instantaneous. Second, the links do not have infinite capacity and therefore cannot be used to transfer an unlimited amount of data. Thus, let a link $l_{ij} \in L$, with i being the source of the link and j being its destination, be a tuple $l_{ij} =< \delta_{ij}, \theta_{ij} >$, with δ_{ij} being the link's latency in seconds and θ_{ij} being the link's maximum capacity in bytes per second. If the link is also being used to transmit data that are unrelated to the application being optimized, then θ_{ij} references only the capacity that is not in use by other traffic.

The IoT applications that DADO supports have an MSA and can be seen effectively as a set of independent microservices. We therefore have a set of microservices M, with each microservice $m \in M$ being a tuple $m =< \xi_m, I_m, O_m, r_m >$, with ξ_m as the workload of executing the microservice (measured as the number of CPU cycles the microservice requires to fully execute), I_m as the size of the input data for the microservice in bytes, O_m as the size of the microservice's output data in bytes and r_m as the amount of RAM the microservice requires in bytes.

The execution model of DADO is based on workflows. When an IoT device requests a certain functionality, this functionality is provided by a workflow of one or more microservices.

Where these microservices run depends on the solution DADO generates. If the first microservice, or two consecutive microservices, is run on the same host, then the network is not used. If the microservice does not run on that device, then the SDN is used to route the request to a host that will run the service. Let W be a set of workflows, with each workflow $w \in W$ being an ordered set of tuples $w = \{c_1, c_2, ..., c_{|w|}\}$. Each tuple c_i must have the exact same format and values as those of one of the microservices in M since each of these tuples represents the microservices that are chained through the workflow to perform a functionality. Therefore, with a slight abuse of notation, we can say that $w = \{c_1, c_2, ..., c_{|w|}\}; c_i \in M \forall i \in [1, |w|]$. Let also $WS(w, h)$ be a binary function equal to 1 if workflow w is started by host h and 0 otherwise. To execute the workflow, the data would have to flow from whatever host starts the workflow to a host that executes c_1, from there to the one that executes c_2, and so on. Moreover, let $\Delta_w; w \in W$ be the maximum tolerable delay of the workflow in seconds.

We also propose the usage of the classic SDN control model [10]. Each controller can be co-located with an SDN switch in the network $s \in S$ (i.e., the controller and the SDN switch are in the same place). Each SDN switch is *mapped* to a controller so that the switch communicates with the controller through in-band traffic. Thus, let ψ be the maximum number of SDN controllers to be placed, and let Ω be the size of the control packets sent from each SDN switch to the controller.

After these parameters have been defined, we must set different different decision variables, which must be changed to optimize the deployment.

In the computing plane, let z be a three-dimensional binary matrix, in which $z^w_{hc_a}$ is 1 if host h is running the microservice indexed as c_a of workflow w and 0 otherwise. This allows DADO to locate microservices in certain hosts. Let f be a five-dimensional binary matrix, in which $f^{hwc_a}_{ij}$ is 1 if the traffic host h generated as a consequence of the microservice indexed as c_a of workflow w is routed through the link l_{ij} and 0 otherwise. This allows DADO to route the input and intermediate output data of the microservices through the network. Let f' be a four-dimensional binary matrix, in which f'^{hw}_{ij} is 1 if the traffic host h generated in response to workflow w is routed through the link l_{ij} and 0 otherwise. This allows DADO to route the final output data of the microservices.

In the networking plane, let x be a binary vector in which x_s is 1 if an SDN controller is set up on switch s and 0 otherwise. This allows DADO to place SDN controllers. Let y be a binary matrix in which $y_{ss'}$ is 1 if SDN switch s is mapped to controller s' and 0 otherwise. This allows DADO to map SDN controllers and switches. Let cf be a three-dimensional binary matrix, in which cf^s_{ij} is 1 if the control traffic switch s generated is routed through the link l_{ij} and 0 otherwise. This allows DADO to route the control data between SDN switches and their mapped controllers.

We must also establish constraints that determine which values are allowed for each variable under different conditions and how changing these values affects the overall deployment. We first assume that a microservice for a certain workflow can be executed only in a single host.

$$\sum_{h \in H} z^w_{hc_a} = 1; \forall w \in W, a \in [1, |w|] \tag{1}$$

Then, each host cannot run unlimited microservices but only as many as its memory allows.

$$\sum_{w \in W} \sum_{a=1}^{|w|} z^w_{hc_a} r_{c_a} \leq r_h; \forall h \in H \tag{2}$$

It is impossible to have more controllers than the maximum amount.

$$\sum_{s \in S} x_s \leq \psi \tag{3}$$

A switch can be mapped to only one controller at a time.

$$\sum_{s' \in S} y_{ss'} = 1; \forall s \in S \tag{4}$$

In addition, it can be mapped to a controller only if that controller is actually placed.

$$y_{ss'} \leq x_{s'}; \forall s, s' \in S \tag{5}$$

Flow variables should be controlled in aggregate, according to the classic flow constraints. When we account for microservice data, we must consider three cases: the first microservice of the workflow (c_1), the response of the workflow and the general case for the rest. In the case of c_1, it can be stated that i) traffic is generated only by the host that starts the workflow, unless c_1 is mapped to the same host (in that case, it will be

executed locally), and that ii) traffic is consumed by the host that has c_1 mapped, unless it is the same host that starts the workflow. Formally:

$$\sum_{j\in V} f_{ij}^{hwc_1} - f_{ji}^{hwc_1} = \begin{cases} 0 & \text{if } i \in S \\ WS(w,h)(1-z_{ic_1}^w) & \text{if } i = h \\ -WS(w,h)z_{ic_1}^w & \text{otherwise.} \end{cases} \quad \forall i \in V, h \in H, w \in W \tag{6}$$

In the case of the response, we can state that i) traffic is generated only by the host with the last microservice mapped, unless it is mapped to the host that started the workflow, and that ii) traffic is consumed by the host that started the workflow, unless it has the last microservice mapped. Formally:

$$\sum_{j\in V} f_{ij}^{\prime hw} - f_{ji}^{\prime hw} = \begin{cases} 0 & \text{if } i \in S \\ z_{hc_{|w|}}^w(1-WS(w,i)) & \text{if } i = h \\ -z_{hc_{|w|}}^w WS(w,i) & \text{otherwise.} \end{cases} \quad \forall i \in V, h \in H, w \in W \tag{7}$$

