UNIVERSIDAD DE EXTREMADURA

TESIS DOCTORAL

DADO: FRAMEWORK PARA EL DESPLIEGUE DE APLICACIONES IOT DISTRIBUIDAS EN ENTORNOS EDGE-FOG-CLOUD

JUAN LUIS HERRERA GONZÁLEZ

PROGRAMA DE DOCTORADO EN TECNOLOGÍAS INFORMÁTICAS (TIN)

Con la conformidad de los directores

Dr. José Javier Berrocal Olmeda y *Dr. Juan Manuel Murillo Rodríguez*

2023

UNIVERSIDAD DE EXTREMADURA

TESIS DOCTORAL

DADO: FRAMEWORK PARA EL DESPLIEGUE DE APLICACIONES IOT DISTRIBUIDAS EN ENTORNOS EDGE-FOG-CLOUD

JUAN LUIS HERRERA GONZÁLEZ

PROGRAMA DE DOCTORADO EN TECNOLOGÍAS INFORMÁTICAS (TIN)

2023

Who even needs alchemy, when
I've got them?

Fullmetal Alchemist
Hiromu Arakawa

Acknowledgments

I have always found ironic that, although this is the first section in the dissertation document, it also is the last section to be written. Now that the marathon that this thesis has been, and the final sprint that finishing this dissertation has represented, it is time to thank everyone that has allowed it to exist as it is. As Ovid said, *"Finis Coronat Opus"*.

First, I would like to make the formal acknowledgments. This work was partially funded by the project PID2021-124054OB-C31 and the grant CAS21/00057 (MCI/AEI/FEDER,UE) 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, GR21133, IB18030), and by the European Regional Development Fund. Furthermore, this work has been supported by the Fernando Valhondo Calaff Foundation. The resources used to create some of the figures include the icons "iot microchip" (created by Philipp Petzka), "Thermometer" (created by Riyan Resdian), "Merger" (created by David Christensen), "Server" (created by WEBTECHOPS LLP), "Phone" (created by rex), "Computer" (created by LUTFI GANI AL ACHMAD), "Network" (created by Кирилл Конюх), "performance" (created by Andrejs Kirma), "Energy" (created by Rahmat Hidayat), "Face" (created by DinosoftLab), "feature owner" (created by Juicy Fish), "matching" (created by Alex Burte), "Server" (created by twist.glyph), "Results" (created by SBTS), "routing" (created by Mas Dhimas), "project price estimation" (created by Juicy Fish). These icons have been obtained from The Noun

Project (`http://thenounproject.com/`).

Next, I would like to greatly thank my directors, Javi and Juanma. Without you, I would not even have had the opportunity to start this thesis. I owe you my academic path, and I hope that we continue to collaborate into the future. Your guidance has been immensely helpful, and your willingness to always help, along with your constant availability, have made this path much more fulfilling than I initially believed. Back in 2019, you placed your bet on me, with the little information that you had, to tackle this work. I hope the bet has paid off, and I hope it will continue to pay off.

This acknowledgment is also extended to my wonderful laboratory colleagues: Luis, Joserra, Sergio, Dani, Javi, Rome, José, Quique, Jaime and Jaime. You have paved my path, and continue to do so every day, and I am really lucky to have you as co-workers. I believe that we have a team full of extraordinary and awesome people, and that is what counts the most. Besides them, I would also like to thank my collaboration colleagues, as well as my colleagues in Pisa: Christine, Hsiao-Yuan, Luca, Paolo, Domenico, Giuseppe, Antonio, Stefano, Jacopo, Luis, Elena, Alejandro, Francisco, Pablo... In different ways, you are all part of this work and of my academic path.

To my friends, who have made this path a lot easier. For all the pool games, each dart we have thrown, and every last round where I randomly chose the unsafest move in the game. Thank you for every single moment and tournament in these years, and for many more to come. Most especially, I would like to thank Kimmy, who is the name that will always come to my mind when I think of my greatest driving force. Your support is my main motivation to go on every day.

Y finalmente, cambio de idioma para agradecer a mi familia. A mis padres, a mis abuelos, a mis tíos, a mis primos, y a absolutamente todos los que me han apoyado en este duro y largo camino. A los que estuvieron, a los que están, y a los que estarán. En especial, me gustaría agradecer a mi hermana, una verdadera artista cuyo apoyo ha sido fundamental para esta tesis y para todo lo demás. Parafraseando al novelista Ryukishi07, *”A la Bruja de los Orígenes, que ostenta el mágico poder de crear unos*

a partir del mar de ceros. Puesto que solo cero se puede obtener de multiplicar cero, pero los unos creados por ella un día superarán incluso a los cielos". Gracias de corazón por vuestro apoyo todo este tiempo, en Plasencia, en Cáceres, e incluso en Pisa.

Abstract

We live in a world where *smart* devices are becoming truly ubiquitous: from wearable devices to vehicle equipment or home appliances, it is now common for them to have small, power-efficient computers and wireless connections that allow them to be leveraged by computer applications, thus becoming part of the Internet of Things (IoT). In fact, this ability to bridge the gap between the real world and computing applications has brought interest from intensive domains, such as industry or healthcare, as it allows for the management, automation, and monitoring of real-world processes.

Nonetheless, the *smart* characteristics of these devices are not usually provided by software that runs directly on the device, and instead, they communicate with software that is usually running in *the cloud*: a set of servers in remote data centers that compute information on demand. This is especially important in intensive domains, where the requirements in terms of Quality of Service (QoS) are especially strict (e.g., low response times, low costs). Due to these strict QoS requirements and the distance between IoT devices and the cloud, which involves a higher network latency, it is complicated to leverage cloud computing for intensive applications. Consequently, new paradigms based on bringing computing resources closer to users are emerging, creating the concept of a continuum of computing resources that ranges from the cloud to the IoT device: the Cloud-to-Thing continuum. Moreover, the communications generated by the mass of IoT devices call for scalability, flexibility, and general programmability in networks. These requirements can be provided by the Software-Defined Networking (SDN) paradigm, in which networks are programmed by using controllers. Furthermore, the need for IoT application software for evolvability, distribution, and interoperability, calls for a migration from monolithic application architectures into flexible Microservice Architectures (MSAs), allowing low-power IoT devices to request application functionality in a lightweight and simple manner.

However, to obtain a high QoS in this environment, it is necessary to optimize how these paradigms are used: the placement of microservices, computing nodes, and SDN controllers must be optimized to meet the organizational needs. Furthermore, an additional consequence of the use of these paradigms is that the organization can hold control over all three dimensions: application, computing, and networking. This calls for holistic optimization, that coordinates the decisions in all three to meet the specified needs. Moreover, these needs can span multiple QoS metrics, requiring a compromise solution that trades them off. In this PhD thesis, we present a set of approaches, models, tools, techniques, and architectures that tackle the problem of holistic, multi-objective optimization of intensive IoT application deployment. We present a total of 14 software artifacts and 22 scientific publications aimed at different aspects of this problem, including holistic optimization, multi-objective optimization, dynamic adaptation of the optimization decisions, and integration with software development practices and methodologies.

Resumen

Vivimos en un mundo en el que los dispositivos *inteligentes* son realmente ubicuos: desde dispositivos *ponibles* hasta equipamiento de vehículos o electrodomésticos, ya es común que todos ellos integren pequeños y eficientes computadores y conexiones inalámbricas que los conectan a aplicaciones informáticas, pasando a ser parte del Internet de las Cosas (IoT). De hecho, esta posibilidad de unir los mundos real e informático ha generado interés por parte de dominios intensivos, como la industria o la salud, ya que permite que procesos del mundo real sean gestionados, automatizados y monitorizados.

Sin embargo, las características *inteligentes* de estos dispositivos no suelen venir de software que ejecute el dispositivo, sino de la comunicación con software lanzado en *la nube*: un conjunto de servidores en centros de datos remotos que procesan la información bajo demanda. Esto es especialmente importante en los dominios intensivos, en los que los requisitos en términos de Calidad de Servicio (QoS) son especialmente estrictos (bajos tiempos de respuesta, bajos costes, etc.). Debido a estos requisitos estrictos y a la distancia entre los dispositivos IoT y la nube, lo que implica una mayor latencia de red, es complejo utilizar la nube para estas tareas intensivas. Consecuentemente, nuevos paradigmas basados en traer recursos de cómputo más cerca de los usuarios finales, creando un continuo de cómputo entre el dispositivo IoT y la nube, empiezan a emerger como el continuo Cloud-to-Thing. Además, las comunicaciones generadas por la masa de dispostivos IoT requiere de escalabilidad, flexibilidad y programabilidad general en redes, requisitos que pueden cumplirse gracias al paradigma de las Redes Definidas por Software (SDN), en el que las redes son programadas a través de controladores. Es más, el propio software de las aplicaciones IoT requiere de flexibilidad, distribución, mantenibilidad e interoperabilidad, lo que está provocando una migración de las arquitecturas monolíticas a las Arquitecturas de Microservicios (MSAs), que permiten que los dispositivos IoT de baja potencia soliciten la ejecución de funcionalidades remotas de forma ligera y sencilla.

No obstante, para obtener una alta QoS en este entorno, es necesario optimizar como se utilizan estos paradigmas: la colocación de microservicios, nodos de cómputo y controladores SDN debe ser optimizada para cumplir las necesidades de la organización. Adicionalmente, como consecuencia del uso de estos tres paradigmas, las organizaciones pueden controlar a la vez las tres dimensiones: aplicación, computación y red. Este hecho atrae interés sobre la optimización holística, que coordina las decisiones en las tres dimensiones para satisfacer los requisitos definidos. Además, estas necesidades pueden incluir varias métricas de QoS, requiriendo de soluciones de compromiso que realicen un *trade-off* entre las diversas métricas. En esta tesis doctoral, presentamos un conjunto de modelos, herramientas, técnicas y arquitecturas para atacar el problema de la optimización holística multiobjetivo del despliegue de aplicaciones IoT intensivas. Presentamos un total de 14 artefactos software y 22 publicaciones científicas enfocadas en diversos aspectos del problema, incluyendo la optimización holística, la optimización multiobjetivo, la adaptación dinámica de las decisiones de optimización y la integración con prácticas y metodologías de desarrollo de software.

Contents

List of Figures

List of Tables

Felix qui potuit rerum cognoscere causa.
(*Blessed are they who have been able to win knowledge of the causes of things.*)

Georgics, Book II
Publius Vergilius Maro (*Virgil*)

Chapter 1

Introduction

The Internet of Things paradigm allows for the interaction of computer applications with the real world through the use of sensors and actuators. In recent years, the interest in this paradigm as a means to automate real-world processes of intensive domains has increased. Nonetheless, the use of the Internet of Things in intensive domains requires the optimization of the Quality of Service in the deployment phases. This thesis dissertation presents the set of approaches, models, tools, techniques, and architectures aimed at the optimization of the Quality of Service of Internet of Things distributed applications that we have developed in recent years. Moreover, these approaches are characterized by considering the characteristics of the application software, the computing devices, and the network equipment using a holistic point of view. In this chapter, we first present the research context of this work in Section 1.1. Then, we state the target problem of the presented works in Section 1.2. The goals of the performed research, as well as the research questions that derive from them, are presented in Section 1.3. The research methodology followed through the PhD is presented in Section 1.4, while the contributions and publications of the research line are summarized in Section 1.5. Finally, Section 1.6 describes the outline of the remainder of the dissertation.

1.1 Research context

When Internet of Things applications are considered, especially those intended to be used in intensive domains, there exist numerous concepts, dimensions, and elements that affect the system's behavior. In this section, we revise the state of the art in some of the most important and fundamental of these concepts. Namely, the Internet of Things paradigm is introduced in 1.1.1, while Quality of Service is defined in 1.1.2. The application design paradigm of Service-Oriented Computing is presented in 1.1.3. The Cloud-to-Thing Continuum computational paradigm is defined in 1.1.4. Finally, the networking paradigm used to communicate the different modules, Software-Defined Networking, is presented in 1.1.5.

1.1.1 Internet of Things (IoT)

In 1991, Mark Weiser presented his vision of how technology in the future would merge with reality [6]. The IoT paradigm has emerged in recent years as one of the most popular enablers of this vision, as it is expected to reach 14.7 billion IoT devices, which represent 50% of the devices connected to the Internet globally, through 2023 [7]. Using IoT, computer applications can interact with the real world, by making use of *sensors* and *actuators*. Sensors allow for the transformation of real-world stimuli into data that can be used as input for applications, e.g., a digital thermometer, a microphone or a presence detector are all considered sensors. On the other hand, actuators perform the inverse process by converting data back into real-world stimuli, allowing applications to output their results to the real world. For instance, a speaker, a buzzer, or a lightbulb are actuators.

However, sensors and actuators are unable to process data or communicate with other devices using a network fabric. Hence, they are generally connected with computing devices, that are also able to provide access to the data of their sensors and actuators through the Internet. These computing devices are called *IoT devices*, and the applications they run and communicate with are known as *IoT applications*. For

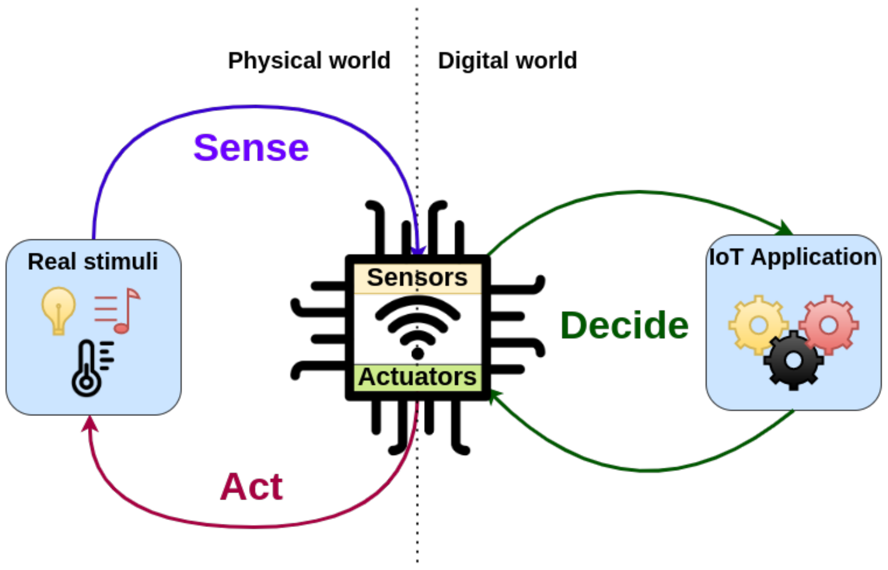


Figure 1.1: Structure of the Sense-Decide-Act loop in IoT applications.

instance, smart lightbulbs or voice assistants are IoT devices, and the application that allows the voice assistant to power the lightbulb on and off through a voice command is an IoT application. These applications can, potentially, automate real-world processes. For instance, using a luminosity sensor, it is possible to automatically power the lightbulb on and off, as well as to automatically regulate its brightness, depending on the light inside the room. This is done in a loop, which starts by obtaining data from the sensors, continues with the IoT application processing the sensed data, and finishes by using the actuators according to the IoT application's result. This loop, known as the Sense-Decide-Act loop [8], is depicted in Figure 1.1

The potential for automation of IoT applications has led to an interest in its application in intensive domains, where it can be used to automate critical processes. For instance, some of the research lines on the Industrial Internet of Things (IIoT) focus on two kinds of applications: process automation applications, which operate systems that perform industrial processes that are autonomous by nature and do not require human intervention (e.g., power system automation), and factory automation, that instead leverages robotic systems to substitute human operators in processes that originally required them (e.g., manufacturing process automation) [9]. Another notable example is the Internet of Medical Things (IoMT), which makes use of sensors, usually embedded in wearable devices, to analyze different healthcare-related metrics of a patient and offer them services, such as artificial intelligence-based diagnostics [10]. The interest in these fields is increasing in recent years, for example, the IIoT market is expected to reach an investment of $276.79 billion by 2029 [11]. Nonetheless, as the nature of the real-world processes is critical in their context, IIoT and IoMT applications are also considered critical. As the processes they automate are crucial for the user's mission (e.g., the manufacturing process of a factory, patient monitoring), intensive IoT applications must meet both the functional requirements of the user and non-functional requirements that are generally related to their performance, consumption, or cost [9, 10]. These non-functional requirements are as important to the application's utility as the functional ones, and failing to meet them results in the application becoming less useful, or even useless (e.g., if the factory automation application is slow, it would be unable to properly manage the manufacturing process, as its analyses and instructions would arrive too late at the robotic machinery) [9, 10]. For this reason, the concept of Quality of Service is crucial to intensive Internet of Things applications.

1.1.2 Quality of Service (QoS)

The International Telecommunication Union Telecommunication Standardization Sector (ITU-T) defines QoS as *the totality of characteristics of a service that bear on its ability to satisfy stated and implied needs of the user of the service* [12]. i.e., QoS refers to the different types of non-functional quality metrics that are provided by a given service, such as response time, reliability or energy consumption. Services that provide a good level of quality, such as services that are very reliable, fast or consume very little energy, are considered to have *high QoS*[1] [12]. Similarly, slow or unreliable services provide *low QoS* [12]. Furthermore, each use case of an application has a minimum QoS that must be met for it to work properly, which is labeled the *QoS*

[1]The concept of high QoS refers to how *good* a quality metric is, even if the value of the metric itself is better when it is low. For instance, an application with a response time of 1 ms has a *higher QoS* than one with a response time of 10 ms, although the metric value itself is lower because it is preferable.

requirement of such use case [12]. For instance, video streaming at 24 frames per second must have a response time under $\frac{1}{24}$ seconds in order to receive 24 frames each second. It is important to note that QoS can be seen from multiple points of view. While it is commonplace to only consider technical QoS, e.g., response time, it is also important to consider points of view such as business QoS, e.g., economic cost. Similarly to how the aforementioned video streaming service would not be useful if its response time was higher than $\frac{1}{24}$, if the economic cost of deploying the appropriate infrastructure exceeded the budget assigned to the service, it would not be useful either.

In the context of IoT for intensive domains, the criticality of the real-world processes is reflected by the IoT applications that automate them with very high QoS requirements. For instance, IIoT and IoMT applications require for short response times, which can be as low as 1 ms for the most critical applications [9, 10]. These QoS requirements are often labeled as *strict* or *stringent*, as high QoS is complex to provide, and such requirements leave a very small room for error. Moreover, these applications are also sensitive to *QoS degradation*, a phenomenon in which changes to the environment of the service (e.g., new versions of the application, network congestion, hardware malfunctions) negatively affect the QoS provided by the service. Providing and maintaining a high enough QoS is key to enabling the correct functioning of applications with strict QoS requirements.

As QoS is not a single metric, and it should rather be seen as a set of metrics obtained from multiple points of view, it is important to consider that these metrics often need to be traded off, balanced, prioritized, or ignored, depending on the exact needs of the company that makes use of the service [12]. For example, cost and response time are QoS metrics that need to be traded off: in general, it is possible to improve the response time by investing more money into the infrastructure (e.g., acquiring more devices, or more powerful devices). Nonetheless, a company may need the application to have the smallest economic cost, even if it implies that the response times will be on the boundary of acceptability (e.g., if 1 ms or lower response times are required, the response time may be very close to 1 ms). Similarly, another company may have the budget to invest as much as necessary for the application to have the lowest response times as possible. Furthermore, other companies may need to find a compromise solution that trades off both objectives, obtaining a relatively low economic cost with a relatively low response time, or perhaps prioritizing one of the objectives over the other. Hence, the consideration of multiple points of view for QoS can be deemed crucial to meet an application's requirements. Meeting the QoS requirements of intensive IoT applications, trading off the necessary elements to obtain the desired QoS from each point of view, is one of the main challenges that this next generation of IoT faces. Furthermore, achieving the required QoS in highly distributed environments is more difficult, due to the complexity of the high number of elements and interactions between them.