We can derive a general case. Traffic is generated by the host that has the previous microservice mapped, as long as the current microservice is not mapped to that host, and it is consumed by the host that has the current microservice mapped, as long as that host does not have the previous microservice mapped.

$$\sum_{j\in V} f_{ij}^{hwc_a} - f_{ji}^{hwc_a} = \begin{cases} 0 & \text{if } i \in S \\ z_{hc_{a-1}}^w(1-z_{ic_a}^w) & \text{if } i = h \\ -z_{hc_{a-1}}^w z_{ic_a}^w & \text{otherwise.} \end{cases} \quad \forall i \in V, h \in H, w \in W, a \in [2,|w|] \tag{8}$$

This formulation contains a multiplication of possible decision variables, which would make the problem nonlinear. We can use some linearization techniques to solve this problem. For this purpose, we create the following new variables: $z_{hc_a}^{\prime iw} = z_{hc_{a-1}}^w(1-z_{ic_a}^w)$, $z_{hc_a}^{\prime\prime iw} = z_{hc_{a-1}}^w z_{ic_a}^w$. For them to have these values, they must follow these constraints:

$$-z_{hc_{a-1}}^w + z_{hc_a}^{\prime iw} \leq 0 \tag{9}$$

$$-1 + z_{ic_a}^w + z_{hc_a}^{\prime iw} \leq 0 \tag{10}$$

$$z_{hc_{a-1}}^w + 1 - z_{ic_a}^w - z_{hc_a}^{\prime iw} \leq 1 \tag{11}$$

$$-z_{hc_{a-1}}^w + z_{hc_a}^{\prime\prime iw} \leq 0 \tag{12}$$

$$-z_{ic_a}^w + z_{hc_a}^{\prime\prime iw} \leq 0 \tag{13}$$

$$z_{hc_{a-1}}^w + z_{ic_a}^w - z_{hc_a}^{\prime\prime iw} \leq 1 \tag{14}$$

(8) can now be rewritten as a linear constraint:

$$\sum_{j\in V} f_{ij}^{hwc_a} - f_{ji}^{hwc_a} = \begin{cases} 0 & \text{if } i \in S \\ z_{hc_a}^{\prime iw} & \text{if } i = h \\ -z_{hc_a}^{\prime\prime iw} & \text{otherwise.} \end{cases} \quad \forall i \in V, h \in H, w \in W, a \in [2,|w|] \tag{15}$$

The flow constraints for the control flows (i.e., the flows of the SDN controllers) are similar. Flow is produced by switches and received by SDN controllers, except for co-located controllers and switches. We consider in-band control traffic; i.e., these control flows are routed through the same network that the application traffic passes through, and no separate links exist specifically for control flows.

$$\sum_{j\in V} cf_{ij}^s - cf_{ji}^s = \begin{cases} 0 & \text{if } i \in H \\ 1-y_{si} & \text{if } i = s \\ -y_{si} & \text{otherwise} \end{cases} \quad \forall i \in V, s \in S \tag{16}$$

With these flow constraints in place, we must also account for the maximum link capacity:

$$\sum_{h\in H}\sum_{w\in W}[(\sum_{a=1}^{|w|} f_{ij}^{hwc_a} I_{c_a}) + (f_{ij}^{\prime hw} O_{c_n})] + \sum_{s\in S}[cf_{ij}^s \Omega] <= \theta_{ij} \quad \forall l_{ij} \in L \tag{17}$$

A constraint should also be added to consider the maximum tolerable delay for each workflow. To assess this delay, we must consider both the computing time and latency to calculate it. To simplify this computation, the function $SW(i)$ is defined, which is 1 if $i \in S$ and 0 otherwise.

$$\begin{aligned} \sum_{h\in H}\sum_{l_{ij}\in l}(\sum_{a=1}^{|w|}(&\frac{z_{hc_a}^w \xi_{c_a}}{P_h} \\ &+ f_{ij}^{hwc_a}\delta_{ij}) + f_{ij}^{\prime hw}\delta_{ij} \\ &+ SW(j)\sum_{l_{k,m}\in L} cf_{km}^j \delta_{km}) \leq \Delta_w \forall w \in W \end{aligned} \tag{18}$$

The model also requires an objective function to determine which metric must be optimized by changing the values of the future decision variables. In our case, the objective is the average response time of all workflows. Formally, the objective function is represented by (19).

$$\begin{aligned} \sum_{h\in H}\sum_{l_{ij}\in l}\sum_{w\in W}(\sum_{a=1}^{|w|}(&\frac{z_{hc_a}^w \xi_{c_a}}{P_h} \\ &+ f_{ij}^{hwc_a}\delta_{ij}) + f_{ij}^{\prime hw}\delta_{ij} \\ &+ SW(j)\sum_{l_{k,m}\in L} cf_{km}^j \delta_{km}) \end{aligned} \tag{19}$$

(19) can be separated into three terms, as shown above. The first term is the execution time, which depends on the

workload of each microservice and the power of the host on which the microservice runs. The second term is the network latency, which depends on the latency of the links used to transmit information. The third term is the control latency of the path taken by each workflow.

Therefore, the final MILP problem is formulated as *minimize (19) subject to (1-7) (9-18)*.

The proposed DADO formulation is meant to be integrated into the development process of time-strict IoT applications. Concretely, the current formulation of DADO is designed to be implemented as part of the design phase. System and network administrators are expected to provide DADO with the infrastructure's definition, either factual or planned, as a graph containing the parameters defined in the formulation (e.g., link latency, host computational power, link capacity or number of SDN controllers). Then, developers should provide the characteristics of their IoT application (e.g., the number of microservices, definitions of the workflows, tolerable delays or microservice specifications). The infrastructure description, as well as the application description, are the inputs to the DADO formulation, which should be implemented using an automatic solver such as the Python MIP library [20]. The output of the formulation gives different information to each participant: network administrators can see where SDN controllers are to be placed, how they should be set up or which routes are more congested. System administrators know which nodes are going to be loaded the most, as well as where each microservice should be deployed. Finally, developers are provided information about the expected response time or if specific parts of the infrastructure should be scaled up to achieve the target response time (e.g., more powerful servers, faster links or more SDN controllers).

IV. Performance evaluation

In this section, a setup based on the scenario presented in Sec. II-A is shown, and tests are conducted using this setup to evaluate the performance of DADO and compare it with the performance of other deployment strategies.

A. Evaluation setup

In Sec. II-A, we presented a well-defined scenario that is our basis for evaluating DADO, the details of which are taken from [12]. Additionally, we estimated values for the parameters that were not reported in the original case study, such as the number of SDN controllers or the length of functionalities. Finally, in [12], IIoT devices are considered to be unable to compute. However, to evaluate the performance of DADO in environments where mist layer devices are available for deployment, we equip the IIoT devices with devices such as the Arduino Pro Portenta H7 [21], a microcontroller designed for IIoT applications, and the inexpensive single-board computer Raspberry Pi Zero [22], which enables IIoT devices to act as complete computers at a relatively low cost.

We test scenarios that allow evaluations of the scalability and performance of DADO under different conditions. Specifically, we consider microservices that took 100, 500 or 1000 MCycles to run, placing between 1 and 4 SDN controllers, with each device requesting between 1 and 4 functionalities. Each functionality workflow can be 1, 2, 3 or 6 microservices long, and the hardware specifications of the IIoT devices stated above are considered. The topologies considered are labeled by their sizes as small (10 IIoT devices, 10 fog servers), medium (25 IIoT devices, 15 fog servers) and large (50 IIoT devices, 25 fog servers), also based on [12]. To analyze the optimization achieved by DADO and the capacity to set tolerable delays, we set the maximum tolerable delay for all workflows to 4 seconds. In general, the evaluation is performed by setting a default value for all parameters, varying the values of one or more parameters and testing over the three topologies. The default values used are those defined in our initial case study: microservices of 500 MCycles, 1 SDN controller, 2 requests per device, noncomputing IIoT devices and 1-microservice-long functionalities. These values are also shown in Table I.

Table I: Evaluation parameters

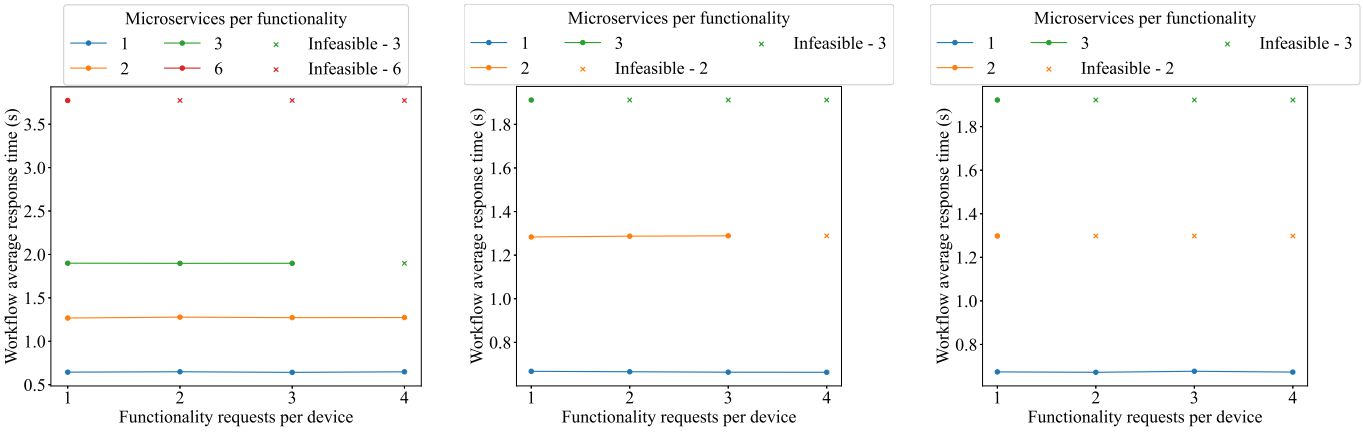
Parameter	**Values**	**Unit**
Microservice workload	100, 500, 1000	MCycles
Number of SDN controllers	1, 2, 3, 4	Controllers
Requests per device	1, 2, 3, 4	Requests
Functionality length	1, 2, 3, 6	Microservices
Topology size	20, 40, 80	Nodes
Maximum tolerable time	4	Seconds
IIoT device hardware	Noncomputing, Arduino Pro Portenta H7, Raspberry Pi Zero	

The evaluation has several objectives: We investigate the validity of DADO and perform tests to show the scalability of DADO as IIoT devices request more functionalities and its computational scalability. This demonstration is crucial, since a company may not deem the investment worthwhile if the response time increases significantly as the system grows. Another key objective is to evaluate the trade-off between latency and response time, to determine if DADO selects a solution with higher latency and reduced execution time only if said solution results in a lower overall response time. We also test whether the tolerant delay constraint is useful for setting a maximum delay for the workflows. Moreover, we compare DADO against alternative deployment strategies to evaluate the reduction in response time. Since routing is also involved, we determine whether higher link loads affect the scalability of the proposal. Finally, we evaluate the optimization time required by the computation of DADO.

B. Evaluation results

The results presented in this section are acquired by combining the solutions obtained by DADO and the values for the parameters previously discussed and calculating the values of the different metrics that are shown to analyze the expected behavior of DADO under different conditions.

In Fig. 3, the scalability of the solution is evaluated by testing the deployment of applications in all three topologies with increasing numbers of microservices, as well as by increasing the number of functionalities requested. These tests are performed with a single SDN controller, microservices of 500 MCycles and noncomputing IIoT devices. We draw



(a) Small topology. (b) Medium topology. (c) Large topology.

Figure 3: Request scalability analysis.

four major conclusions from these results: First, the solution is scalable. When the number of requests per device increases, the response time remains stable. This result implies that the solution provides a good scaling and therefore is able to maintain the response time. The second conclusion, however, is that the solution can scale only as long as the architecture has enough resources to deploy the expected applications; the points labeled *Infeasible* are the estimated positions for parameter values that require more resources (e.g., memory or networking capacity) than the architecture has available, rendering DADO unable to produce solutions. Third, the functionality length (in microservices) is very relevant to the response time, mainly because a functionality workflow with more microservices implies a heavier computational load (e.g., a 2-microservice-long functionality carries twice the computational load of a single-microservice-long functionality) but also because there is an extra cost in latency if these microservices are not executed on the same host. Finally, as the IIoT devices-per-fog-servers ratio rises in larger topologies, fewer microservices per request can be supported, but the results are very similar under all topologies. This also applies to the rest of the performed analyses; therefore, for the remainder of the paper, we show the results for a single topology, while the rest of the results are provided as additional content. In summary, DADO provides scalable solutions to deploy applications as long as the architecture has sufficient capabilities to manage the given application.