1.1.3 Service-Oriented Computing (SOC)

In the IoT environment, it is common for IoT devices to be integrated with the *things* themselves: smart coffee machines, smartwatches, smart TVs, and many other smart appliances often have the IoT devices integrated in them, a trend that also holds true in intensive domains [9, 10, 13]. As the final user is expected to acquire the *smart thing* itself, rather than separately, it is common for manufacturers to integrate low-capacity, low-power, and low-cost IoT devices into them to reduce the final price of the product. However, this complicates the use of some computationally expensive applications, such as complex AI models or high-capacity databases, as the IoT devices lack the power to run them [9, 10]. To enable the use of these applications, it is necessary to distribute the execution of the application among multiple devices. In this regard, the SOC paradigm is a reference for IoT applications, as it provides desirable features for the distribution of the application's execution [14].

SOC is a paradigm for the development of applications that considers *services* as the main building blocks of an application [14]. In this context, a service is an autonomous component of an application, that contains a part of the application logic, is generally autonomous and platform-independent, and represents a cohesive and loosely-coupled block of the application's functionality [14][2]. For instance, a traditional shopping application such as Amazon can be broken down into multiple cohesive and independent services, e.g., catalog, inventory, shopping basket, payment processor, and shipment tracker. Traditional applications that are architectured as a single module, and could thus be considered a single service from a SOC perspective, are known as *monolithic* applications. Nonetheless, techniques exist to extract multiple services from existing monolithic applications, converting them into applications that comprise multiple services [15]. SOC allows applications to be massively distributed, interoperable, and highly evolvable [14], desirable attributes for IoT applications.

Within SOC, one of the architectural patterns that can be applied is the Microservices Architecture (MSA), in which the application is split into very small services that must collaborate to carry out the application's functionalities [16]. The different modules of an MSA-based application, and a comparison with those of a classic monolithic application, are shown in Figure 1.2. As the services in MSAs tend to be small, i.e., they contain a small part of the application's functionality, they are known as *microservices*. MSAs emphasize the properties of SOC, as smaller services tend to also be more cohesive, more evolvable, and can potentially be deployed in a more distributed manner [14]. As some microservices tend to collaborate very often, they are considered to be highly coupled and are often grouped into larger modules, named *Bounded Contexts* or *APIs* [16]. While it is possible to deploy each of the microservices independently, all the microservices in the same API are usually deployed together [16]. However, the main drawback of MSAs is that, unlike monolithic applications, each microservice may be deployed multiple times (i.e., a microservice can be replicated, usually for load balancing or QoS purposes), in different machines, and may lack the knowledge of which services it should collaborate with [16]. Thus, MSAs usually require control modules that perform architecture-wide tasks. For example, aggregators that unify the outputs of multiple microservice replicas

[2]It is important to differentiate the concept of service in SOC, which refers to an application component, from the concept of service in QoS, which refers to the totality of systems, communications, hardware and software that are involved to provide certain functionality. From this point onwards, the term *service* will always refer to application components.

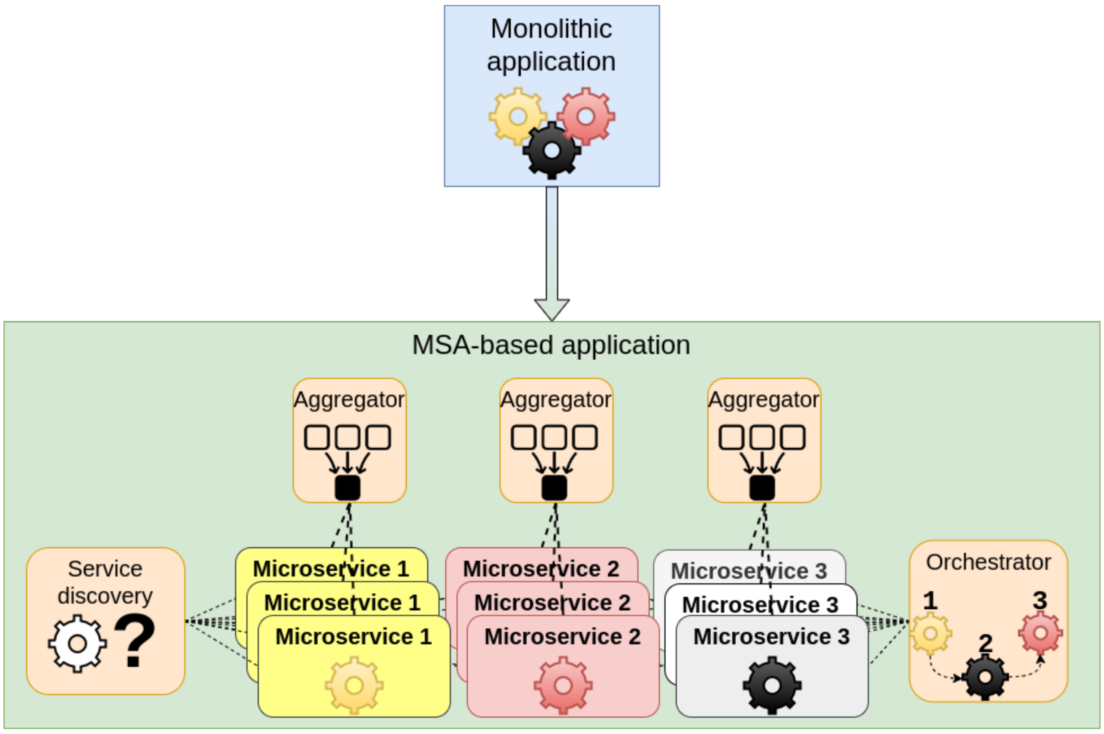


Figure 1.2: Modules in a monolithic application VS modules in an MSA-based application.

into a single data stream, service discovery modules that allow for requesters and other microservices to find the machine where a given microservice is deployed, or orchestrators that manage microservice collaboration.

Another interesting benefit to MSAs is that, in general, they can be easier to maintain than monolithic applications [14]. As microservices are only loosely coupled, it is possible to add new functionalities to an MSA-based application by simply implementing new microservices, with little to no changes to the existing MSA [14, 16]. New functionalities can also reuse the existing microservices to implement the parts of their functionality that overlap with the previously implemented application [16]. Furthermore, as long as the inputs and outputs of a microservice do not vary, it is possible to make changes to its implementation without affecting other microservices in the MSA, even if the unmodified microservices invoked the modified microservice [16]. This is especially desirable if application development practices such as DevOps [17] are used. DevOps is a methodology that focuses on

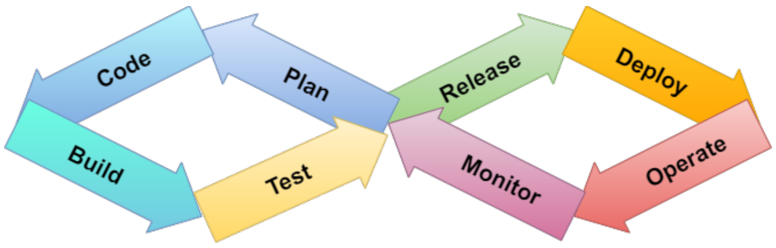


Figure 1.3: DevOps activities.

accelerating the delivery of features to final users, shortening the time between their implementation and their deployment to production [17]. To do so, DevOps establishes that the application developers (*Devs*) and operators (*Ops*) must collaborate, rather than *siloing* knowledge and development [17]. The process that is followed in DevOps for each feature is shown in Figure 1.3: each new planned and implemented feature is automatically built and tested. If the test provides positive results, it is integrated into the application's codebase and deployed to production. Moreover, the infrastructure is continuously monitored to ensure its correct functioning. As this process is followed continuously, MSAs tend to have less friction in terms of integration and deployment, as only the modified or new microservices may need to be re-released and re-deployed [16]. Hence, DevOps is considered an adequate methodology for MSA-based applications in general and IoT applications specifically [17].

Once the MSA is deployed, each microservice is offered through a given *interface*, which normally comprises a communication protocol and a data format required for communication. For instance, the SOAP protocol and the XML format [18], or HTTP and JSON [19]. As most interfaces are based on standard and open protocols and data formats, it is possible to request them using a wide variety of libraries, programming languages, and other software, making them interoperable among themselves and possible to request from many devices. It is especially interesting to develop IoT applications using an MSA, not only because of their properties in terms of evolvability and interoperability, but also because it allows IoT devices to request application functionalities with very thin, computationally light, and free clients (e.g., the CURL HTTP client), enabling heterogeneous low-capacity, low-power, and low-cost IoT devices to obtain the functionality of complex applications without the need to develop a specific IoT client from scratch. Therefore, this thesis focuses on MSA-based IoT applications.

Another interesting characteristic of microservices is that they can be deployed independently, as they are different, standalone application modules [16]. This allows the operator of an MSA-based application to decide in which machine should each microservice be deployed, based on the characteristics of the microservices, machines, or other organizational criteria. Furthermore, as each microservice can be replicated if so required, it is possible to deploy multiple replicas of a microservice in different machines, if it is desirable in that given situation. The focus of the thesis is aimed at the challenge of finding out how many replicas of each microservice are required to exist, as well as how to map each of the microservice replicas into the underlying network and computing infrastructure.

1.1.4 Cloud-to-Thing Continuum

Traditionally, IoT applications make use of *cloud computing* to execute their computational tasks [20]. The NIST defines cloud computing as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources [21]. This shared pool of resources, which is usually provided by multiple servers in a remote data center, is usually labeled as a *cloud server*, or *the cloud*. Furthermore, cloud computing is characterized by its *elasticity*: if the resources provisioned by the cloud are not enough to meet the demand in a given moment, it is possible to allocate more resources to the virtual pool to meet said demand [21]. Similarly, once the demand decreases to a normal level, the cloud is able to release these additional resources. The cloud computing paradigm has been one of the crucial enablers for IoT, as it has allowed the deployment of IoT applications to the cloud, allowing IoT devices to focus on sensing and acting while relegating the processing of information to the cloud, accelerating the deployment process and lowering its economic cost. Cloud computing is mainly offered through three service models:

Infrastructure as a Service (IaaS), *Platform as a Service* (PaaS), and *Software as a Service* (SaaS) [21], with each of them requiring less configuration and effort in the side of the consumer, but also allowing less control, as more of the infrastructure is managed by the cloud provider. For the context of this PhD thesis, we will focus on IaaS. In IaaS, the cloud provider offers the consumer virtual machines, provisioned from the resources of the general pool, and allows the consumer to control the software executed by these machines (e.g., operating system, installed libraries and programs, applications).

However, some problems stem from the necessity of communicating two distributed environments that are physically very far away from each other [13]. One of the main features of cloud computing is that the applications deployed into the cloud are executed in a remote data center, which is in the *network core* [21]. Especially in the case of IoT applications, it is complicated to obtain a very high QoS due to the distance between the IoT devices and the data center. For example, if we take Amazon Web Services' data centers, the closest one to Croatia is the Milan data center [22]. Thus, if an IoT application developed for Croatia were to be deployed at Amazon Web Services, all the requests from IoT devices would need to be sent to Milan, processed in the cloud, and the result sent back. These long-distance communications take longer than communications within Croatia, and can be affected by more problems (e.g., while a network problem within Croatia would affect the application's QoS similarly if it was deployed in Croatia, problems in the north of Italy will have an effect on QoS, that would not exist if it was deployed in Croatia). This is acceptable for user-grade IoT applications, but using exclusively a cloud computing deployment for intensive IoT applications can prove complicated due to the stringent QoS requirements they have [13].

To enable the deployment of intensive IoT applications, it would be desirable to make use of a paradigm that ran their critical microservices closer to the devices themselves, while sharing the key traits of cloud computing, such as ubiquity, on-demand provision, or ease of management. Some of the most interesting proposals from academia and industry are a series of computing paradigms, that can be referred to under the umbrella terms *Cloud-to-Thing Continuum*, *Computing Continuum* or *Cloud Continuum*, such as mist computing, edge computing, or fog computing [13, 23]. The main idea behind these proposals is to complement cloud computing by deploying some of the key microservices closer to the IoT devices [13]. Mist computing proposes to have some of the microservices be executed at the IoT devices themselves [23], whereas edge computing allows them to execute at devices named *edge nodes*, which are located within the access network, at one hop of the device, such as Wi-Fi, 4G or 5G access points [23]. Fog computing proposes to host some microservices at different points in the network, allowing for multiple layers of *fog nodes* to be used [13], with layers that contain more devices in parts of the network that are closer to IoT devices and less populated layers on points closer to the network core. It is important to note that these paradigms do not propose to use closer devices instead of the cloud, they propose to use them alongside the cloud: critical microservices with strict QoS requirements that do not require excessively powerful hardware can be executed in close nodes, whereas non-critical microservices can be executed in the cloud. Thus, by applying these paradigms, the *gap* between the IoT device (or, abstractly, the *thing* which is an embedded IoT device) and the cloud is bridged, and computations are performed continuously on the network between them. Hence, these paradigms are referred to as the *Cloud-to-Thing Continuum*. An example Cloud-to-Thing Continuum infrastructure that makes use of all the mentioned paradigms is depicted in Figure 1.4: at the bottom, the mist layer integrated by IoT devices can be used to execute some of the least demanding microservices. The mist is connected, through a single network hop, to the edge layer, which may execute some more powerful microservices. The edge can comprise some small servers, user computers, or even mobile devices[3]. The next part of the infrastructure represents a two-layered fog computing environment, where the lower layer contains more, smaller servers, while the upper layer contains

[3]The mobile devices are considered part of the edge because they provide microservices to IoT devices. If the final devices themselves are mobile, they should be considered to be in the mist layer, rather than in the edge.

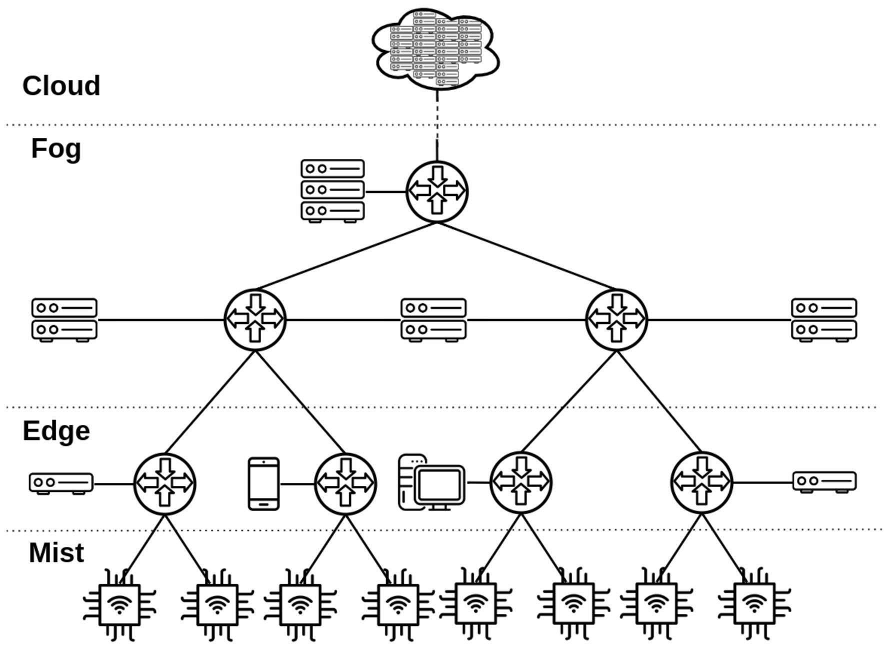


Figure 1.4: Example Cloud-to-Thing Continuum infrastructure.

a single, more powerful server. Finally, a single, vastly powerful cloud server is considered to be the furthermost, and most powerful, part of the infrastructure. The challenge of optimally selecting which devices, paradigms, and layers are used in the final Cloud-to-Thing continuum infrastructure to host IoT applications is a key tenet of this thesis.

1.1.5 Software-Defined Networking (SDN)

Similar to how the traditional computing paradigm for IoT applications is cloud computing, the network paradigm that is commonly used is IP [24]. The nature of the network is fundamental to IoT, especially if parts of the application run on edge, fog, or cloud devices, as it integrates the communication fabric through which they send and receive real-world information. Legacy or IP networks are comprised of *routers*, devices that are able to send traffic between networks. The tasks of the network fabric are generally divided into two planes: the *data plane* and the *control plane*. The data plane is in charge of forwarding information: as a unit of information or *packet* arrives at a router through a port, it should be sent through the appropriate output port [24].

On the other hand, the control plane decides the routing, i.e., it selects the path that flows of packets from a given source to a given destination should take, including the routers it should go through and the output ports that the packets should be forwarded to on each router [24]. In IP, each of the routers implements both the control and data plane, and thus, it must determine which port to output packets to and forward the traffic accordingly. Routing is performed by creating a forwarding table on each router, which describes which port to forward a packet through based on its destination address. This forwarding table is usually created by using *link-state algorithms* such as OSPF, which have a global view of the network and route using the shortest path approach [24]. The data plane part of the router is then in charge of correctly applying the forwarding table to incoming traffic.

However, IP networks have some shortcomings when they are applied to IoT applications, especially if they are MSA-based applications deployed through the Cloud-to-Thing Continuum. First, although link-state algorithms require a global view of the network, every router must obtain this view, which is inefficient [24]. Ideally, it should be possible for a centralized entity to calculate the forwarding tables for the complete network and send them to each router, which would save work for each of the routers and require less traffic to be sent. While it is possible to use decentralized algorithms that do not require global knowledge to calculate routing tables (distance-vector), they suffer from convergence problems such as the count to infinity [24], which can interrupt the provided service. Moreover, IP forwarding tables use the destination address as the only criterion, therefore forcing all packets with the same destination to be sent through the same path. Nonetheless, it is desirable in modern networks to perform *traffic engineering*, for instance, sending traffic directed to the same destination through multiple paths to balance network load and avoid congestion. Additionally, intensive IoT applications require for network features such as flexibility or virtualization features [25], which are not supported by IP. Finally, changing the criteria used for the shortest path algorithm or changing the routing algorithm, in general, requires a complex upgrade in every router of the network.

To address these problems, it is possible to use the SDN paradigm [26], instead of IP. SDN proposes to centralize the control plane in an entity named the *SDN controller* [26], although it is worthy of note that multiple coordinated SDN controllers may exist in a network [27]. The network devices, called *SDN switches*, are exclusively data plane devices, dedicated to forward traffic based on the *flow tables* they have installed. It is important to note that the flow tables allow for *generalized forwarding*: an SDN switch can use multiple criteria (e.g., source address destination address, protocol, size, source port, destination port) to determine the appropriate port to forward traffic to [26]. Moreover, SDN switches can perform three data plane actions on a packet: they can forward it to a port, they can delete the packet (drop), or they can modify some of its information before forwarding it [26]. When a new packet arrives at an SDN switch, it tries to find a matching entry in the flow table. If no match is found, they send the packet to the SDN controller in a PACKET-IN message. The SDN controller must then analyze the packet and determine what should be done with it, responding with a PACKET-OUT message to the switch, which explains whether to forward, drop or modify and how so. If the same action should be performed with every packet according to one or more criteria (e.g., to all packets with the same source and destination addresses), the controller can also send a FLOW-INSTALL message to add the action and matching fields as a new entry to the flow table [26]. This complete process is known as *SDN flow setup*, or simply *flow setup*.

Another important feature of SDN is that the SDN controller is programmable [26, 28]. Once the SDN controller receives a PACKET-IN message, the logic that controls the PACKET-OUT response to send, whether to send a FLOW-INSTALL and what matching and action fields should be in it is not defined by a particular protocol or algorithm, and the network administrator is free to program said logic. By using SDN controller frameworks, such as ONOS [29] or POX [30], it is possible to implement applications that define the behavior of the network, also known as *network-level applications*. These applications are not only in control of flow setups, but they can also perform other tasks such as network monitoring, and multiple applications may run in the same SDN controller [28]. This feature of SDN is known as *network programmability*.

In the context of intensive IoT applications, SDN addresses the shortcomings of IP networks [28]: the control plane is centralized, having a desirable and efficient global view of the network, generalized forwarding enables traffic engineering to be performed and SDN controllers can be customized through network-level applications to provide flexibility and virtualization. For MSA-based applications, SDN can be even more interesting, as some control modules such as aggregators, service discovery, or compositors, can be implemented as network-level applications [28]. Implementing the control modules at the network level has two main advantages. First, as the control is performed by network equipment with specialized hardware (e.g., TCAM) with better performance than general-purpose computing hardware, the QoS provided by network-level control modules is higher [28]. Second, this allows for a clear separation of concerns: the application is only concerned about what data is processed and how (i.e., microservice definition and collaboration descriptions), and the network is only concerned about how information must flow between machines hosting different microservices (i.e., MSA control modules and traditional network tasks) [28]. Determining the number of controllers to be placed in an SDN network used as the communication fabric for IoT applications, as well as optimally deciding on their placement, is a challenge that must be considered to achieve an optimal QoS.

1.2 Problem statement

Bringing automation and digitalization to intensive domains is one of the key objectives of the next generation of IoT [25]. These next-gen IoT applications are characterized by their stringent QoS requirements [9, 10, 13, 25] and their MSA-based design [25, 31]. Due to these characteristics, the Cloud-to-Thing Continuum and SDN are key enablers of next-gen IoT applications [13, 28]. Nonetheless, from a deployment perspective, obtaining a high QoS depends on how these paradigms are used and configured. For instance, if all the microservices of the application were to be deployed to the cloud, the QoS of a Cloud-to-Thing Continuum deployment for that application would be very similar to the QoS obtained with a pure cloud computing deployment, as the mist, edge, and cloud layers would be unused. Similarly, if due to the configuration (e.g., controller placement) the network provides a poor QoS, even if the microservices are placed properly, the QoS of the communication between different microservices will be very poor, and therefore, the QoS of the system will be low. Thus, to obtain an optimal QoS, three dimensions must be optimized: the application dimension, the computing dimension, and the networking dimension. Furthermore, the concrete characteristics of the paradigms leveraged, i.e., MSA, Cloud-to-Thing Continuum, and SDN, respectively, must be considered during such optimization. The problem to solve is the following:

Problem statement

Which deployment decisions of the application, computing, and networking dimensions in a scenario, which make use of the MSA, Cloud-to-Thing continuum, and SDN paradigms, respectively, should be optimized, and how should they be optimized, in order to obtain an optimal QoS?

The problem tackled by this thesis is thus a joint problem between three dimensions. Nonetheless, to maintain a holistic approach, the joint problem cannot merely be the sum of three individual problems, it must also consider the effects of a decision in each problem for the other two, effectively coordinating the decisions in all three dimensions with the objective of optimizing the QoS. To give a clear overview of the problem, we present it from a bottom-up view. First, we will analyze the decisions that exist on each separate dimension, i.e., the optimization problems that arise on each of the dimensions. Once each of the three problems is presented, we tackle the description of the relationships and effects of the decisions on each dimension over the rest of the dimensions.

On the application dimension, there are two key decisions that must be optimized. On the one hand, how many replicas of each microservice should be deployed?

Deploying additional replicas allows the application to be more distributed, providing multiple points of access, balancing the load among the different replicas, and potentially enhancing QoS metrics such as response time. However, each new replica takes resources from the devices in the infrastructure, and different microservices may require a different amount of resources per replica, which negatively affects QoS metrics such as battery consumption or the economic cost due to the need for a resourceful infrastructure. On the other hand, each replica must be deployed to a computing device. In this case, there is a key trade-off to be considered: the distance to the IoT device and the computing power. As represented by Figure 1.5, the lower layers (i.e., mist, edge) are closer to IoT devices and able to provide lower latencies and higher location awareness. However, the lower layers are also less powerful and resourceful in terms of computation: IoT devices generally lack the amount of power and computational resources of the cloud, and similarly, edge nodes are not as resourceful as cloud data centers. This is abstractly depicted in Figures 1.4 and 1.5, where the upper layers are depicted to have more servers. The more resourceful a device is, the more microservices can be instantiated in it, while computational power directly impacts the execution time, one of the components of response time, a key QoS metric. The problem of calculating the number of replicas of each microservice that should be deployed, as well as the machine that they should be deployed to in order to optimize the achieved QoS is labeled in literature as the Decentralized Computation Distribution Problem (DCDP) [32]. Solving the DCDP is key to optimizing the QoS of the application dimension. Moreover, the decisions on how many replicas of each microservice should exist and where to deploy them is one of the key challenges to achieving optimal QoS for MSA-based applications.

Regarding the computing dimension, due to the criticality of the applications, it is not unusual to add self-hosted computing devices to the infrastructure, usually at the lower layers of the continuum [33, 34]. In this regard, the operators of the system have control over where these fog or edge nodes are placed, especially when the application is to be used in large geographical areas, such as large industrial facilities [35].

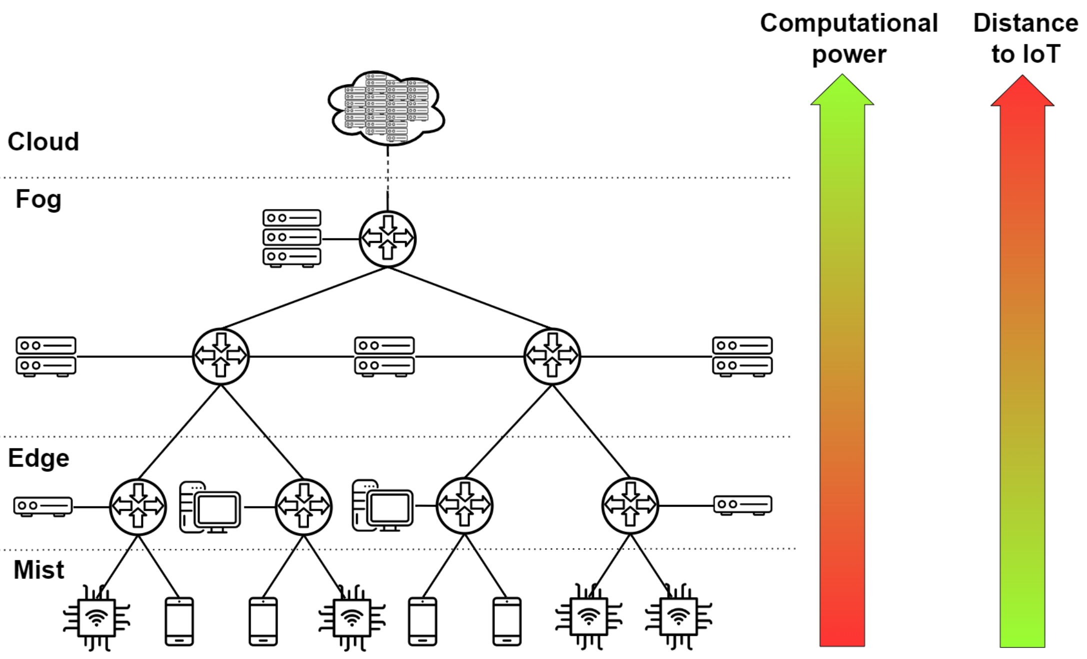


Figure 1.5: Computational power vs. distance trade-off in the Cloud-to-Thing continuum.

Nonetheless, the placement of the nodes is not trivial and has important effects on the QoS [35]. In this case, the amount of nodes is more clearly and directly related to QoS metrics like infrastructure cost, as the hardware of each node must be acquired and maintained. At the same time, placing more nodes gives the infrastructure more resources to deploy microservices on, affecting technical QoS metrics. Moreover, where each of the nodes is placed also affects the technical QoS: the closer a node is to the IoT devices, the better the QoS will be for such nodes, at the cost of a smaller *coverage* (i.e., other nodes will be further away, and thus, will not have a high enough QoS to properly make use of it) [36]. On the other hand, placing fog nodes in more central points of the infrastructure allows more devices to exploit its capabilities, but the QoS will not be as high as if the node was deployed closer to them [35, 36]. The problem of optimally determining the amount and placement of self-hosted nodes in Cloud-to-Thing Continuum infrastructures is known as the Fog Node Placement Problem (FNPP) [37][4], and its solution is crucial for the QoS in

[4]Despite the name specifically refers to fog computing due to the lack of standard terminology, the FNPP also applies to nodes deployed in any other layer of the Continuum.

the computing dimension. Moreover, as FNs may be placed at various layers of the infrastructure, solving the FNPP is directly related to the challenge of selecting which layers in the Cloud-to-Thing continuum should be used to optimize the QoS of a given scenario.

Finally, regarding the networking dimension, the QoS of the network, and hence, the QoS of the distributed system, is directly related to the QoS of the communications between each SDN switch and its controller [27]. Whenever a message that does not match any of the installed rules arrives at an SDN switch (e.g., when the system is first started and no rules are installed in any switch), the switch and the controller must communicate to perform a flow setup [38]. For example, if flow setups are slow due to poor switch-controller QoS, the whole network will suffer from poor QoS, as the incoming traffic has to wait until the corresponding flow setup is performed for the switch to handle it. This is known as *control QoS*, and must also be optimized [27]. Analogously to computing devices, the QoS between switches and controllers is directly related to the amount of SDN controllers deployed (more controllers negatively impact on QoS metrics such as cost and positively on metrics such as latency and response time) and their placement in the network, which is also directly related with the SDN switches assigned to the controller (e.g., the communication QoS will be better if each switch can be assigned to a controller in the same area of the network, rather than if control traffic needs to traverse the whole network) [38]. The problem of optimally placing controllers is the SDN Controller Placement Problem (CPP) [27, 38], and is a crucial challenge that must be tackled for QoS optimization in SDN.

Traditionally, the DCDP, the FNPP, and the CPP have been treated as separate problems, that are optimized independently [3, 27, 39]. While this is acceptable for traditional infrastructures, where the operators only hold control of a single dimension, in next-gen IoT environments, all three dimensions can be jointly controlled. Thus, it is important to consider the relationships between the problems on all three dimensions. The DCDP depends on the available computing resources and their placement [39], which is the solution of the FNPP, and the QoS of the network [39], which depends on the CPP solution. Similarly, solving the FNPP implies knowledge of how the computing resources will be used, i.e., which microservices will be deployed in them [3], which is the DCDP solution; as well as on the network QoS between the different nodes [3], that can only be known after the CPP is solved. Finally, the CPP requires knowledge of the traffic flows of the network [27], which are created by the application and depend on how it is placed (DCDP), as well as which nodes must communicate with each other and where they are placed [27] (FNPP). Hence, the problem that is tackled in this thesis is the problem that merges the DCDP, the FNPP, and the CPP as a single, joint optimization effort, from a holistic point of view.

1.3 Goals and research questions

Considering the presented research context and the stated problem, the main goal of this PhD Thesis is as follows:

Goal

Development of a framework for the optimization of the deployment of next-gen IoT applications, to jointly optimize multiple QoS metrics and holistically optimize the application, computing, and networking dimensions.

To achieve this goal, this thesis attempts to respond to five key Research Questions (RQs). These RQs motivate the different contributions presented in this document, and each work made in the context of the thesis is related to one or more RQs. These RQs are as follows:

RQ1 Are the decisions taken by QoS optimizations performed in each separate dimension different from those taken with a holistic approach? This RQ is the core motivation of this PhD Thesis, as it marks the relevance of its main novelty. Before the inception of this thesis, it is unclear whether the additional information provided by the three dimensions affects the decisions made in each of them, or if a locally optimal optimization of each dimension also leads to globally optimal decisions.

RQ2 Is it possible to jointly consider multiple points of view on QoS? One of the foundations of this thesis is the definition of QoS as multi-perspective, not only as multiple technical metrics but also as metrics from non-technical domains such as business. In some cases, these metrics may have to be traded off, as they are non-orthogonal (i.e., changes to affect one metric positively may affect other metrics negatively). We hypothesize it is possible to consider them jointly and trade them off if the operator wishes to optimize multiple QoS metrics.

RQ3 Which models and techniques are useful for solving the joint optimization problem? There are models that describe the optimization models for each separate dimension, that need to be merged into a single problem model. Furthermore, optimization problems can be solved using a variety of techniques with different properties, such as different computational complexities, optimality gaps, or multi-objective support.

RQ4 Can the decisions taken during optimization adapt to changes in the dimensions' environments? Once the deployment of a next-gen IoT application is optimized, it is natural for changes to occur in each of the dimensions. The application may change over time with new versions, that may implement additional features or change the existing ones, affecting the properties of the defined microservices. The devices in the computing dimension may differ in their capabilities due to changes in their operating systems or changes in hardware. Networking equipment may be subject to errors or congestion, changing the available routes. Thus, it is key to research if it is possible to adapt this optimization over time to maintain optimal QoS.

RQ5 How does this optimization fit in current development lifecycles and practices? The motivation for developing a QoS optimization framework is

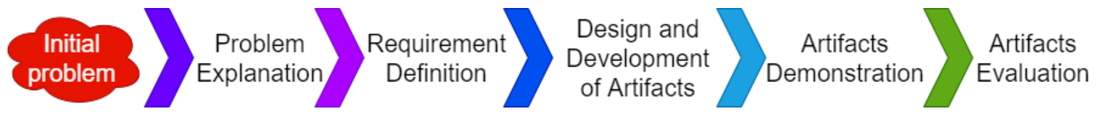


Figure 1.6: Design Science iterative structure.

to enable the development of next-gen IoT applications by providing application operators and developers with tools that help them in this task. For these tools to be truly helpful, we must consider in which phase or phases of the application development lifecycle they should be used, for example, which models and techniques are more useful for the design phase due to their characteristics, or which ones are more suitable for the runtime phase. Furthermore, how to integrate them into existing development practices, such as DevOps, should also be addressed.

1.4 Research methodology

The methodology used throughout the development of this PhD thesis is *Design Science* [40], a methodology normally applied to research projects in the disciplines of Engineering and Computer Science [41]. The objective of Design Science is to obtain knowledge of a problem and its domain, to then create artifacts, such as pieces of software, that mitigate said problem [40]. By using Design Science, the researcher is expected to act not only as an observer of the problem but also as a designer of artifacts and products for solving it. Design Science defines five steps that must be followed, as depicted in Figure 1.6. In the following, we define each of the five steps in the context of this thesis:

1. *Problem explanation*. In this phase, the research problem that will be addressed must be defined, justifying the value of the solution. In this thesis, the problem is the holistic optimization of QoS for next-gen IoT. This problem was identified through a literature review, where related literature addressed the problem in one of the dimensions rather than considering them jointly. A joint solution would

be interesting in this regard, as it would consider the effects that the decisions of each dimension affect the QoS of the other dimensions. This would enable developers and operators to obtain higher QoS, making possible applications that would originally be hardly feasible due to their high QoS requirements.

2. *Requirement definition*. This phase involves the definition of the objectives that must be completed to solve the identified problem. The main objective of this thesis is to provide a framework for the holistic optimization of QoS during deployment. This entails five sub-objectives, embodied by the defined RQs (Section 1.3).

3. *Design and Development of Artifacts*. The third phase involves the creation of one or multiple artifacts that address the defined objectives. This thesis has involved the development of a total of 14 artifacts in 8 related lines. Out of these 14 artifacts, we highlight the main artifact, the Distributed Application Deployment Optimization (DADO) framework, initially presented in [2]. DADO originally optimized microservice replication and placement, as well as SDN controller placement, and has been extended throughout different works to tackle the defined RQs. Moreover, an artifact to optimize QoS through fog node placement was also developed [37], which was eventually merged with DADO and created the *Umizatou* framework [42]. These were complemented with artifacts such as Continuous DADO (ConDADO), which is an adaptive version of the DADO framework for dynamic environments and for its integration with DevOps.

4. *Artifacts Demonstration*. In the fourth phase, the developed artifacts are tested to show their capabilities with regard to solving the defined problem. In this thesis, this stage was mainly done as a preliminary assessment of the developed artifacts before proceeding to the formal evaluation.

5. *Artifacts Evaluation*. The final phase involves measuring how well the artifacts solve the problem, and how they compare with the existing state of the art.

DADO and its related artifacts have been tested in case studies from multiple intensive environments such as IIoT, IoMT, or IoT video surveillance. Moreover, they have been tested against naïve approaches to their respective problems, as well as against state-of-the-art benchmarks that solved the problems of each of the dimensions independently. This testing included the obtention and comparison of metrics such as network load, hardware load, optimization times, or the concrete QoS metrics that were optimized.

The activities in Design Science do not have to be followed in a strictly sequential manner: often, Design Science is applied iteratively. While Figure 1.6 shows the relationship between activities in terms of inputs and outputs, it does not necessarily represent the sequence of activities. Therefore, authors such as Hevner [43] establish a three-cycle view of Design Science as shown in Figure 1.7:

- The *Relevance Cycle* initiates research by analyzing the environment and the problem to be addressed, defining the requirements for the artifacts and, therefore, the acceptance criteria for artifact evaluation. The need for additional iterations to solve the problem is also identified in this cycle.

- The *Design Cycle* focuses on building and evaluating the developed artifact. The activities in this cycle are guided by the requirements and criteria obtained in the Relevance cycle, and these activities are performed by applying the methods and techniques identified in the Rigor cycle.

- The *Rigor Cycle* provides knowledge to the project to ensure it is innovative, thus ensuring that the performed research contributes to the general knowledge base.

Design Science produces both artifacts and knowledge of general interest. Therefore, it presents three additional requirements: use of rigorous research methods, original and grounded results, and communication of results. During the development of this thesis, the research strategy leveraged to plan the work was Design Science,

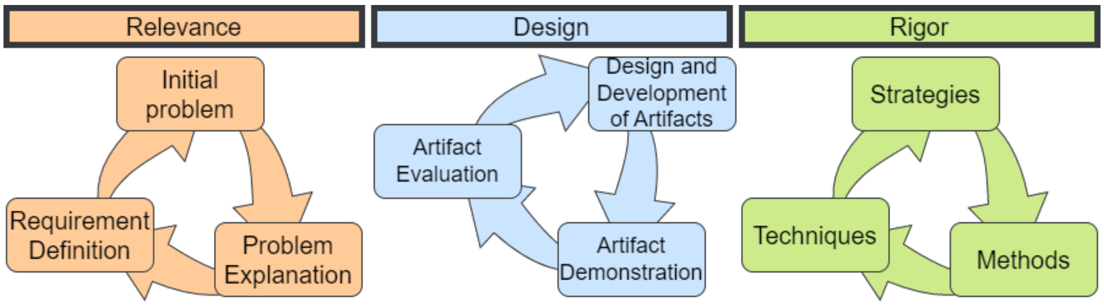


Figure 1.7: Proposed model for the Design Science methodology.

while the research methods used to obtain and analyze data are based on the same methods that related, state-of-the-art works make use of. Finally, case studies are used to validate and demonstrate the artifacts. Furthermore, as it has been applied iteratively, some of the 14 developed artifacts are still within the Design Cycle and, as a result, do not yet have an associated publication

1.5 Contributions

This section serves as a summary of the main contributions of the research work that has been performed as part of this thesis. These contributions are reflected in multiple publications in scientific journals, conferences, and workshops, as well as developed software artifacts.