Fig. 4 shows the scalability of the application for computational load changes in the small topology. These tests are performed with a single SDN controller, 2 requests per device and noncomputing IIoT devices. First, the slope is steeper when functionalities are longer, which implies that the greater the number of microservices that are executed on a functionality workflow, the longer it takes to execute said workflow, responding to the nature of this service composition (e.g., a single-microservice-long functionality with 1 GCycle microservices has to execute 1GCycle, whereas a 2-microservice-long functionality would have to execute 2 GCycles). However, the relationship is not directly proportional: while a single-microservice-long functionality with 1

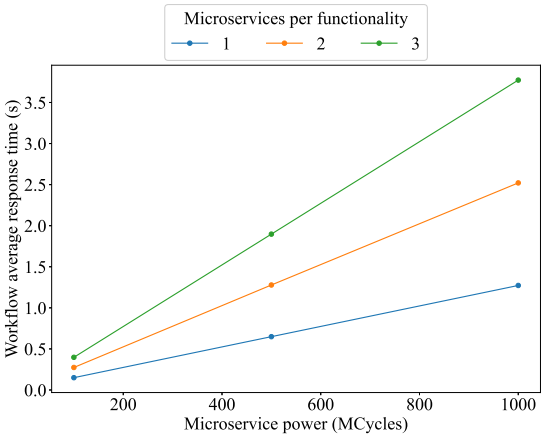


Figure 4: Computational scalability analysis in the small topology.

GCycle microservices and a 2-microservice-long functionality with 500 MCycle microservices have the same computational workload, the response time of the shorter functionality is slightly lower, by 6 ms. This outcome occurs because of the communication latency, since there is a communication delay between each of the microservices in the functionality workflow that does not exist for a single microservice. In addition, while shorter functionalities have lower response times, longer functionalities can be parallelized, and their microservices can be used for other requests, thus compensating for that slight overhead. Overall, DADO provides a solution that is able to minimize the effect of latency in the response time.

Fig. 5 shows the analysis of the scalability of the application under different topologies. These tests are performed with a single SDN controller, 1 request per device, microservices of 500 MCycles and noncomputing IIoT devices. These results further prove the scalability of the solutions provided by DADO, with only a slight increase in response time as the topology size doubles or even triples. The main conclusion to draw from the results is that the IIoT application that is to be implemented, as explained in Sec. II, can be scaled into a larger network without causing a significant increase

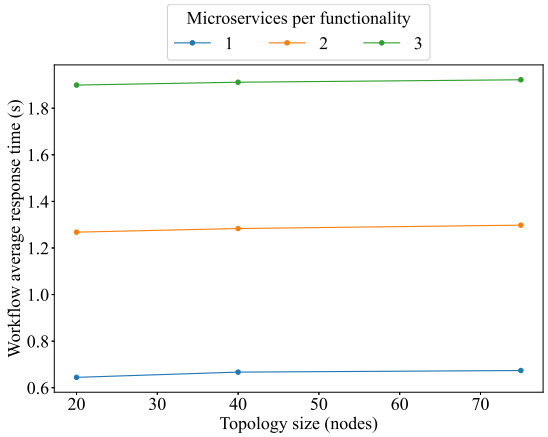


Figure 5: Comparison of response times under different topologies.

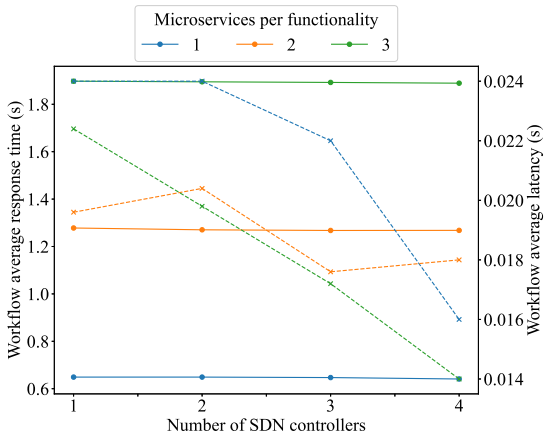


Figure 6: Latency and controller analysis in the small topology. Solid lines show the response time, and dashed lines show the latency.

in response time and thus that system growth will not result in a serious QoS decrease.

Fig. 6 presents an analysis of the effect of adding SDN controllers on the latency and response time in the small topology. The solid lines show the response times and refer to the leftmost Y-axis, while the dashed lines show the latency and refer to the rightmost Y-axis. These tests are performed with 2 requests per device, microservices of 500 MCycles and noncomputing IIoT devices. When short functionalities (1 microservice per functionality) are considered, latency does not decrease steeply when a single controller is added. Nonetheless, the decrease becomes steeper as more controllers are added. For longer functionalities (2 microservices per functionality), latency rises when there is an even number of controllers: reaching 0.8 ms in the case of 2 controllers and 0.4 ms in the case of 4. Despite this phenomenon, latency declines steeply when odd numbers of controllers are considered, with a minimum of 17.6 ms in the case of 3 controllers. On even longer functionalities (3 microservices per functionality), the latency decreases almost linearly, an average of 2.4 ms per controller, as the number of controllers increases. However, the response time decreases slightly and steadily as controllers

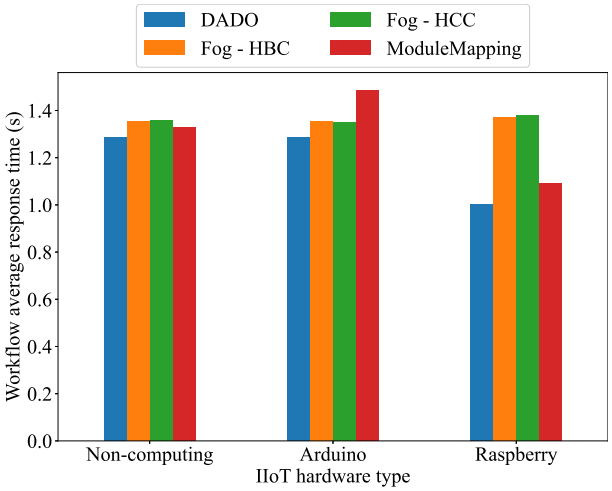


Figure 7: Performance benchmark of DADO in the medium topology.

are added in every case. In the case of 2 microservices per functionality, we find that the response time decreases for 1 ms and 0.4 ms for 2 and 4 controllers, respectively. This result is interesting because the decrease in response time comes with an increase in latency of a similar amount. The essential conclusion is that the effects of adding SDN controllers over latency heavily depend on how traffic flows are steered, providing further indication that the computing and networking dimensions affect one another's QoS. Furthermore, the latency rises for two-microservice-long functionalities as a result of a trade-off – DADO chooses a deployment with higher latency because it decreases the execution time and minimizes the overall response time, something that would not be as simple if both dimensions were considered separately. Thus, DADO is able to find the solution to this trade-off between execution time and latency, which not only enables a smart computation distribution scheme, offloading computation when latency is low enough for it to be worth it, but also enables smart controller placement that considers and complements these offloading decisions.

In Fig. 7, the performance of DADO is compared against that of other solutions in the medium topology. Concretely, DADO is compared with deploying the application directly in the fog and placing the SDN controllers in the nodes with the highest betweenness centrality (HBC) and highest closeness centrality (HCC). Moreover, the ModuleMapping solution, proposed in [23], is used as a benchmark. Fog deployments with the HBC and HCC are performed by matching microservices with fog servers in a round-robin fashion, making sure that the total memory of the fog servers is never surpassed. ModuleMapping, on the other hand, is a microservice placement method for IoT applications in fog environments, created specifically to serve as a benchmark. Due to the lack of methods that jointly include microservice deployment, routing optimization and controller placement, the focus has been placed on service placement in the case of ModuleMapping. Thus, in all three cases, the routing is still optimized through the formulation proposed by DADO.

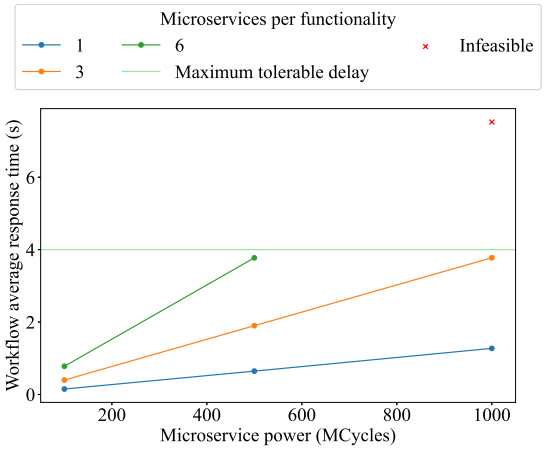


Figure 8: Maximum tolerable delay compliance of DADO in the small topology.

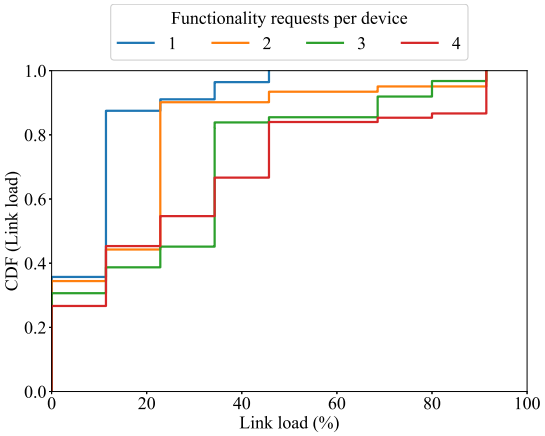


Figure 9: Empirical CDF of the link load in the medium topology.

These tests are performed with a single SDN controller, 2 requests per device, 2-microservice-long functionalities and microservices of 500 MCycles on different IIoT devices. These values allow a test focused on microservice deployment while maintaining similar values to those of previous tests. Focusing on the HBC and HCC, the analysis clearly shows that DADO provides shorter response times and is able to speed up the response time by up to 37.89%. The largest differences between the HBC, the HCC and DADO appear for the most powerful device, Raspberry Pi, because of the hybrid deployment capabilities of DADO. While the HBC and HCC solve the DCDP only on the fog layer, DADO uses all available layers to optimize the response time. This outcome is clearly shown on Arduino hardware: DADO refuses to deploy in this type of device due to the fact that the derived slow execution would increase the response time. Nonetheless, hybrid deployment is not the only factor of DADO enhancement, as ModuleMapping also features this capability. Moving to ModuleMapping, we also find that DADO consistently obtains lower response times. The largest speed-up is found with the Arduino devices, in which the difference is approximately 198 ms. While ModuleMapping considers only the capabilities of devices and microservices for the deployment, DADO has a holistic view of the architecture, network and application. Therefore, it is able to change the placement of SDN controllers and the paths taken by traffic to shorten the response time. Moreover, network latency is considered to deploy different microservices in a single workflow. We conclude that DADO outperforms the considered benchmarks.

Fig. 8 shows the impact of maximum tolerable delays in the solutions DADO yields. The solid light green line represents the maximum tolerable delay, while each dot represents a successful deployment, and a cross indicates deployments that could not fulfill the delay. These tests are performed with a single SDN controller, 1 request per device and noncomputing IIoT devices. Most configurations of microservices per functionality and microservice power can be deployed, but for 6-microservice-long functionalities and 1 GCycle microservices, DADO determines that it is infeasible to satisfy the constraint.

Table II: Multivariate analysis of the response time.

	Coefficient	Std. Error	P-value
Intercept	1.2276	0.272	$\mathbf{2.3 \cdot 10^{-5}}$
Topology size	0.0002	0.003	0.935
SDN controllers	-0.0163	0.052	0.757
Requests per device	-0.0051	0.069	0.941
Functionality length	0.6503	0.046	$\mathbf{2.5 \cdot 10^{-23}}$
Cycles per microservice	-1.1725	0.091	$\mathbf{3.3 \cdot 10^{-21}}$
IIoT device hardware	-0.3071	0.171	0.077

If it were to be relaxed, the response time of the workflows would be at the point indicated by the cross, 7.5244 seconds, almost double the defined maximum tolerable delay. Therefore, because of the delay constraint, DADO indicates that the delay cannot be met without scaling up the computational power.