1.5.1 Summary of Contributions

In pursuing the goal of this thesis, different contributions, which can be classified under three main, closely related, research lines, were achieved. For each of these lines, we provide in the following a context of the line, as well as a discussion of the problems addressed, the solutions proposed for them, and the publications that derived from these contributions. A summarized version of the contributions related to each research question can be seen in Table 1.1.

Research Question	**Associated publications**	**Associated artifacts**
RQ1: Are the decisions taken by QoS optimizations performed in each separate dimension different from those taken with a holistic approach?	[**GLOBECOM 2021, LATINCOM 2021, JISBD 2021, JCIS 2021, JITEL 2021, IoTJ 2021, ICC 2022, PERCOM 2022, JCIS 2022, IoTJ 2022-2**]	DADO, NIoTO, Umizatou, MO-SFO
RQ2: Is it possible to jointly consider multiple points of view on QoS?	[**JISBD 2021, ICC 2022, PERCOM 2022, JCIS 2022, IoTJ 2022-2**]	NIoTO, MO-SFO
RQ3: Which models and techniques are useful for solving the joint optimization problem?	[**ISCC 2020, GLOBECOM 2020, ICC 2021, GLOBECOM 2021, LATINCOM 2021, JISBD 2021, JCIS 2021, JITEL 2021, IoTJ 2021, ICC 2022, PERCOM 2022, JCIS 2022, IoTJ 2022-1 IoTJ 2022-2**]	FNPP MILP, FNPP heuristic, DADO, NIoTO, MO-SFO, ConDADO, S-DADO
RQ4: Can the decisions taken during optimization adapt to changes in the dimensions' environments?	[**PERCOM 2022, CISTI 2022, Computing 2022**]	ConDADO, S-DADO
RQ5: How does this optimization fit in current development lifecycles and practices?	[**PERCOM 2022, CISTI 2022, Computing 2022**]	ConDADO

Table 1.1: Contributions related to each research question.

Holistic optimization of QoS in IoT application deployment

Context The central topic of this thesis is the holistic approach to QoS optimization for the deployment of IoT applications. To holistically make deployment-related decisions, its three main dimensions must be considered: application, computing, and networking. Contextually, this research line involves considering the characteristics of the paradigms used in the three dimensions (MSAs, the Cloud-to-Thing Continuum, and SDN, respectively), as well as how they affect the relationship between the decisions taken in each dimension and the environment in the other two. This view was developed throughout different works that build upon this holistic approach.

1. *Problem statement*: The DCDP and the CPP have been traditionally considered as two separate problems, as application operators generally did not have control in the network and vice-versa. However, as there is an increasing interest in using the Cloud-to-Thing Continuum for the deployment of intensive IoT applications, operators are more likely to have control over both, the application and the network. While this could allow for further optimization by coordinating the decisions taken in both problems, there was a lack of systems that consider this problem. Due to its complexity, it was not yet known whether this coordination could lead to QoS improvements, nor how the decisions were related.

   Contribution: We studied the relationship between the DCDP and the CPP in one of the seminal works of this thesis. Based on this relationship, we developed the initial version of DADO, which fused both SDN controller placement and microservice placement into a single optimization effort. Moreover, DADO has a distinct feature in how it considers the networking dimension, by also optimizing the routing of both application and control traffic. The evaluations performed in this initial work exhibit that coordinating the decisions on both dimensions by creating a joint model allows it to take specific decisions, based on the knowledge from the other dimension, that improve the QoS over separate optimization. This initial contribution was published in the *IEEE Internet of Things Journal* (2021) [2].

2. *Problem statement*: As operators now hold control of multiple dimensions, they can also decide where their computational nodes will be placed. Hence, it must be considered whether this placement has an impact on QoS, and which metrics had it an impact on. If the impact existed, there would be a need for an optimization model on computing node placement.

Contribution: As we noticed this new area of control, we defined the FNPP as another relevant problem in IoT application deployment. Continuing the holistic view, we studied the impact of computing-related metrics, such as the capacity of the computing nodes, as well as network-related metrics, such as the size and routes followed by traffic. We also formally modeled the FNPP, declaring the decisions that should be taken for it to be optimized (traffic routing, node placement, IoT-computing node mapping). The contributions to the FNPP as a standalone research line include communications in the *IEEE Symposium on Computers and Communications (ISCC 2020)* [36], the *IEEE Global Communications Conference (GLOBECOM 2020)* [37], the *IEEE International Communications Conference (ICC 2021) [44]*, and a publication in the *IEEE Internet of Things Journal (2022)* [3].

3. *Problem statement*: The FNPP may also have relationships with the DCDP and CPP, as the decisions taken in the computing dimensions also affect the application and networking dimensions. There was a lack for a system that optimized all three problems jointly.

   Contribution: Noticing the gap in the holistic view of the optimization, we developed a new version of DADO (labeled *Umizatou*), which integrated the FNPP along with the CPP and the DCDP. This integration greatly changed the previous optimization models, both for DADO and for the FNPP, as some decisions were completely dependent on the decisions of other dimensions, instead of merely being conditioned. Umizatou was presented in communication at the *IEEE Global Communications Conference (GLOBECOM 2021) [42].*

Optimization of multiple QoS metrics

Context: Another key tenet of this thesis is the consideration of QoS from multiple points of view. This includes not only allowing for the optimization of different QoS metrics, such as response time or deployment cost but also allowing multiple metrics to be optimized at the same time. This implies that the system needs to consider the trade-offs that may appear if the different QoS metrics are non-orthogonal (i.e., if obtaining a better result on one metric requires obtaining a worse result in the other one).

1. *Problem statement*: Considering multiple objectives in QoS optimization for IoT application deployment. It is especially interesting to solve this problem if the optimization is performed using a holistic view.

   Contribution: Throughout the development of the different frameworks for QoS optimization, we have aimed at extending them to consider multiple QoS objectives, especially if the objectives are non-orthogonal, as it makes our frameworks consider the additional trade-offs. An initial version of multi-objective DADO was presented in a communication at the national conference *Jornadas de Ingeniería de Software y Bases de Datos (JISBD 20/21)* [45]. This version was later expanded to become a new version of DADO (labeled NIoTO, *Next-generation IoT Optimization*), and published in the *IEEE Internet of Things Journal (2022)* [5]. Furthermore, continuing the trail of Umizatou, a new version of DADO that considers the FNPP as well, and allows for multi-objective optimization, which has been labeled MO-SFO (*Multi-Objective SDN-Fog Optimization*), has been developed. A communication describing MO-SFO has been submitted to the *IEEE International Communications Conference (ICC 2023)*, although at the time of writing this document there is not yet a decision on the manuscript.

Adaptativeness of QoS-related decisions at runtime

Context: Another important aspect of these systems is their ability to adapt their decisions to changes in the scenario. These include changes to the application dimension (e.g., an increase in requests, a new version of the application with new features, the addition of more microservices, the deployment of additional applications), to the computing dimension (e.g., an operating system update that changes the available resources of a device, the addition of new devices, changes to their hardware, server downtime), and to the networking dimension (e.g., decreases in bandwidth, congestion, changes to latency, addition of new network equipment, packet loss, network equipment downtime). Adaptativeness comprises two main factors: change awareness and reactiveness. The system's change awareness represents how the system understands change: the least change-aware systems consider each new situation as a completely new scenario, with no regard for the previous situation, while more change-aware systems are able to tell which parts of the system have changed and compare multiple situations. Reactiveness refers to the time the system requires to adapt its decisions to a new situation, i.e., the optimization time of the system. The faster a system is, the quicker it is able to react to changes, and hence, it is deemed *more reactive*.

1. *Problem statement*: The change-awareness of systems that perform QoS optimization for application deployment is mostly small and unexplored.

   Contribution: We have developed a system that treats change awareness as a first-class concern for adaptativeness. This system is an extension of DADO, and it is called *Continuous DADO* or simply *ConDADO*. ConDADO makes use of continuous reasoning, a technique that analyzes the changed elements in a scenario to only re-optimize the parts of the scenario that require it. This also shrinks the problem, allowing it to be up to 4 times more reactive than DADO. ConDADO is currently under review, after undergoing a major revision, in the journal *Computing* (2023). Moreover, we have also developed another continuous reasoning-based system that is more reactive than ConDADO, named Faustum. This system is meant to be used along ConDADO and DADO to adapt to changes of different kinds in a stack we call *multi-layered continuous reasoning*. We are currently working on a paper that encompasses

Faustum and multi-layered continuous reasoning.

2. *Problem statement*: The adaptability of these systems is yet to be explored using a stochastic approach.

   Contribution: We have developed a system, currently named Stochastic DADO (S-DADO), to adapt the replication of microservices, as well as their placement, based on Lyapunov optimization [46]. This kind of optimization allows the system to make sure the QoS requirements of certain tasks are met on average making use of virtual queues. S-DADO is also significantly faster than DADO, and hence, it is more reactive. We are currently working on a paper on S-DADO, that compares the Lyapunov optimization approach to other, non-stochastic systems.

3. *Problem statement*: The systems described in this thesis are centralized, but it would be desirable to have a decentralized system, especially for mobile computing optimization and for its use in opportunistic networks.

   Contribution: We have developed a framework for the delegation of services in mobile and pervasive computing named PODS (Pervasive Opportunistic Delegation of Services). PODS comprises a set of modules that allows for microservices offered from devices such as mobile phones, IoT devices, edge nodes or fog nodes, to determine how many replicas of the microservice should exist, as well as which device should run them, in a decentralized manner. Furthermore, PODS regularly adapts the placement using a modified version of DADO, meant to be more reactive, small, and aimed at its use in opportunistic networks: μDADO. μDADO maintains the multi-objectiveness of DADO, but reduces the size of the problem for the opportunistic and pervasive context. We are currently working on a paper to communicate our results with PODS in the migration of Docker containers between Android devices.

1.5.2 Publications

In this subsection, we present a complete list of the publications derived from the research work carried out during the PhD, in chronological order. The publications listed here include those that are directly related to the main topic of the PhD thesis, i.e., that directly address one of the RQs and are part of the research lines described in subsection 1.5.1, as well as publications that are closely related (e.g. publications about IoT, QoS optimization, the Cloud-to-Thing Continuum) but do not address the RQs nor are part of the main research line. Along with each publication, we detail the quality ratings of the venue they were published in. For conference papers, we detail the conference's scope (national or international), as well as its rating according to the *GII-GRIN-SCIE (GGS)* conference rating. For journal articles, we present the journal's *Impact Factor (IF)*, as well as the quartile and category of the journal according to the *Journal Citation Reports (JCR)*. A summary of all international journal publications ordered by their rating is offered in Tab. 1.2, while national conferences and workshop publications are summarized in Tab. 1.3. Journal articles, ordered by their JCR quartile and IF, are shown in Tab. 1.4. Finally, a summary image of every publication, including the information about the RQs it tackled, its venue of publication, the quality metrics of the venue, the year, and whether it is directly or closely related to the thesis, is included as Figure 1.8.

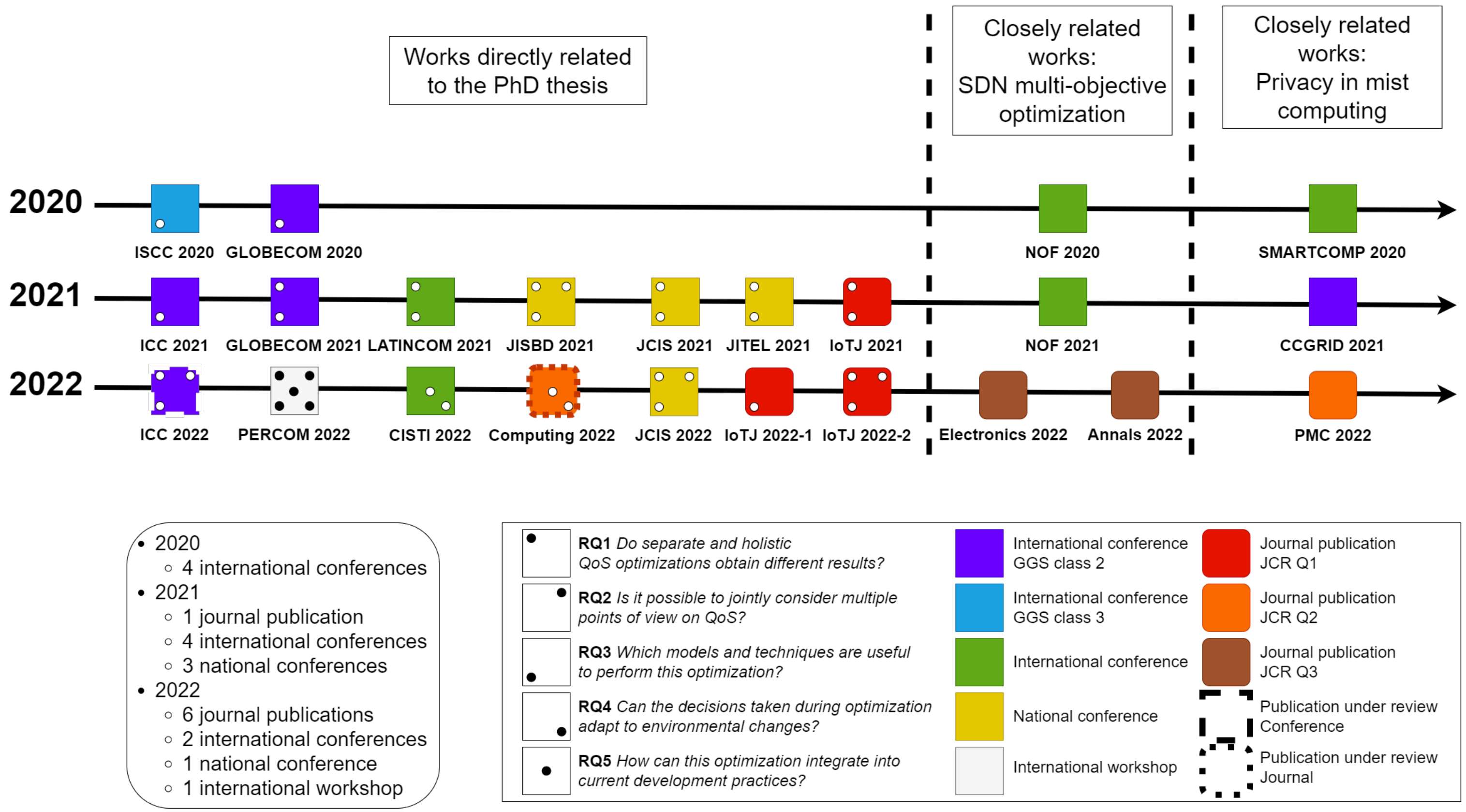

Works directly related to the PhD thesis
Closely related works: SDN multi-objective optimization
Closely related works: Privacy in mist computing
2020
ISCC 2020
GLOBECOM 2020
NOF 2020
SMARTCOMP 2020
2021
ICC 2021
GLOBECOM 2021
LATINCOM 2021
JISBD 2021
JCIS 2021
JITEL 2021
IoTJ 2021
NOF 2021
CCGRID 2021
2022
ICC 2022
PERCOM 2022
CISTI 2022
Computing 2022
JCIS 2022
IoTJ 2022-1
IoTJ 2022-2
Electronics 2022
Annals 2022
PMC 2022
2020
4 international conferences
2021
1 journal publication
4 international conferences
3 national conferences
2022
6 journal publications
2 international conferences
1 national conference
1 international workshop
RQ1 Do separate and holistic QoS optimizations obtain different results?
RQ2 Is it possible to jointly consider multiple points of view on QoS?
RQ3 Which models and techniques are useful to perform this optimization?
RQ4 Can the decisions taken during optimization adapt to environmental changes?
RQ5 How can this optimization integrate into current development practices?
International conference GGS class 2
International conference GGS class 3
International conference
National conference
International workshop
Journal publication JCR Q1
Journal publication JCR Q2
Journal publication JCR Q3
Publication under review Conference
Publication under review Journal

Figure 1.8: Summary of the publications related to this thesis.

2020

During the start of the PhD and the research activities, we prepared the ground for addressing **RQ1** (*Are the decisions taken by QoS optimizations performed in each separate dimension different from those taken with a holistic approach?*), as it is the core motivation of the thesis. In order to answer **RQ1**, we first needed to, at least partially, address **RQ3** (*Which models and techniques are useful for solving the joint optimization problem?*): once there is a development on the techniques and modeling of the problem, it is possible to develop a prototype of it and test whether the decisions made separately and jointly are different. The research made during this year was mainly focused on three research topics. The one directly related to the thesis is the topic of the FNPP, which led to two publications in international conferences [36, 37]. The other two, which are closely related, are about multi-objective QoS optimization in SDN networks, with a publication in an international conference [47]; and about security in mist computing, with another publication in an international conference [48].

- **ISCC 2020** [36]: Juan Luis Herrera, Luca Foschini, Jaime Galán-Jiménez, Javier Berrocal: *The Service Node Placement Problem in Software-Defined Fog Networks*. Published in IEEE International Symposium on Computers and Communications (ISCC). **International conference, GGS Class 3 (B)**.

- **GLOBECOM 2020** [37]: Juan Luis Herrera, Paolo Bellavista, Luca Foschini, Jaime Galán-Jiménez, Juan Manuel Murillo, Javier Berrocal: *Meeting Stringent QoS Requirements in IIoT-based Scenarios*. Published in IEEE Global Communications Conference (GLOBECOM). **International conference, GGS Class 2 (A-)**.

- **NOF 2020** [47]. Jaime Galán-Jiménez, Javier Berrocal, Juan Luis Herrera, Marco Polverini: *Multi-Objective Genetic Algorithm for the Joint Optimization of Energy Efficiency and Rule Reduction in Software-Defined Networks*. Published in the International Conference on Network of the Future (NoF).

International conference.

- **SMARTCOMP 2020** [48]: Juan Luis Herrera, Javier Berrocal, Juan Manuel Murillo, Hsiao-Yuan Chen, Christine Julien: *A Privacy-Aware Architecture to Share Device-to-Device Contextual Information*. Published in IEEE International Conference on Smart Computing (SMARTCOMP). **International conference**.

2021

In the second year of the PhD, the main focus changed slightly: while there was still some work on **RQ3**, the efforts were mainly aimed at answering **RQ1**. The question was finally answered in our first journal article in the context of this thesis [2]: DADO is able to take informed decisions that would be impossible without the joint view. This also led us to fuse the existing works of DADO with the FNPP to create Umizatou, a system that considered all three dimensions [42]. This year's research was focused on four main research topics. The first two are directly related to this thesis: one involves further work on the FNPP as a standalone problem, obtaining a publication in an international conference [44]; the other one focuses on the optimization of deployments with a holistic view, obtaining a journal publication [2], two publications in international conferences [42, 49] and three publications in national conferences [45,50,51]. The other two continue from the year prior: multi-objective QoS optimization in SDN networks, with a publication in an international conference [52]; and about access control and privacy in mist computing, with another publication in an international conference [53].