In Fig. 9, the empirical CDF of the link load in the medium topology is depicted. These tests are performed with a single SDN controller, 2 requests per device, 1-microservice-long functionalities, noncomputing IIoT devices and microservices of 500 MCycles. The main conclusion to draw from these results is that there is a directly proportional relationship between the number of requests per device and the link load. If one request per device is considered, all links present a load below 46%. If more requests are made, the higher bound rises to 91.42%. This increase comes from the fact that, from two requests onward, at least one fog node is always fully loaded. Therefore, more traffic flows need to reach it through its unique link, which results in link load increase. Another interesting result is that the majority of the links are not heavily loaded: the median load is approximately 11.42% for one request, 22.85% for two requests, 34.28% for three requests and 22.85% for four requests, all of which are well below 50%. Finally, we determine that the paths of traffic flows are all 2 hops long. This result implies that the workflows are launched, executed and returned back to the source IIoT node. Thus, DADO is scalable networkwise as well.

To statistically validate the results described above, we performed a multivariate analysis evaluating the impact of a set of parameters on the response time, which is set as the dependent variable. Table II shows the results of this

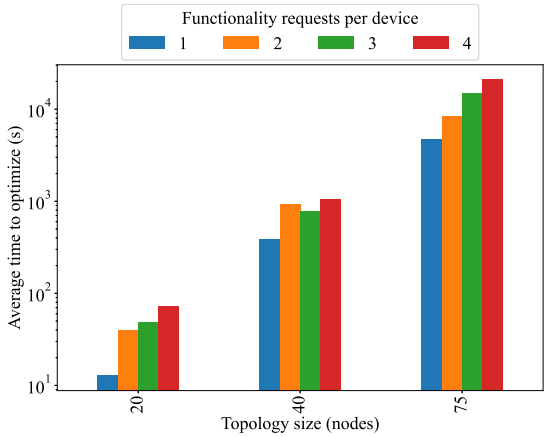


Figure 10: Average optimization times of DADO.

analysis. The independent variables are topology size (the number of nodes), the number of SDN controllers, the number of requests per device, the length of functionalities, the number of cycles per microservice and the type of IIoT hardware as a binary variable (0 for noncomputing and Arduino, 1 for Raspberry Pi). The R^2 coefficient of this analysis is 0.841. This analysis indicates that the size of the application, mainly expressed through the length of the functionalities and the cycles per microservice, is the most statistically significant variable for the response time. However, they are bound to be in balance: to reduce the cycles per microservice, the number of microservices per functionality must be increased, increasing the microservice length and likely adding overhead. To shorten a functionality, different microservices can be merged into a single microservice, but the resulting microservice is heavier (i.e., more cycles per microservice) and often requires more memory. Other interesting conclusions can be extracted from the coefficients of each variable, which explain how the response time increases or decreases as the coefficients are modified (e.g., adding a controller reduces the response time to 0.0163 s).

More conclusions can be drawn from the overall results of the performance analysis, especially related to implementing DADO in real scenarios. First, DADO is able to identify when an infrastructure needs to scale by labeling a deployment as infeasible. In our experiments, most scalability issues were related to a lack of computing power to run all the required microservices. Moreover, the evaluation shows that the main bottleneck for performance in this case study is computational power, since the infrastructure does not contain very powerful servers. However, we believe that this bottleneck is infrastructure-specific and may even be application-specific (e.g., an application with light computing requirements but large amounts of data may be limited by the network). In addition, DADO can be used to analyze the computational scalability, as reported in Fig. 4, and the performance gains from the addition of controllers, as shown in Fig. 6, which serves as a method for finding the specific bottleneck in each situation and planning the infrastructure and deployment more effectively.

In Fig. 10, the average optimization times for DADO are analyzed in all three topologies. These tests were conducted with a single SDN controller, 1-microservice-long functionalities, noncomputing IIoT devices and 500 MCycle microservices. The main outcome from this analysis is the fact that the response time exponentially increases with the topology size. Optimizing smaller topologies takes between 12 and 73 seconds, whereas optimizing the medium topology takes between 381 and 1041 seconds and optimizing the larger topology requires between 4664 and 20982 seconds. Moreover, the more requests per device there are, the more time is required to optimize. This phenomenon is due to the fact that each request requires new microservices to be deployed and new traffic flows to be routed, thus generating a larger problem. Furthermore, the impact of these new microservices and traffic flows needs to be considered to optimize the rest of the requests. Therefore, the MILP solution for DADO is mainly suitable during the design time, and it can be leveraged only during the execution time in small topologies, where it can be optimized within seconds.

Since the Fog-SDN deployment problem is a combination of the DCDP and the CPP, both of which have been proven to be NP-hard [10], [13], and the combination of any given NP-hard problem with another problem is, by definition, also NP-hard, the Fog-SDN deployment problem is also NP-hard. This NP-hardness is reflected in the limitations of DADO. The MILP solution of DADO can be feasibly applied only to infrastructures with under 300 nodes. In larger infrastructures, the formulation indicates that nearly 18 exabytes of memory would need to be allocated. Furthermore, with the current implementation of DADO, adapting the deployment to changes in the environment implies re-running the proposed solution from the beginning due to the characteristics of the MILP version. This makes the MILP version of DADO suitable for design-time optimization, as there are no strict timing limitations on the optimization time. As future work, we plan to add heuristic solutions to DADO. These heuristics will allow DADO not only to be executed periodically in short loops but also to reuse previous solutions as the basis for further optimization, enabling DADO to adapt the deployment to new conditions over time.

V. Related work

In an attempt to shorten response times in different applications, researches are studying fog computing paradigms both to apply these paradigms and to standardize them. Yousefpour *et al.* [2] surveyed the different cloud and fog computing paradigms, providing insights into their similarities and differences. Though in this paper, we evaluate DADO in a hybrid fog and mist computing scenario, DADO is designed to optimize deployments that make use of other fog computing paradigms as well, such as pure fog computing. Bellavista *et al.* [5] surveyed different proposals that leverage the fog computing paradigm for use in IoT applications as an approach to support the strict response-time requirements of some of these applications. However, these proposals mostly involve different platforms that make use of fog computing and provide services such as communication, security or resource

virtualization that simplify the deployment of IoT applications in fog environments; most do not provide solutions for optimizing deployment on the proposed platforms.

The problem that must be addressed when optimizing the deployment of microservices to a fog computing infrastructure, focusing on the computing dimension, is the DCDP. The term DCDP was coined by Choudhury *et al.* [6], who solved the DCDP in mist computing environments. The main idea behind the DCDP in their work is the replication of an application's cloud services in smartphones and IoT devices to enhance their response time, allowing nearby devices to consume the replicated services within an ad hoc network. This solution to the DCDP focuses on optimizing the resources used for replication in the mist layer while delivering an appropriate response time instead of focusing on delivering optimal response times or optimizing the networking dimensions in other fog computing environments, as DADO does. Mukherjee et al. present in [24] an approach to the DCDP able to deploy microservices to the fog considering the delay due to the execution time and the energy consumption of service execution. Sun *et al.* [12] propose a double-auction heuristic scheme for optimizing the deployment of IIoT applications in a fog environment. The response times offered by fog servers and required by services are assessed as prices, and an auction-based algorithm is leveraged to optimize the deployment. However, they do not optimize the response time and instead try to optimize the number of services that can be successfully deployed. Several proposals for solutions to the DCDP that use a variety of techniques such as MILP, heuristics or game theory are surveyed in [13]. Although some partially optimize network latency through routing, as DADO does, none optimize the latency of the network through SDN controller placement.

Fog computing paradigms, while supporting QoS requirements that are difficult to support with cloud computing, also have challenges that need to be addressed, such as the previously discussed service discovery problem. [8] presents a proposal for applying SDNs as a solution to these challenges that is transparent to the different hosts involved in the network. Other research efforts, such as [14] and [25], solve the DCDP using an underlying SDN. Although these solutions are also built to support IoT applications, they do not consider the effects on latency of SDN controller placement.

On the networking plane, the SDN CPP is a well-known problem that has a significant impact on network latency. [16] adds the idea of dynamic flows to the CPP, providing a solution to the CPP that not only works for predefined, static flows but can also be varied in response to flow variation (e.g., because of changes in routing or traffic). Although this is a relevant contribution to the CPP, the relationship between the networking and computing planes is not considered. [18] not only solve the CPP with a Varna-based heuristic approach but also classify the CPP into 12 types based on the SDN features that are considered. DADO is a type 4, uncapacitated CPP, but it also adds routing capabilities that are not considered in this classification; in addition, it relates the computing and networking planes. Finally, [10] surveys several proposals for CPP solutions that use techniques similar to those used in solutions to the DCDP but also fail to integrate the computing plane as DADO does.

Moreover, there is a similar problem to the DCDP in the networking field when the network function virtualization (NFV) paradigm is leveraged: function orchestration in service function chaining (SFC). NFV allows networking equipment, such as routers or switches, to perform network functions (e.g., firewall, access lists) without dedicated boxes (i.e., virtual network functions). In SFC, a network flow can request for some of these virtual network functions to be performed over it, in a concrete order. SFC is very similar to the MSA in IoT applications: a set of services that can be called independently or jointly by following a sequence. Function orchestration in SFC consists of finding the optimal routes for said flows, as well as the optimal placement for virtual network functions, for SFC requests to be fulfilled. The main conceptual difference is that a flow in SFC has a defined source and destination, and functions have to be performed along the way. In an MSA, if the source has every microservice deployed to it, there may be no flow at all. Furthermore, even if the flow exists, there is no defined destination in an MSA, as every host that deploys one or more of the requested microservices is a possible destination. In [26], an approach to solving this problem is presented, which optimizes the energy consumption and network side-effect of optimal function placement. Several algorithms are presented, including algorithms for online and offline optimization. A similar approach is found in [27]. In this case, the failure probability is also accounted for in the optimization objective. Furthermore, the system can be triggered in response to failures to perform failure recovery with minimal network side-effects. Despite the similarities between problems, DADO is conceptually different from them, as the problem DADO solves is not related to SFC.

Table III presents a categorization and comparison of the presented works for quick reference. As depicted, DADO is the only system that considers both service and controller placement. While both the DCDP and CPP are well known, to the best of our knowledge, no other work integrates the DCDP and the CPP into a single, cohesive problem as DADO does, nor does any other work take into account the relationship and influences between the two problems. Therefore, the contribution of DADO is the integration of the CPP and the DCDP to provide optimal response times for IIoT applications.

VI. Conclusions and future work

The potential for real-world interactions that IoT provides has drawn interest regarding the use of IoT in intensive domains such as industry or healthcare [3], [4]. IoT applications from these domains are critical, and as such, achieving an optimal response time becomes crucial. Bringing computing resources closer to the edge through fog computing is essential for this achievement but not enough to obtain it. Service discovery, monitoring and routing optimization must also be considered. Moreover, modern IoT applications require service discovery and monitoring to be performed transparently. All these services can be provided by an underlying SDN. However, to achieve optimal QoS, it is necessary to optimize the deployment of microservices and the placement of SDN

Table III: Categorization of related work

Work	Category	Service placement	Controller placement	Routing optimization
[6]	DCDP	Yes	No	No
[24]	DCDP	Yes	No	No
[12]	DCDP	Yes	No	No
[13]	DCDP	Yes	No	No
[14]	DCDP with SDN	Yes	No	No
[25]	DCDP with SDN	Yes	No	No
[16]	CPP	No	Yes	No
[18]	CPP	No	Yes	No
[10]	CPP	No	Yes	Yes
[26]	SFC	Yes	No	Yes
[27]	SFC	Yes	No	Yes
DADO	Fog-SDN Deployment Problem	Yes	Yes	Yes

controllers. This work defines and formalizes the problem of optimizing fog computing and SDN infrastructures for QoS-strict IoT applications with an MSA. To do so, the optimization efforts from both the computing and the networking dimensions are merged. By optimally distributing microservices between nodes, optimally placing SDN controllers and taking into account the mutual influences between both dimensions, optimal decisions are made, and suboptimal solutions are avoided. To solve the joint problem of computation distribution and controller placement, a framework named DADO is proposed. The performance evaluation over an IIoT scenario shows that DADO provides scalable deployment plans that trade-off execution time and latency optimally. Moreover, DADO reduces the response time by up to 37.89% by optimizing deployment and allowing for hybrid (e.g., fog and mist layer) fog computing deployments, as well as providing scalable solutions.