- **ICC 2021** [44]: Juan Luis Herrera, Paolo Bellavista, Luca Foschini, José García-Alonso, Jaime Galán-Jiménez, Javier Berrocal: *Fog Node Placement in IoT Scenarios with Stringent QoS Requirements: Experimental Evaluation*. Published in IEEE International Communications Conference. **International conference, GGS Class 2 (A)**.

- **GLOBECOM 2021** [42]: Juan Luis Herrera, Jaime Galán-Jiménez, Paolo Bellavista, Luca Foschini, José Manuel García Alonso, Juan Manuel Murillo, Javier Berrocal: *Optimal Deployment of Fog Nodes, Microservices and SDN Controllers in Time-Sensitive IoT Scenarios*. Published in IEEE Global Communications Conference (GLOBECOM). **International conference, GGS Class 2 (A-)**.

- **LATINCOM 2021** [49]: Guilherme Werneck de Oliveira, Rodrigo Toscano Ney, Juan Luis Herrera, Daniel Macêdo Batista, Roberto Hirata, Jaime Galán-Jiménez, Javier Berrocal, Juan Manuel Murillo, Aldri Luiz dos Santos, Michele Nogueira: *Predicting Response Time in SDN-Fog Environments for IIoT Applications*. Published in IEEE Latin-American Conference on Communications (LATINCOM). **International conference**.

- **JISBD 2021** [45]: Juan Luis Herrera, Jaime Galán-Jiménez, José García-Alonso, Javier Berrocal, Juan Manuel Murillo: *Despliegue Óptimo de Aplicaciones IoT Distribuidas*. Published in Jornadas de Ingeniería del Software y Bases de Datos (JISBD). **National conference**.

- **JCIS 2021** [50]: Juan Luis Herrera, Jaime Galán-Jiménez, Javier Berrocal, Juan Manuel Murillo: *Optimizing the Response Time in SDN-Fog Environments for Time-Strict IoT Applications (Summary)*. Published in Jornadas de Ciencia e Ingeniería de Servicios (JCIS). **National conference**.

- **JITEL 2021** [51]: Juan Luis Herrera, Jaime Galán-Jiménez, Javier Berrocal, Juan Manuel Murillo: *Optimizing the Response Time in SDN-Fog Environments for Time-Strict IoT Applications*. Published in Jornadas de Ingeniería Telemática (JITEL). **National conference**.

- **NOF 2021** [52]: Jaime Galán-Jiménez, Marco Polverini, Francesco Giacinto Lavacca, Juan Luis Herrera, Javier Berrocal: *On the tradeoff between load balancing and energy-efficiency in hybrid IP/SDN networks*. Published in the International Conference on Network of the Future (NoF). **International conference**.

- **CCGRID 2021** [53]: Juan Luis Herrera, Hsiao-Yuan Chen, Javier Berrocal, Juan Manuel Murillo, Christine Julien: *Privacy-Aware and Context-Sensitive Access Control for Opportunistic Data Sharing*. Published in the IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID). **International conference, GGS Class 2 (A)**.
- **IoTJ 2021** [2]: Juan Luis Herrera, Jaime Galán-Jiménez, Javier Berrocal, Juan Manuel Murillo: *Optimizing the Response Time in SDN-Fog Environments for Time-Strict IoT Applications.* Published in IEEE Internet of Things Journal. **JCR Q1 (*Computer Science, Information Systems*) IF: 10.238**.

2022

The third year of the PhD was marked by a focus on the three remaining RQs: **RQ2** (*Is it possible to jointly consider multiple points of view on QoS?*), **RQ4** (*Can the decisions taken during optimization adapt to changes in the dimensions' environments?*), and **RQ5** (*How does this optimization fit in current development lifecycles and practices?*). Once **RQ1** was answered in the year prior, the work could be expanded with multi-objective considerations, adaptive versions, and how DADO could be integrated into the development methodologies that are currently used. Multi-objective QoS was a research topic on its own, while RQ4 and RQ5 were worked on jointly. Along with these two research lines, a third directly related research line, the FNPP line, was also in development. These three research lines were combined with the ongoing research lines on multi-objective QoS optimization in SDN networks, as well as access control and privacy in mist computing. On the topic of multi-objective optimization, we created NIoTO, a version of DADO that could optimize the deployment cost along with the response time, which obtained a publication in a national conference [54] and in a journal [5]. Moreover, a new version of Umizatou, called MO-SFO, was also developed, bringing multi-objective considerations to Umizatou. A publication on MO-SFO is currently undergoing review at an international conference. On the second topic, we developed an architecture meant to integrate deep reinforcement learning-based agents into production environments while further refining them, the Deep QoS Adaptive Learning Environment (DeQALE), published in an international conference [55]. The core work, nonetheless, is ConDADO, an adaptive version of DADO, along with Continuous Adaptation (CA), a proposal for the integration of QoS optimization and adaptation frameworks in DevOps. A publication detailing CA and ConDADO is currently undergoing review after a major revision in a journal. Spanning the two research topics at once, a summary of the objectives and development of the thesis at the moment was presented in the PhD Forum of an international conference [4]. On the third research line, the FNPP, the adaptiveness of the solution was improved through the development of a heuristic based on machine learning techniques. This new improvement was published in a journal [3]. Regarding the fourth research line, two publications in journals were obtained [56, 57]. Finally, a paper detailing a privacy-aware access control system including a formal model is currently under review after a minor revision in a journal.

- **CISTI 2022** [55]: Juan Luis Herrera, Javier Berrocal, Jaime Galán-Jiménez, José García-Alonso, Juan Manuel Murillo: *Quality of Service-Adaptive Industrial Internet of Things leveraging Edge Computing and Deep Reinforcement Learning: The Deep QoS-Adaptive Learning Environment (DeQALE) Architecture*. Published in IEEE Iberian Conference on Information Systems and Technologies (CISTI). **International conference**.

- **JCIS 2022** [54]: Juan Luis Herrera, Jaime Galán-Jiménez, José García-Alonso, Javier Berrocal, Juan Manuel Murillo: *Joint Optimization of Response Time and Deployment Cost in Next-Gen IoT Applications (Summary)*. Published in Jornadas de Ciencia e Ingeniería de Servicios (JCIS). **National conference**.

- **PERCOM 2022** [4]: Juan Luis Herrera, Javier Berrocal, Juan Manuel Murillo: *Deploying Next Generation IoT Applications Through SDN-Enabled*

Fog Infrastructures. Published in the PhD Forum of IEEE International Conference on Pervasive Computing and Communications (PERCOM). **Workshop associated with an international conference, GGS Class 1 (A+)**.

- **ICC 2022**: Juan Luis Herrera, Jaime Galán-Jiménez, Paolo Bellavista, Luca Foschini, José García-Alonso, Juan Manuel Murillo, Javier Berrocal: *Multi-Objective Optimal Deployment of SDN-Fog Infrastructures and IoT Applications*. Under review in IEEE International Communications Conference. **International conference, GGS Class 2 (A)**.

- **IoTJ 2022-1** [3]: Juan Luis Herrera, Jaime Galán-Jiménez, Luca Foschini, Paolo Bellavista, Javier Berrocal, Juan Manuel Murillo: *QoS-Aware Fog Node Placement for Intensive IoT Applications in SDN-Fog Scenarios*. Published in IEEE Internet of Things Journal. **JCR Q1 (*Computer Science, Information Systems*) IF: 10.238**.

- **IoTJ 2022-2** [5]: Juan Luis Herrera, Jaime Galán-Jiménez, José García-Alonso, Javier Berrocal, Juan Manuel Murillo: *Joint Optimization of Response Time and Deployment Cost in Next-Gen IoT Applications*. Published in IEEE Internet of Things Journal. **JCR Q1 (*Computer Science, Information Systems*) IF: 10.238**.

- **Electronics 2022** [56]: Manuel Jiménez-Lázaro, Juan Luis Herrera, Javier Berrocal, Jaime Galán-Jiménez: *Improving the Energy Efficiency of Software-Defined Networks through the Prediction of Network Configurations*. Published in Electronics. **JCR Q3 (*Computer Science, Information Systems*) IF: 2.690**.

- **Annals 2022** [57]: Jaime Galán-Jiménez, Marco Polverini, Francesco G Lavacca, Juan Luis Herrera, Javier Berrocal: *Joint energy efficiency and load balancing optimization in hybrid IP/SDN networks*. Published in Annals of Telecommunications. **JCR Q3 (*Telecommunications*) IF: 1.901**.

- **PMC 2022** [58]: Juan Luis Herrera, Hsiao-Yuan Chen, Javier Berrocal, Juan

Manuel Murillo, Christine Julien: *Context-Aware Privacy-Preserving Access Control for Mobile Computing*. Published in Pervasive and Mobile Computing. **JCR Q2 (*Computer Science, Information Systems*) IF: 3.848**.

- **Computing 2022**: Juan Luis Herrera, Javier Berrocal, Stefano Forti, Antonio Brogi, Juan Manuel Murillo: *Continuous QoS-Aware Adaptation of Cloud-IoT Application Placements*. Under review after a major revision in Computing. **JCR Q2 (*Computer Science, Theory and Methods*) IF: 2.420**.

ID	Reference	Venue	Rating
ICC 2021	[44]	IEEE ICC	GGS Class 2 (A)
ICC 2022	Under review	IEEE ICC	GGS Class 2 (A)
CCGRID 2021	[53]	IEEE/ACM CCGRID	GGS Class 2 (A)
GLOBECOM 2020	[37]	IEEE GLOBECOM	GGS Class 2 (A-)
GLOBECOM 2021	[42]	IEEE GLOBECOM	GGS Class 2 (A-)
ISCC 2020	[36]	IEEE ISCC	GGS Class 3 (B)
NOF 2020	[47]	NoF	N/A
SMARTCOMP 2020	[48]	IEEE SMARTCOMP	N/A
LATINCOM 2021	[49]	LATINCOM	N/A
NOF 2021	[52]	NoF	N/A
CISTI 2022	[55]	IEEE CISTI	N/A

Table 1.2: Publications in international conferences.

ID	Reference	Venue	Type
JISBD 2021	[45]	JISBD	National conference
JCIS 2021	[50]	JCIS	National conference
JITEL 2021	[51]	JITEL	National conference
JCIS 2022	[54]	JCIS	National conference
PERCOM 2022	[4]	IEEE PERCOM	Workshop

Table 1.3: Publications in national conferences and workshops.

ID	Reference	Journal	JCR & IF
IoTJ 2021	[2]	IEEE IoT Journal	**Q1** (IF: 10.238)
IoTJ 2022-1	[3]	IEEE IoT Journal	**Q1** (IF: 10.238)
IoTJ 2022-2	[5]	IEEE IoT Journal	**Q1** (IF: 10.238)
PMC 2022	[58]	Pervasive and Mobile Computing	**Q2** (IF: 3.848)
Computing 2022	Major revision	Computing	**Q2** (IF: 2.420)
Electronics 2022	[56]	Electronics	**Q3** (IF: 2.690)
Annals 2022	[57]	Annals of Telecommunications	**Q3** (IF: 1.901)

Table 1.4: Publications in journals.

1.5.3 Developed artifacts

Many of the contributions to the state of the art presented in this thesis have been accompanied by software artifacts and tools, which were used to perform some of the conducted research, and can also be used to reproduce it. Concretely, we highlight 5 main artifacts, that are presented as the *Relevant publications* in the dissertation. Nonetheless, we want to clearly show that the work of the thesis has included the development of a total of 14 software artifacts along a total of 8 *lines*. 9 of these artifacts have publications associated with them, while the remaining 5 do not yet have a publication associated (i.e., they currently are in the *Design and Development* or **Artifact Demonstration** phases of the Design Science methodology). Furthermore, there are some additional supporting artifacts (e.g., DADO's QoS reporting tools, the DADOJSON and DADOSIM file formats) that are not counted among the 14. Figure 1.9 depicts each of the artifacts and the lines developed for the thesis. In the following, we develop a small summary of each line and artifact:

- *DADO main line*: this line represents a core part of this thesis, as it presents DADO [2] and NIoTO [5], the holistic and, in the case of NIoTO, multi-objective frameworks for the optimization of QoS. Following them, there is Vernier, an artifact named after the *Vernier thrusters* of rockets, which is an implementation of a heuristic named Adaptive Kernel Search [59]. Vernier allows for MILP programs to be solved using Adaptive Kernel Search, as long as all the variables of the problem are binary. While Vernier, as the name suggests, was initially meant as a work to accelerate DADO, it can be used outside of the scope of DADO as a general implementation of a heuristic.

- *FNPP main line*: this line is another key component of the thesis, as it presents the artifacts necessary to solve the FNPP. The first artifact [3, 36, 37, 44] makes use of MILP as a means to solve the FNPP, and could thus be integrated with DADO. On the other hand, the heuristic [3] makes use of a machine learning technique (K-medoids) to place fog nodes nearly optimally in a network topology.

- *DADO+FNPP line*: continuing with the core artifacts of this thesis, this line presents the integration of DADO with the FNPP solver, as proposed in [3]. This initially created Umizatou [42], an all-in-one holistic optimization framework. Umizatou was further developed by adding NIoTO's multi-objective orientation, creating MO-SFO [**ICC 2022**].

- *Continuous reasoning line*: the main line proposed for the adaptation of service placement over time, which also allows DADO's adaptation to be included in DevOps, are the artifacts contained in this line. Continuous DADO (ConDADO) [**Computing 2022**] makes use of the continuous reasoning techniques that are already used in Continuous Integration [60], and applies them to the model of DADO, only re-optimizing the parts of the deployment that need it and mitigating the deployment costs. Moreover, to react very quickly to small changes in the scenario, we also developed Faustum, a reduced version of the model to deal with specific, small, and recurrent changes in a much faster manner.

- *Stochastic line*: as a means to allow DADO to adapt service placement over time, we have developed a new version named Stochastic DADO (S-DADO). This system is based on Lyapunov optimization [46], and balances the minimization of the average response time with keeping the response times of all the requests under the given deadlines. Moreover, the scenarios for S-DADO allow for it to be evaluated using probabilistic models for changing the scenario over time.

- *Emulation testbed line*: as a means to test, not only DADO but other similar approaches in the future using emulated network testbeds, this line presents a single artifact. Pascal is a system that, based on a file that declares the status of a scenario (e.g., network topology, characteristics of the computing devices, requests for functionalities, microservice, and SDN controller deployment), generates a set of configured Docker containers for the Kathará network emulator [61]. Pascal automatically generates the specifications for each container, the network topology, its performance characteristics, and configures all the network interfaces and the SDN controller, leaving the user with a nearly-ready testbed. Moreover, Pascal supports the use of both IP and SDN networks and is compatible with a wide variety of Docker images already available at Docker Hub.

- *Privacy in mist computing line*: this line includes two artifacts, namely, PADEC [53], and PFE-PADEC [58]. Privacy-Aware DEvice Communication (PADEC) is a framework that enables service-oriented applications to control access to their endpoints. Using PADEC, it is possible to define under which contextual conditions can an endpoint be accessed, as well as the granularity of the released information, using rules and filters that both users and developers can define and tailor. Furthermore, the privacy of the requesters is also considered, minimizing the information they must share. Private Function Evaluation PADEC (PFE-PADEC) allows for the evaluation of these access control rules using encrypted data, thus maintaining privacy on both sides. Both PADEC and PFE-PADEC were evaluated in opportunistic mist computing scenarios.

- *Opportunistic mist computing line*: merging the prior line with DADO, we obtain Pervasive Opportunistic Service Delegation (PODS), a framework that enables collaborative service delegation in mist computing. PODS is a complete framework that allows for a *community* of devices to execute collaborative services that can be requested by all the devices but can be hosted only in some of them. Inside PODS, we highlight μDADO, a reduced version of DADO that decides how many replicas of each service should exist and which devices should host them.

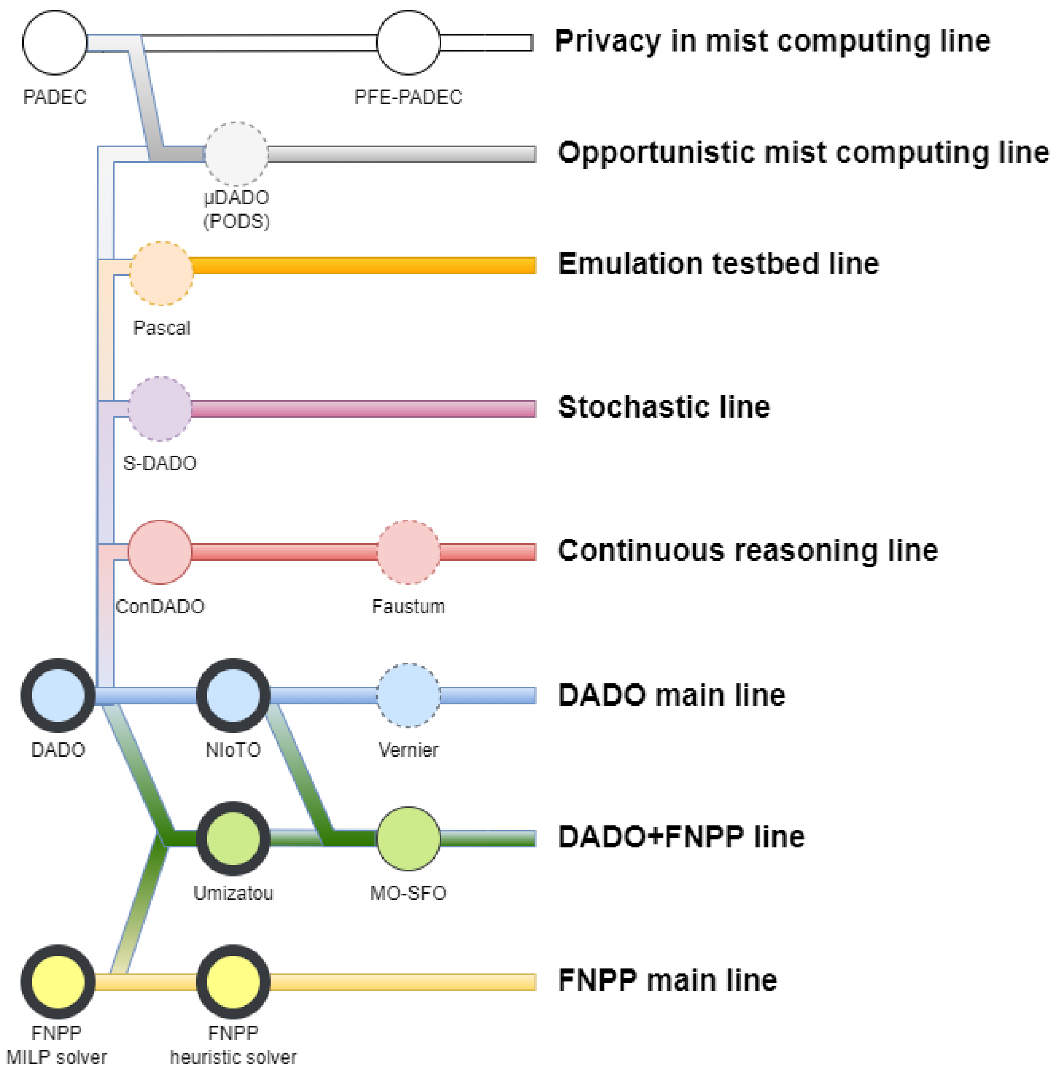


Figure 1.9: Artifacts and lines of artifacts developed for this thesis.

1.5.4 Collaborations

During the development of this PhD thesis, we have worked closely with both national and international research groups for the development of multiple artifacts and publications. These collaborations have not only been helpful because of the shared know-how and the additional points of view, but also for the closeness and fruitfulness of each of the collaborations. Concretely, we have collaborated with a total of 7 research groups in 4 countries, developing 11 artifacts and publishing 14 works in various conferences and journals. The location of the institutions that we have collaborated with is depicted by Figure 1.10, and Tab. 1.5 details, for each research group, its institution, country, and developed artifacts and publications[5].

We have also carried out an international research stay during the PhD, at the Department of Computer Science of the University of Pisa (*Dipartimento di Informatica, Università di Pisa (UniPi)*) at Pisa, Italy. This research stay was performed under the supervision of the renowned expert in the Cloud-to-Thing Continuum and SOC, the leader of the SOCC research group, Professor Antonio Brogi. During a total of three months, from March 28th, 2022 until June 28th, 2022, we explored the possibilities of continuous reasoning and their integration with DADO. In particular, we prepared the publication for the ConDADO artifact [**Computing 2022**], and we developed two additional artifacts, Faustum and Vernier. Moreover, we defined the concept of *multi-layered continuous reasoning*, as a system consisting of multiple artifacts for QoS optimization, arranged in a layered fashion. The upper layers of the system contain artifacts that are faster and specifically designed to deal with small and frequent changes. Conversely, the lower layers contain slower systems that can deal with a wider variety of situations, including deeper, yet less frequent, changes. After a change is detected in the environment, its depth is analyzed, and, if so required, an adaptation of the deployment plan is requested to the appropriate layer. Furthermore, if the selected layer is unable to deal with the change, the problem will be sent to the layer below, which is more likely to be able to address it. Faustum was born as an

[5]In some cases, the acronym *WIP* (Work In Progress) is shown for the cases in which we are still under the development of our first publication or artifact with a given research group.

artifact for the utmost layer of the system, while Vernier is an ongoing effort to speed up other systems. We currently are working on the publication of Faustum, Vernier, and the multi-layered continuous reasoning system.

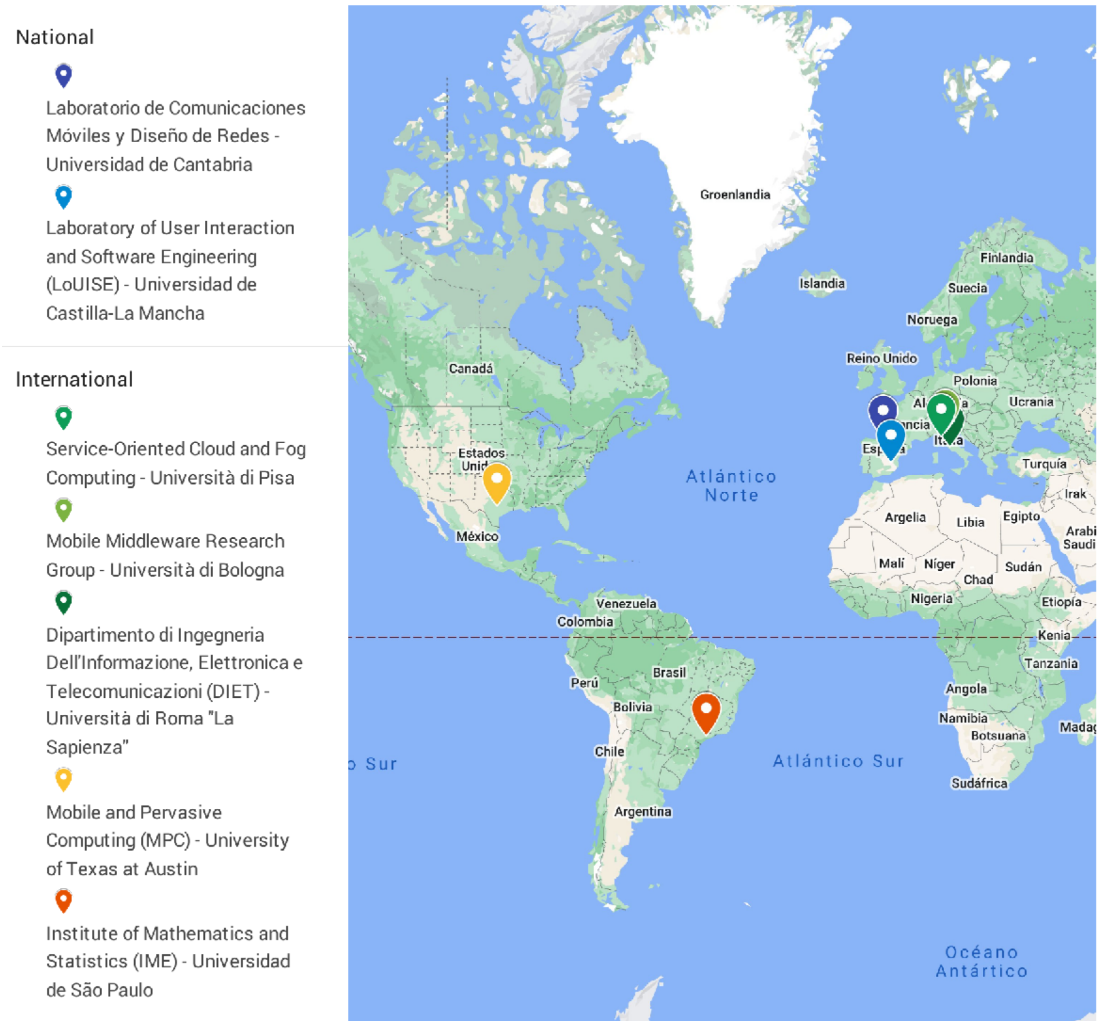

National
Laboratorio de Comunicaciones Móviles y Diseño de Redes - Universidad de Cantabria
Laboratory of User Interaction and Software Engineering (LoUISE) - Universidad de Castilla-La Mancha
International
Service-Oriented Cloud and Fog Computing - Università di Pisa
Mobile Middleware Research Group - Università di Bologna
Dipartimento di Ingegneria Dell'Informazione, Elettronica e Telecomunicazioni (DIET) - Università di Roma "La Sapienza"
Mobile and Pervasive Computing (MPC) - University of Texas at Austin
Institute of Mathematics and Statistics (IME) - Universidad de São Paulo
Groenlandia
Islandia
Finlandia
Suecia
Noruega
Reino Unido
Polonia
Ucrania
Canadá
Estados Unidos
Atlántico Norte
Turquía
Irak
México
Argelia
Libia
Egipto
Arabia Saudí
Mali
Níger
Chad
Sudán
Nigeria
Etiopía
Venezuela
Colombia
Kenia
Tanzania
Brasil
Perú
Bolivia
Angola
Namibia
Botsuana
Chile
Atlántico Sur
Sudáfrica
Argentina
Océano Antártico

Figure 1.10: Collaborations with research groups carried out as part of this thesis

Research group	University	Location	Artifacts	Publications
Laboratorio de Comunicaciones Móviles y Diseño de Redes	Universidad de Cantabria (UC)	Santander, Cantabria, Spain	S-DADO	WIP
LoUISE (Laboratory of User Interaction and Software Engineering)	Universidad de Castilla-La Mancha (UCLM)	Albacete, Castilla-La Mancha, Spain	WIP	WIP
SOCC (Service-Oriented Cloud and Fog Computing)	Università di Pisa (UniPi)	Pisa, Tuscany, Italy	ConDADO, Faustum, Vernier	[**Computing 2022**]
Mobile Middleware Research Group	Università di Bologna (UniBo)	Bologna, Emilia-Romagna, Italy	FNPP (MILP & heuristic), Umizatou, MO-SFO	[**ISCC 2020**, **GLOBECOM 2020**, **ICC 2021**, **GLOBECOM 2021**, **ICC 2022**, **IoTJ 2022-1**]
DIET (Dipartimento di Ingegneria dell'Informazione, Elettronica e Telecomunicazioni)	Università di Roma "La Sapienza" (UniRoma)	Rome, Lazio, Italy	N/A	[**NOF 2020**, **NOF 2021**, **Annals 2022**]
MPC (Mobile and Pervasive Computing)	University of Texas at Austin (UT)	Austin, Texas, United States of America	PADEC, PFE-PADEC, μDADO (PODS)	[**SMARTCOMP 2020**, **CCGRID 2021**, **PMC 2022**]
IME (Institute of Mathematics and Statistics	Universidad de São Paulo (USP)	São Paulo, São Paulo, Brazil	N/A	[**LATINCOM 2021**]

Table 1.5: Artifacts and publications per collaboration.

1.6 Outline

This thesis dissertation is organized into four main parts, each structured in chapters as follows:

- **Part I: Summary.** This part contains the two main chapters, including the Introduction (Chapter 1) and Chapter 2, and offers an overview of the main contributions of the thesis and its themes.
- **Part II: Selected Publications.** This part includes the core publications for the thesis. Chapter 3 presents an overview of the core and supporting publications, while Chapters 4 through 7 include full-text copies of the core publications that conform this compendium.
- **Part III: Final Remarks.** This part only comprises Chapter 8, which concludes the thesis dissertation with a discussion of the achieved results, the limitations of the presented work, and future research lines.
- **Part IV: Appendices**. The appendices include the supporting publications for this thesis (Appendices A through S) and documentation on the implementation of artifacts (Appendices T through AB), along with a glossary of the acronyms and terms used throughout the document (Appendix AC).

The detective is bound to declare
any clues which he may discover.

Knox's Decalogue, 8th Rule
Ronald A. Knox

Chapter 2

Results

In this chapter, we present a detailed view of the main research contributions of the PhD, grouped by their topic. Section 2.1 details the contributions related to the holistic optimization of QoS for the deployment of next-gen IoT applications. This is followed by Section 2.2, which introduces the contributions on the joint optimization of multiple QoS metrics at the same time for said deployment. Finally, Section 2.3 presents the work on the adaptation of application deployments at runtime. A summary of the impact of each contribution in the software development lifecycle of an application is provided as Figure 2.1

2.1 Holistic optimization of QoS in IoT application deployment

One of the main challenges of next-gen IoT applications is to obtain an acceptable, and desirably optimal, QoS [2, 4, 5, 9, 10, 37]. Modern application, computing, and networking paradigms, such as MSA [14, 16], the Cloud-to-Thing Continuum [13, 23], or SDN [24, 26–28], respectively, are crucial enablers for the obtention of a high QoS. However, these paradigms' high modularity and evolvability make them notably more complex to manage: the number of application modules, computing devices, and manageable networking elements is much higher than in legacy paradigms [13,

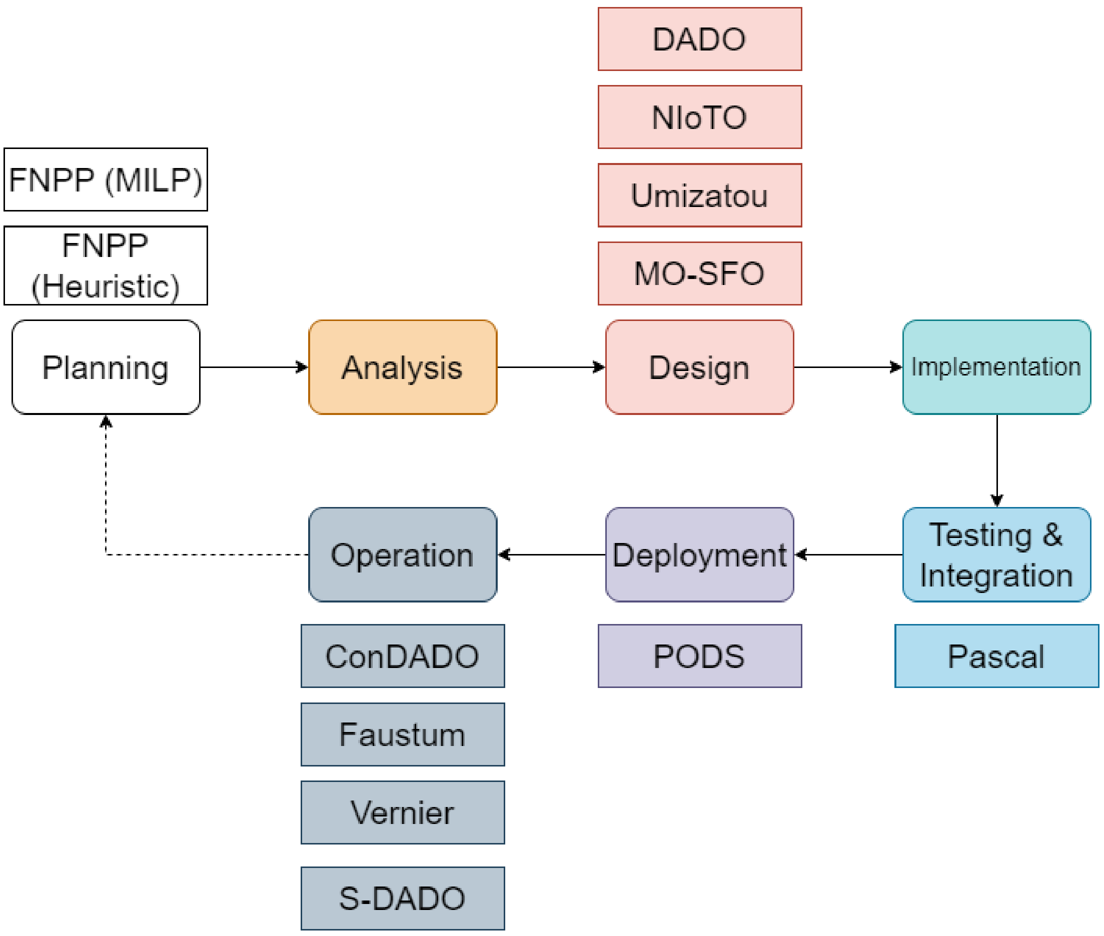


Figure 2.1: Contributions related to each of the activities in the software development lifecycle.

27, 39]. Moreover, the focus on location awareness and performing computing tasks closer to IoT devices gives operators more control over both their application and their infrastructure, allowing them to coordinate a joint optimization of multiple dimensions, improving the overall QoS [2].

In what follows, we describe our contributions to this coordinated, holistic optimization effort. Concretely, we present our results in terms of optimization of microservice deployment and SDN controller placement, our advances in the joint optimization of the computing and networking dimension, and our all-in-one optimization of all dimensions, as well as a summary of which system should a developer or operator use, based on their situation and needs.

2.1.1 Placing microservices and SDN controllers optimally

In order to understand the importance of the placement of microservices and SDN controllers, the model for both IoT applications and SDN control must be detailed. Regarding IoT applications, while it has been established that an MSA is used to architect the application, it should be possible for microservices to be composed to perform complex functionalities [16]. There are multiple models for this composition [14, 16], but we consider the *chained microservices* model [62] is used in IoT applications, which is commonly used in research literature [39]. With this model, IoT devices are expected to request *workflows*, which represent a pipeline or chain of microservices that must be executed. The order in which they are requested is important, as the input of each microservice is the output of the previous one in the workflow. Finally, the output of the final microservice of the workflow can optionally be returned to the IoT device that requested it, although it may stay at the deployment site of the final microservice (e.g., if the final service is a database that stores the information, there is no need to return a result to the device).

As an example of this model, we define an IoT video recognition application, following the model from [63]. For our example, we will define a workflow for performing the *facial recognition* functionality. This workflow will request three

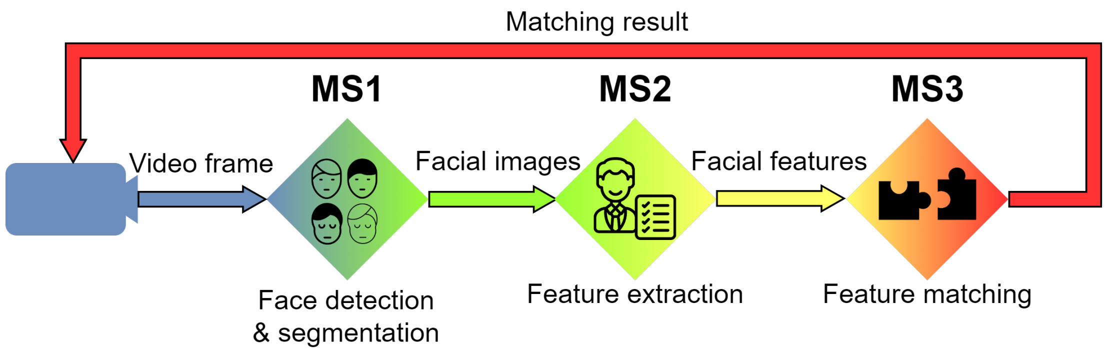
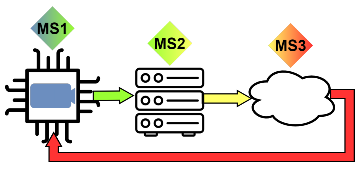


Figure 2.2: Example facial recognition workflow

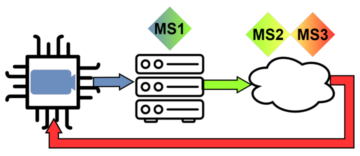
microservices [63]. First, the raw video stream is fed to a detection and segmentation microservice (**MS1**), which detects human faces in the video and removes other elements and the background. These facial images are then fed to the second microservice (**MS2**), which extracts features that are useful for facial recognition (e.g., the distance between eyes, mouth curvature [63]). Finally, the features are sent to a matching microservice (**MS3**), that compares them with an existing database of features. The final answer on whether the recognized faces are known or not is sent back to the requesting device. Figure 2.2 depicts this example workflow.

Following the example, let us have a very simple example infrastructure for this single workflow request. On the one hand, it is possible to run some microservices on the IoT device with the camera, i.e., in the mist layer [23]. On the other hand, a fog node also exists, close to the device, that can run microservices. Finally, it is also possible to host some microservices in the cloud. In this situation, there are a total of 27 (3^3) valid alternatives for placing the microservices[1]. Three of these alternatives are depicted in Figure 2.3: in all three devices in Figure 2.3a, using only the fog and the cloud in Figure 2.3b, and using exclusively the fog node in Figure 2.3c. Moreover, Figure 2.3 also represents communications: if a single device is leveraged (Figure 2.3c), only two messages are necessary: the initial video frame and the matching result, as all microservices placed in the same computing device

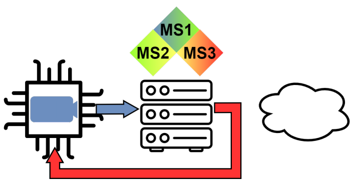
[1]In general for DADO, given the set H, which contains all the computing devices (hosts), the set WF, containing all workflow requests, and defining the length of a workflow request ($|wf|, wf \in WF$) as the number of microservices requested by the workflow, the total number of alternatives for microservice placement and replication can be calculated as $(\sum_{wf \in WF} |wf|)^{|H|}$.



(a) Placement along the continuum.