In the future, we expect to extend DADO. First, we intend to develop heuristics that will allow DADO to be applied to infrastructures larger than 300 nodes while still finding near-optimal solutions. Moreover, heuristics will allow DADO to reuse previous solutions in the adaptation process during the execution time. We also intend to add mobility considerations to these heuristics, allowing DADO to consider and trade off the QoS degradation of maintaining a deployment plan with the cost of a reconfiguration. Finally, we intend to expand DADO to consider other QoS features, such as reliability, and to combine these QoS features to develop a multiobjective version of DADO.

REFERENCES

[1] J. Singh, T. Pasquier, J. Bacon, H. Ko, and D. Eyers, "Twenty Security Considerations for Cloud-Supported Internet of Things," *IEEE Internet of Things Journal*, vol. 3, no. 3, pp. 269–284, jun 2016.

[2] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *Journal of Systems Architecture*, vol. 98, pp. 289–330, 2019.

[3] 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.

[4] L. Greco, G. Percannella, P. Ritrovato, F. Tortorella, and M. Vento, "Trends in IoT based solutions for health care: Moving AI to the edge," *Pattern Recognition Letters*, vol. 135, pp. 346–353, 2020.

[5] 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.

[6] B. Choudhury, S. Choudhury, and A. Dutta, "A Proactive Context-Aware Service Replication Scheme for Adhoc IoT Scenarios," *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, pp. 1797–1811, 2019.

[7] N. Wang, K. H. Ho, G. Pavlou, and M. Howarth, "An overview of routing optimization for internet traffic engineering," *IEEE Communications Surveys and Tutorials*, vol. 10, no. 1, pp. 36–56, mar 2008.

[8] A. C. Baktir, A. Ozgovde, and C. Ersoy, "How can edge computing benefit from software-defined networking: A survey, use cases, and future directions," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2359–2391, 2017.

[9] P. Bellavista, C. Giannelli, T. Lagkas, and P. Sarigiannidis, "Quality management of surveillance multimedia streams via federated sdn controllers in fiwi-iot integrated deployment environments," *IEEE Access*, vol. 6, pp. 21 324–21 341, 2018.

[10] T. Das, V. Sridharan, and M. Gurusamy, "A survey on controller placement in sdn," *IEEE Communications Surveys & Tutorials*, pp. 472–503, 2019.

[11] A. Carrega, M. Repetto, P. Gouvas, and A. Zafeiropoulos, "A middleware for mobile edge computing," *IEEE Cloud Computing*, vol. 4, no. 4, pp. 26–37, 2017.

[12] W. Sun, J. Liu, Y. Yue, and H. Zhang, "Double Auction-Based Resource Allocation for Mobile Edge Computing in Industrial Internet of Things," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4692–4701, 2018.

[13] 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.

[14] Z. Lv and W. Xiu, "Interaction of Edge-Cloud Computing Based on SDN and NFV for Next Generation IoT," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 5706–5712, 2019.

[15] B. Zhang, X. Wang, and M. Huang, "Multi-objective optimization controller placement problem in internet-oriented software defined network," *Computer Communications*, vol. 123, pp. 24–35, 2018.

[16] M. T. I. ul Huque, G. Jourjon, and V. Gramoli, "Revisiting the controller placement problem," in *2015 IEEE 40th Conference on Local Computer Networks (LCN)*, 2015, pp. 450–453.

[17] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba, "Dynamic controller provisioning in software defined networks," in *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*, 2013, pp. 18–25.

[18] A. K. Singh, S. Maurya, and S. Srivastava, "Varna-based optimization: a novel method for capacitated controller placement problem in SDN," *Frontiers of Computer Science*, vol. 14, no. 3, p. 143402, 2020.

[19] P. Bellavista, C. Giannelli, and D. D. P. Montenero, "A reference model and prototype implementation for sdn-based multi layer routing in fog environments," *IEEE Transactions on Network and Service Management*, 2020.

[20] T. A. Toffolo and H. G. Santos, "Python-MIP," 2020. [Online]. Available: https://www.python-mip.com/

[21] Arduino, "Arduino Pro," 2020. [Online]. Available: https://www.arduino.cc/pro/hardware/product/portenta-h7

[22] Raspberry Pi Foundation, "Raspberry Pi Zero," 2018. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-zero/

[23] M. Taneja and A. Davy, "Resource aware placement of iot application modules in fog-cloud computing paradigm," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2017, pp. 1222–1228.

[24] M. Mukherjee, V. Kumar, S. Kumar, R. Matamy, C. X. Mavromoustakis, Q. Zhang, M. Shojafar, and G. Mastorakis, "Computation offloading strategy in heterogeneous fog computing with energy and delay constraints," in *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. IEEE, 2020, pp. 1–5.

[25] M. Huang, W. Liu, T. Wang, A. Liu, and S. Zhang, "A Cloud-MEC Collaborative Task Offloading Scheme with Service Orchestration," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 5792–5805, 2019.

[26] M. M. Tajiki, S. Salsano, L. Chiaraviglio, M. Shojafar, and B. Akbari, "Joint energy efficient and qos-aware path allocation and vnf placement for service function chaining," *IEEE Transactions on Network and Service Management*, vol. 16, no. 1, pp. 374–388, 2018.
[27] M. M. Tajiki, M. Shojafar, B. Akbari, S. Salsano, M. Conti, and M. Singhal, "Joint failure recovery, fault prevention, and energy-efficient resource management for real-time sfc in fog-supported sdn," *Computer Networks*, vol. 162, p. 106850, 2019.

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.

Jaime Galán-Jiménez received a Ph.D. in computer science and communications from the University of Extremadura in 2014. He is currently with the Computer Science and Communications Engineering Department, University of Extremadura, as an Assistant Professor. His main research interests are SDNs, 5G network planning and design, and mobile ad hoc networks.

Javier Berrocal (IEEE Member) is a cofounder 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.

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