(b) Placement in the fog and the cloud



(c) Placement in the fog

Figure 2.3: Example deployment alternatives of the workflow

can communicate locally. This is also true if some microservices are executed in the IoT device itself (Figure 2.3a), as the device would feed the processed data of the microservice's output (in this case, facial images), rather than the original raw data. Finally, it is also important to note two crucial considerations on the feasibility of placements. First, a machine cannot run an unlimited number of microservices, only as many as it can withstand. This can be understood in terms of resources: computing devices have a certain amount of resources (e.g., CPU, RAM), and microservices consume these resources to function. Hence, while the more resourceful cloud or fog can run all three microservices, this is not feasible for the IoT device. The second consideration is that microservices run faster in more powerful machines. There are a variety of metrics and estimators for defining the computational size of a microservice, as well as the computational power of a computer. Nonetheless, to maintain the model as application-agnostic and device-agnostic as possible, we assess the computational size of a microservice as the number of CPU cycles it takes to run it. Similarly, the computational power of a device is assessed as its CPU clock speed. These assessments allow for the calculation of the execution time, one of the components of the response time, a crucial QoS metric for next-gen IoT applications.

However, the model of the example so far lacks a dimension: the network. Following the premises, the network paradigm for the example is SDN, and thus, the behavior of the network is managed by one or more SDN controllers through SDN flow setups. Following the classic model in research [27], each SDN controller is co-located with an SDN switch, i.e., at least one SDN switch serves both as a switch and as a controller. As the network is expected to perform service discovery [2, 28], the communication to each microservice is mediated by the SDN controller, which will set up the appropriate flows. Depending on where the SDN controller is placed, the flow setups can have a different QoS, as the switch that is directly connected to the SDN controller can perform it locally, while the other switches must send their requests across the network. Considering the deployment from Figure 2.3a, we show an example of workflow communications if a single SDN controller was placed in the leftmost switch in Figure 2.4. To perform this communication, the IoT device first executes MS1 and sends the facial images to the network (1). The switch must then request a flow setup to find MS2 (2). This flow setup leads the traffic to the fog node, where MS2 is deployed (3). The fog node extracts the features of the images using MS2 and sends these features to MS3 (4). Similarly, the switch performs service discovery to find MS3. This is done by, first, sending the request to the leftmost switch (5). This switch sends the request to the SDN controller, which performs the flow setup (6). The flow setup messages must then be sent back to the original switch (7) to complete the process. Once MS3 is located in the cloud, the feature information follows this path (8), and MS3 performs the matching operation. The result of this matching operation is then sent back to the IoT device using standard routing, which is assumed to be already installed (9 - 10). A similar model could be followed for every SDN controller placement, including having multiple SDN controllers, as well as for all 27 possible placements, which would lead to a total of 81 possible placements of microservices and SDN controllers[2]. Finally, an additional consideration that is crucial to the networking dimension is that the maximum capacity of each link must not be surpassed. To maximize the feasibility of the problem and guarantee that there is no overload, as well as to avoid congestion, it would also be desirable to optimize the traffic routing, i.e., allowing traffic to use the QoS-optimal routes if the capacity constraint holds while routing traffic through alternative routes if so required. Such considerations imply that, for each of the messages from Figure 2.4 except 2 and 6, a

[2]In general, if a given number of controllers c is provided, and assuming a set S with every SDN switch, the total number of possible different microservice and controller placements is $(\sum_{wf \in WF} |WF|)^{|H|} \binom{|S|}{c}$. If the number of SDN controllers is to be selected at runtime up to a maximum number c_{max}, as in DADO, the total number of combinations is of $(\sum_{wf \in WF} |WF|)^{|H|} (\sum_{i=1}^{c_{max}} \binom{|S|}{i})$

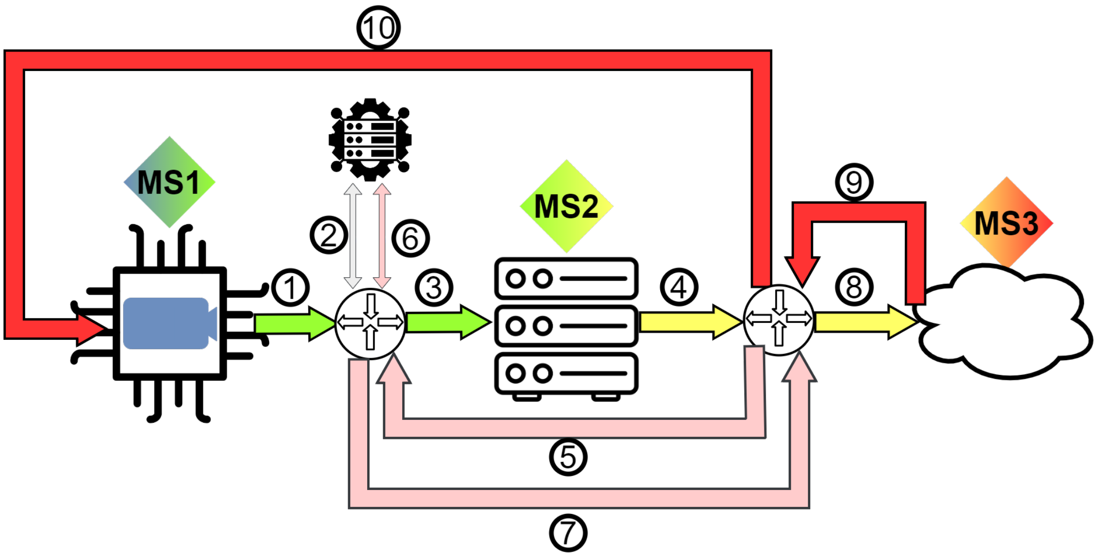


Figure 2.4: Example workflow deployment with controller placement

route must be chosen out of all the alternatives. Even assuming that only two routes exist for each of the messages, this would lead to a total of 256 routing alternatives, or an approximate total of 20736 alternatives[3].

Based upon this system model, where the computing dimension is considered for its maximum capacity to host microservices and its QoS characteristics (e.g., computing power), the application dimension is considered for its defined microservices, workflow requirements, and their characteristics (e.g., computing complexity, input/output information), and the networking dimension is considered for the SDN controller placement and its QoS characteristics (e.g., location of each device in the network, link latency, link capacity), we created the original DADO framework, which makes use of MILP solvers to optimize the deployment [2]. DADO requires two main inputs: a description of the IoT application and a description of the infrastructure. The IoT application description includes the microservice stereotypes that exist, the workflows that are requested, which devices (and therefore users) request each workflow, and the characteristics of each stereotype (e.g., resource consumption,

[3]As the number of communications varies between alternative microservice-controller placements, the number of routing alternatives also varies for each placement, and thus, we cannot give a general formula for the number of alternatives. Nonetheless, in the worst-case scenario, given the set of links of the infrastructure L, the number of alternatives would be $(\sum_{wf\in WF}|WF|)^{|H|}(\sum_{i=1}^{C_{max}}\binom{|S|}{i})[(\sum_{wf\in WF}|wf|)+|S|+1]|L|$

input/output information size, CPU cycles). The infrastructure description is, in essence, a graph that represents the expected topology, including the network and computing nodes. Each of the vertices is annotated with extra information about their characteristics for the infrastructure, such as whether a vertex is a computing node or a switch, and the characteristics of the computing node (e.g., RAM, CPU, clock speed). Edges are also annotated with extra information: the capacity of the link, and the latency derived from using it. These descriptions are very loosely coupled with each other, and therefore, it is simple to reuse a given application with multiple infrastructures and vice-versa [5]. Furthermore, it is possible to describe the QoS metrics to be optimized by the framework (see Section 2.2). By providing these inputs, DADO is able to optimize the microservice placement, SDN controller placement, and network routing, as well as give additional execution results (e.g., the optimization time taken by each part of the framework). This information can optionally be fed to post-analysis tools, such as DADO's QoS reporting tools or Pascal, to obtain an estimation of the QoS of the system (e.g., the response time of each workflow, the cost of the deployment, or the link load). Finally, the relevant information is given to the system operator, who is in charge of performing the initial deployment. If the deployment was to be adapted rather than calculated from scratch (see Section 1.5.1), the system operator can be substituted for automatic deployment orchestrators such as Kubernetes [64]. A summary of this description of DADO is shown in Figure 2.5. Finally, in our evaluations in IIoT [2] and IoMT [5] scenarios, DADO has provided better results than other state-of-the-art application placement optimization frameworks, such as ModuleMapping [1], as well as geometry-based, separate approaches to application and controller placement. An example of this difference, extracted from the IIoT-based evaluation, can be seen in Figure 2.6

For further information on DADO's MILP formulation, we refer the interested reader to Chapter 4. For further details of DADO's implementation, Appendix T includes documentation on the matter.

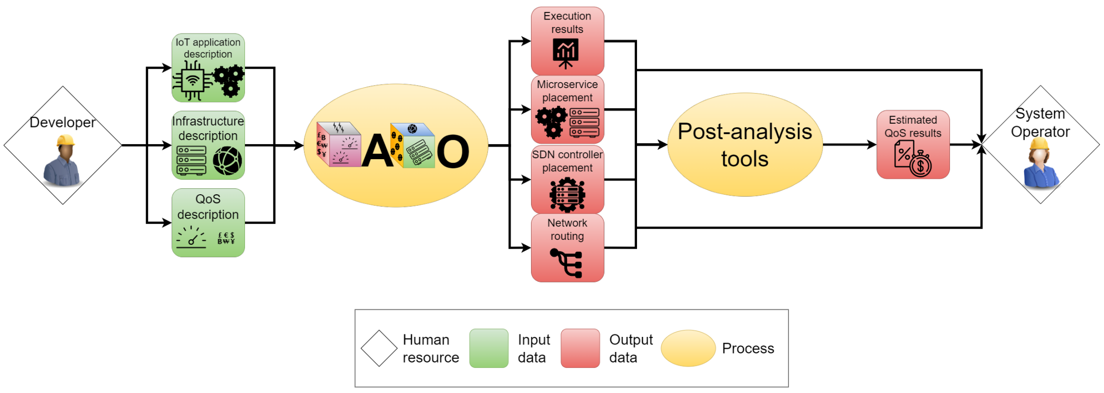


Figure 2.5: Summary of the expected usage of the DADO framework.

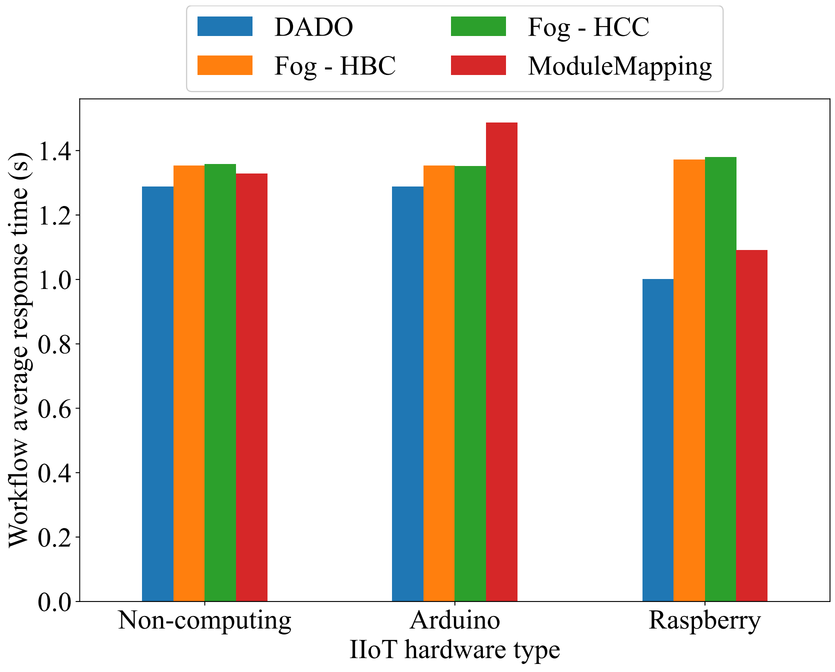


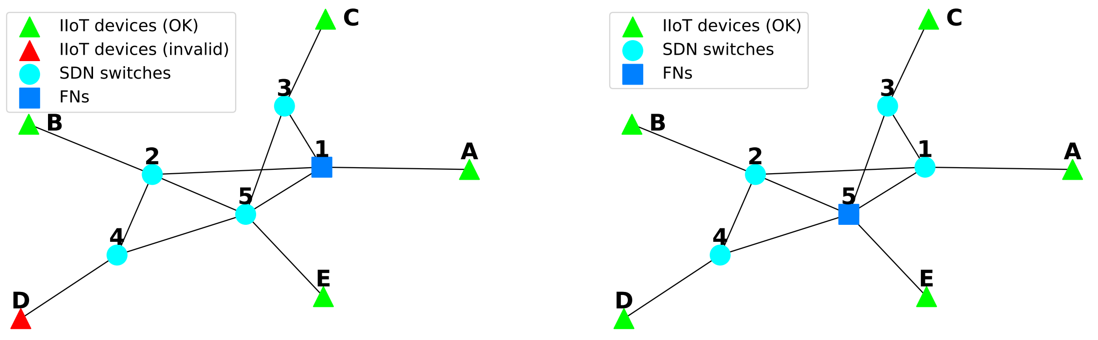
Figure 2.6: Comparison of DADO with alternative approaches to deployment optimization, such as ModuleMapping [1], or graph geometry-based placements (Highest Closeness Centrality/HCC, Highest Betweenness Centrality/HBC). Figure source: [2].

2.1.2 The FNPP: Networking and computing optimization

As the problem of solving both the DCDP and the CPP stems from the fact that operators are in control of both the application deployment and the network, there is a possibility that the operators are also in control of the computing dimension. The Cloud-to-Thing continuum contains computing devices that are completely managed by an external provider, including not only cloud servers, but also fog and edge nodes managed by providers in a similar manner, such as Edge instances [65]. Nonetheless, especially in intensive scenarios, it is not uncommon that the operators also include self-hosted devices, such as in-premises edge and fog nodes [13]. Thus, while their control over the devices managed by providers is limited, they fully control their self-hosted devices. Considering that these environments tend to use the SDN paradigm [28], it is desirable to use a model for these nodes as the one defined by Bedhief *et al.* [35]. This model consists of a Fog Node (FN)[4] that contains not only a computing device but also an SDN switch, easing its integration into existing SDN networks. However, the placement of fog nodes in the infrastructure is known to affect the QoS of the systems and applications that make use of them [35]. It is therefore necessary to optimally place these nodes to obtain the desired QoS. This is a problem we initially defined and solved: the FNPP [3, 36, 37, 44].

To exemplify the interest and cases that can appear in the FNPP, we use an IIoT-based example extracted from [3]. In this example, a factory automation application is going to be deployed on an SDN network topology, which consists of five IIoT devices and five SDN switches. The factory automation application that will be deployed has very strict latency requirements [9], and thus, the factory owner has decided to transform the SDN topology into an SDN-fog infrastructure. To enable this transformation, the factory owner makes use of FNs that follow the model from [35]: hardware boxes that include an SDN switch and a computing device, that will substitute existing SDN switches. To facilitate the understanding of the example, we assume that all links have the same latency, and thus, latencies can

[4]Although it is called ”Fog Node”, FNs can also be used in other infrastructure layers, e.g., edge.



(a) FN placement in 1 (b) FN placement in 5

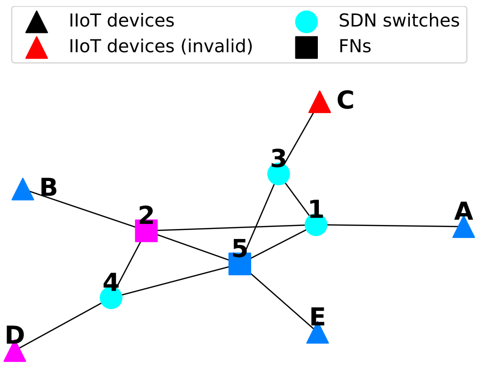
Figure 2.7: FNPP first scenario: placement of one FN. Figure source: [3]

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). In the first scenario of this example, the factory owner will replace a single SDN switch with an FN. The topology has five SDN switches, hence, there are five possible placements for the FN, as depicted by Figure 2.7. However, not all placements are equally valid. Take, for example, the placement from Fig 2.7a. Assuming the use of 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 Figure 2.7b. 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 simple.

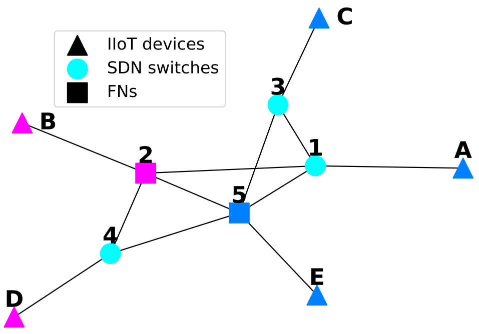
In the second scenario, shown in Figure 2.8, rather than a single FN, two SDN switches are to be replaced with FNs. However, these FNs are less powerful than the

FN from the previous scenario and, therefore, each FN can only process the workload of up to three IIoT devices. Thus, the solution from the previous scenario is not valid anymore: 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 optimized: which IIoT devices should be served by each FN, meeting the capacity constraints of the FNs. This is not a simple decision to take, as 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, as represented in Figure 2.8a. 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 Figure 2.8b. Hence, placing multiple adds to the complexity of the problem, not exclusively because of the higher number of combinations, but also because there are more decisions to take. 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 could be infeasible.

This problem, including the placement of FNs by replacing SDN switches, the assignment between IoT devices and FNs considering capacity constraints, and the routing between each IoT device and its assigned FN, considering the possible congestion, and the optimization of all these decisions as a means to optimize QoS, is what we originally defined as the FNPP [36]. Furthermore, as the FNPP is very complicated to solve manually, especially in larger network topologies, we developed automated systems to solve the FNPP [3, 37]. On the one hand, we developed two



(a) First assignment option.

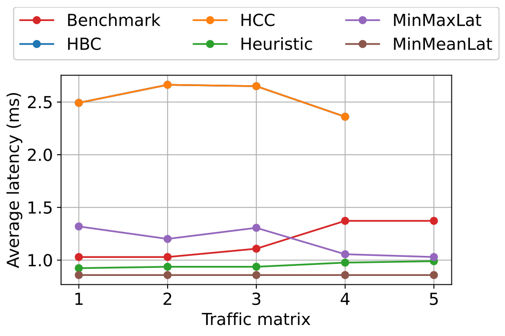


(b) Second assignment option.

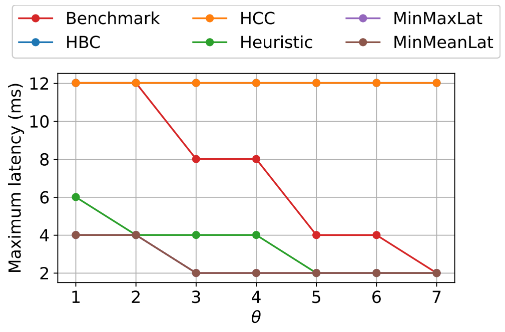
Figure 2.8: FNPP second scenario: placement of two FNs. Figure source: [4]

optimal methods based on MILP solving: *MinMeanLat* [37], which minimizes the mean latency; and *MinMaxLat* [3]. which minimizes the maximum latency of each IoT device with its assigned FN (i.e., instead of ensuring the mean latency is minimal, even if it implies that some IoT devices get very small latencies and other devices get higher latencies, MinMaxLat tries to keep all the devices with a similar latency). The third method is a heuristic that uses a machine learning technique named k-medoids, modified to use latency as the main metric. Figure 2.9 includes a comparison of these solutions with naïve HBC and HCC placements, as well as a benchmark from a proposal similar to the FNPP made by Maiti *et al.* [66]. While both HBC and HCC provide similar solutions, referring to the average latencies (Figure 2.9a), the heuristic and, in more loaded traffic matrices, MinMaxLat, provide better solutions than the benchmark. Furthermore, as it is the solution of the mathematical formulation of the problem, MinMeanLat provides the best result. Referring to Figure 2.9b, both MinMeanLat and MinMaxLat provide optimal results, followed by the heuristic, all of which provide better solutions than the naïve approaches and the literature benchmark.

For further information on the FNPP's MILP formulations, both for MinMeanLat and MinMaxLat, as well as the design of the heuristic, we refer the interested reader to Chapter 6. For further details of the implementation of MinMeanLat, MinMaxLat, or the heuristic, Appendix U includes documentation on the matter.



(a) Comparison of average latencies, 4 FNs placed.



(b) Comparison of maximum latencies overall, θ refers to the number of FNs placed.

Figure 2.9: Comparison of methods for FNPP solving. Figure source: [3]

2.1.3 Umizatou: optimal placement of computing devices, microservices, and SDN controllers

A theme that repeats itself throughout this section is the fact that the possibility to perform the optimizations (and thus, the optimization problems themselves) stem from the control of the developer or the operator over different decisions. So far, we have considered the union of control over the application and networking dimensions with DADO, and the control over the computing dimension with FNPP solvers. Nonetheless, it is possible to have control over all three dimensions. In such a case, the operators would have to solve all three, the DCDP, the FNPP, and the CPP. Moreover, we have established that all three dimensions are inherently related, and these relationships play an essential role in such optimization. Figure 2.10 provides a small summary of the relationships between dimensions and problems. The DCDP and the FNPP are related, as the placement of microservices depends on how often the microservices are requested together in workflows, the complexity of each microservice and the power of candidate machines to run them, and on the size and type of their I/O information. From the perspective of the FNPP, where to place each FN depends on which microservices are to be deployed in it. Similarly, the FNPP and the CPP are related, as the location of the SDN controller depends on the communication between FNs and the traffic to be sent. Moreover, the location of FNs themselves

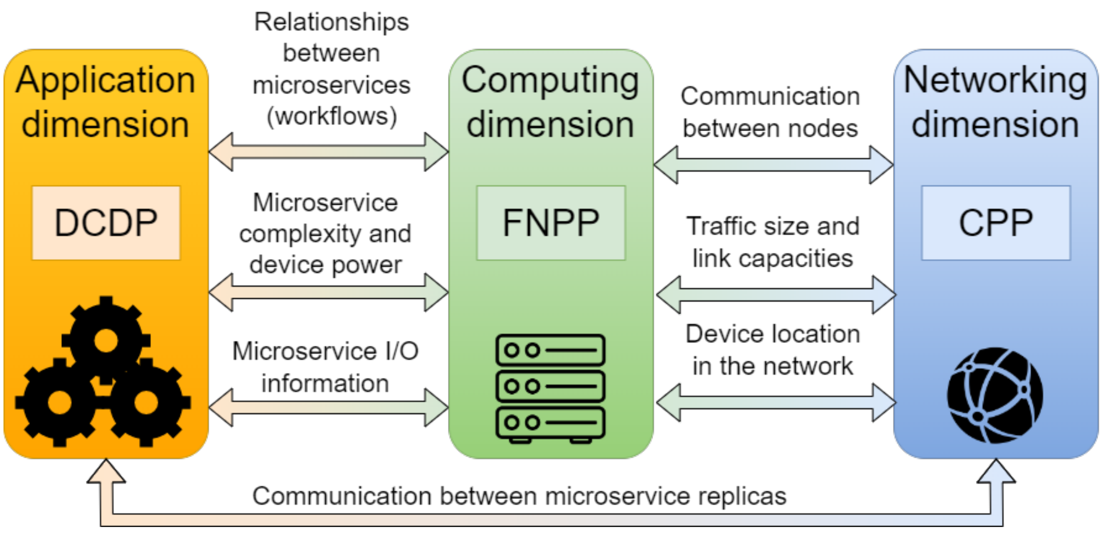

Application dimension
DCDP
Relationships between microservices (workflows)
Microservice complexity and device power
Microservice I/O information
Computing dimension
FNPP
Communication between nodes
Traffic size and link capacities
Device location in the network
Networking dimension
CPP
Communication between microservice replicas

Figure 2.10: Summary of the problems on each dimension and their relationships.

depends on the network's QoS and routing capabilities. Finally, the patterns of how traffic flows in the network depend on the DCDP and greatly affect the CPP. It is thus desirable that the DCDP, the FNPP, and the CPP are solved in a joint effort, i.e., to merge DADO with the FNPP solvers.

To respond to this need, we created Umizatou [42]: a tool to optimally place FNs, microservices, and SDN controllers, as well as optimally routing the traffic containing both SDN control messages and microservice data and requests. Umizatou merges the model of DADO with the model of the FNPP, making some information more concrete and diffusing the *a priori* lines between networking and computing equipment. On the one hand, the abstract *capacity* of FNs becomes concrete, relating to the computing capacity of each FN w.r.t. the microservices. On the other hand, the input topology consists exclusively of IoT devices and SDN switches, as the FNs will replace some SDN switches due to the FN model [35]. To the best of our knowledge, Umizatou is the first system to perform this all-in-one optimization, and thus, we could not compare it with state-of-the-art alternatives. Furthermore, the multi-objective version of Umizatou was the first artifact to be evaluated under our emulated testbed generator, Pascal.

For more information on the model of Umizatou and the formulation of its implementation problem, we refer the interested reader to Chapter 7. For more details on the implementation of Umizatou, Appendix Z includes documentation on the matter.

2.1.4 Which system should a developer use?

After presenting the three main families of artifacts for holistic optimization: DADO, the FNPP solvers, and Umizatou, there is a crucial question that may be complicated to answer. Which family of systems is a developer or operator expected to use? As it has been stated, each of the three solve different problems, and thus, depending on the characteristics of the concrete situation, a different family of systems may be more or less suitable. The main question that must be answered is *How many dimensions does the developer or operator control?*, considering the dimensions and problems presented in Figure 2.10. Thus, *controlling the application dimension* refers to *being in control of microservice replication and placement*. Similarly, control over the computing dimension refers to FN placement; and control over the network refers to SDN controller placement and traffic routing. Moreover, we assume that the developer or operator has control over, at least, one of these dimensions.

If the developer controls a single dimension, it may be any of the three. If they only control the application or networking dimensions, like in the traditional cloud computing case, we refer the developer to more traditional, non-holistic systems for DCDP solving [39], or CPP solving [27], respectively. Nonetheless, if the developer only holds control of the computing dimension, the FNPP may prove useful to place the resulting FNs and assign them to IoT devices. However, the FNPP also optimizes network routing, and thus, there is a special consideration that the developer must keep in mind for the input of the solver: the network must be modeled as if it were fully connected.

If the developer or operator holds control over two dimensions, three possibilities exist for which dimensions are controller: application and computing (e.g., if the network belongs to an external provider), computing and networking (e.g., they operate

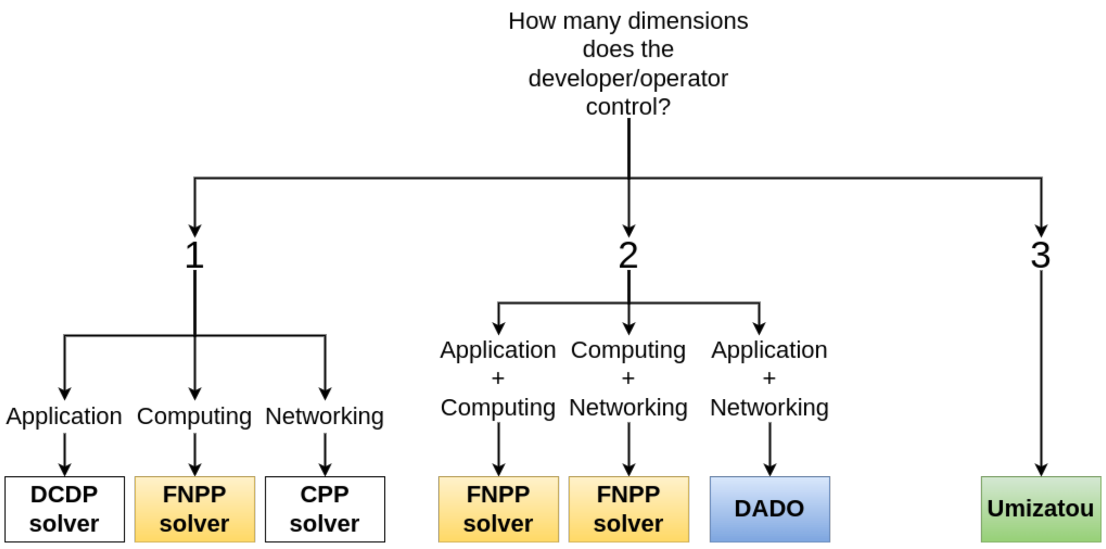


Figure 2.11: Illustrative chart on which family of systems to use.

a Cloud-to-Thing infrastructure for developers), or application and networking (e.g., externally provided Cloud-to-Thing continuum devices). In the first case, they should use the FNPP solvers with the estimates of their application and computing devices, which will allow the FNPP to be modeled accurately. In the second case, they should use the FNPP as-is, because it is a system developed specifically for this situation. Finally, in the third case, DADO will optimize the SDN network controller placement, routing, and microservice replication and placement.

Only if the developer controls all three dimensions, Umizatou could be used as the best alternative for a system. While it is an all-in-one holistic system, as the problem stems from control over all three dimensions, such control is a requirement for its use. Hence, none of the three families of artifacts is inherently better than any other, even if they can be seen as extensions, as they should be applied to different scenarios. Figure 2.11 presents a short summary chart to guide developers in selecting one of these systems.

2.2 Optimization of multiple QoS metrics

In the context of this thesis, the joint consideration of multiple elements for optimization is a staple: on the one hand, we consider multiple dimensions holistically to optimize deployments, and on the other hand, we optimize multiple QoS metrics jointly. In optimization, if a problem requires the optimization of two or more metrics, especially if they must be traded off, it is known as a *multi-objective optimization problem* [5, 27, 39]. If an optimization problem considers multiple metrics but does not *optimize* them (e.g., instead of optimizing the cost, it considers that the solution should not be more expensive than a given budget), it is called *multi-criteria* [39]. It is, therefore, important to remark that the problems and artifacts from this section are truly multi-objective, rather than simply multi-criteria.

The artifacts presented in this section are the evolution of DADO and Umizatou: NIoTO [5] and MO-SFO, respectively, contribute to their previous artifacts by adding the deployment cost as a second objective to their models. This does not only mean that they share most of their details, such as the optimization decisions they take, but also that their holistic nature is preserved. Hence, if the original DADO was a holistic optimization framework, NIoTO is a holistic, multi-objective optimization framework. Furthermore, the same guidelines used for deciding between Umizatou and DADO can be used to choose between MO-SFO and NIoTO.

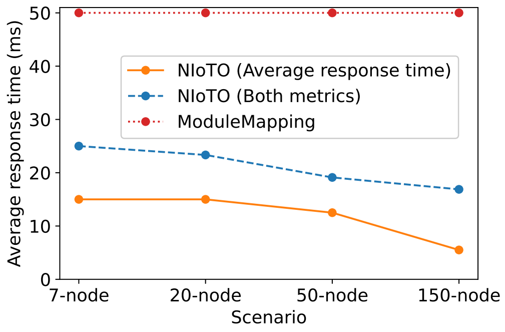
With regard to the response time objective, it is measured using the same model as with DADO and Umizatou: for each workflow request, the system decides where each of the requested microservices is to be deployed. If two or more workflows that request the same microservice stereotype have that specific microservice deployed to the same machine, it is assumed to be the same replica. If they were to be deployed in different machines, they are considered to be multiple replicas. While the system decides the microservice placement and replication, it also optimizes how traffic is routed from each microservice to the next, as well as back to the device. For each SDN switch that a traffic flow traverses, it is assumed that service discovery must be performed, and thus, a flow setup will be required. The placement of SDN controllers, as well as controller-SDN switch assignment and control traffic routing, are also computed by the optimization system. Finally, the response time of the workflow is computed as the sum of the execution time of the requested microservices (i.e., the number of CPU cycles per execution divided by CPU clock speed), the latency of the routed traffic flows (i.e., the sum of the latencies of the links traversed by each traffic flow that carries workflow data), and the latency of the associated flow setups (i.e., the sum of the latencies of the links from each SDN switch traversed to its assigned SDN controller). Hence, response time integrates the three dimensions: application (CPU cycles), computing (CPU speed), and networking (latency), highlighting the interest in the holistic approach in multi-objective optimization.

For the cost, both NIoTO and MO-SFO have also the same model. First, they separate costs into two categories: Capital Expenditures (CAPEX) and Operational Expenditures (OPEX). CAPEX refers to the cost of buying the new fixed assets (e.g., buying FNs, servers, SDN switches, IoT devices) [67]. OPEX, on the other hand, is the constant cost of having said assets operational (e.g., cost per use of cloud servers, cost of energy consumption of IoT devices) [67]. This separation allows the model to support scenarios with self-hosted devices (mainly FNs), externally provided servers (e.g., cloud servers, Edge instances [65]), as well as hybrid scenarios. To make calculations over both costs, the system analyzes which devices are considered to be *used* in the scenario: if a device is not used, it can safely be removed from the scenario, which means that both its CAPEX and OPEX would become 0 [5]. In the case of computing devices, it is considered that one is used if either they request at least one workflow, or if they host at least one microservice replica, as in both cases, they play a role in the scenario and cannot be removed. For SDN switches, if at least one flow of traffic arrives at them, either because it is routed through the switch or because the switch contains an SDN controller, they are considered as used. Used devices are considered to have their CAPEX as cost, which is an input expected from the user. Moreover, SDN switches with co-located controllers are considered to have two summed CAPEX: the switch's CAPEX, and the controller's CAPEX.

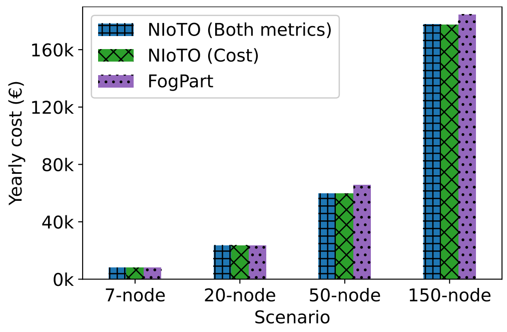
On the other hand, OPEX models differ: for networking devices, they are traditionally considered to have an OPEX per unit of time, due to their energy consumption, whenever they are powered on, regardless of their load [5]. This is also true for SDN controllers, which have an additional OPEX, in the same manner as CAPEX. Computing devices are more complex: their OPEX depends on how they are used, especially for pay-per-use externally provided instances, which are billed by second of use, a model that is the norm in the cloud computing space [20]. In our model, the OPEX per second is transformed to OPEX per CPU cycle, a straightforward conversion by dividing the cost per second by the CPU speed. Hence, it is considered that the OPEX of the device is the OPEX per execution cycle of the device, multiplied by the sum of the CPU cycles of all the deployed microservice replicas. This is the model used for cost, which adds an additional feature to both NIoTO and MO-SFO: rather than manually specifying the number of SDN controllers or FNs to place, they automatically adjust them to optimize cost and response time.

On the multi-objective optimization model of the two approaches, both NIoTO and MO-SFO allow the user to select the objective or objectives they desire. This is, a developer or operator can choose to optimize cost only, response time only, both, or none. Hence, various use cases can be covered with them: finding the least costly feasible setups, regardless of their performance; the fastest setups, regardless of the cost; balanced setups that trade cost and response time off; or just a feasible deployment, regardless of both cost and response time. Moreover, to ease the task of the developer, the model only requires input information that is related to at least one of the objectives. For example, if response time is the only objective, the operator is not required to provide information on the CAPEX or OPEX of any of the elements.

Finally, we remark on the novelty of both approaches. For the same reasons as Umizatou, MO-SFO, to the best of our knowledge, no other system aimed at solving the all-in-one problem exists. For NIoTO, we have been unable to find a multi-objective system to compare it to, but it can be compared with two single-objective systems: ModuleMapping [1], which focuses on performance, and



(a) Comparison of average response times, NIoTO vs ModuleMapping.



(b) Comparison of deployment costs, NIoTO vs FogPart.

Figure 2.12: Comparison of NIoTO with single-objective benchmarks. Figure source: [5]

FogPart [68], which focuses on cost. Figure 2.12a provides a comparison with ModuleMapping, while Figure 2.12b compares with FogPart. As a summary of the response time comparison, the gap between ModuleMapping's response times and NIoTO's is 40.22 ms, and NIoTO achieves an average speed-up of 5.11 w.r.t. ModuleMapping. Following up with costs, the cost gap between FogPart and NIoTO increases with topology size, being on average of 3954€ (6.22%) in the tests performed in [5]. Furthermore, in both cases, the multi-objective approach also obtains better results than the state-of-the-art benchmarks.

For further details on the formulation of NIoTO and its implementation, we refer the interested reader to Chapter 5. As MO-SFO is still under review, a pre-print version of the article is included as Appendix F. We also provide technical documentation on NIoTO in Appendix T, and on MO-SFO in Appendix Z.

2.3 Adaptation of QoS-related decisions at runtime

The contributions presented so far are meant for optimization at the late design phase of an application's lifecycle: developers and operators are expected to have an estimation of their application's characteristics (e.g., microservice complexity, devices planned to be used for deployment, planned network topology, the volume of workflow requests), which will be used by the selected framework to find an optimal solution [5]. Nonetheless, it is not uncommon for changes to occur in the environment: new versions of the application may change the complexity and resource usage of microservices, as well as create new microservices or modify the workflows for some functionalities, additional network traffic (e.g., due to an unrelated application being used in the same infrastructure, or operating system telemetry) may change the capacity or technical characteristics of links, any of the hardware devices or equipment may fail, new hardware elements may be added to the infrastructure or even a different volume of requests.

In these situations, the presented frameworks could be used to re-optimize the situation by redefining the input scenario to reflect the new situation and executing them again. However, most of the artifacts make use of MILP solvers, namely, MinMeanLat [3], MinMaxLat [3], DADO [2], NIoTO [5], Umizatou [42], and MO-SFO. While one of the traits of these solvers is the fact that they provide optimal solutions [2], their main caveat is their high optimization time for large problems, such as big infrastructures, applications with many microservices, or a large volume of requests. For example, DADO can take nearly 6 hours to optimize a large topology [2], and it can take more than three times as much if multi-objective optimization is selected [5]. Hence, there is a need for adaptive systems for our problem. It is important to note, in this regard, that only two of the decisions could be adapted at runtime: microservice placement-replication, and traffic flow routing, this is, DCDP and routing optimization. Moving SDN controllers or FNs requires manual, human intervention, which is not only unable to be automated but also a larger bottleneck in terms of adaptation times than the framework's optimization times. Hence, in this section, we present our contributions to runtime optimization adaptation: monitoring-based, stochastic, and opportunistic, as well as a final summary to aid developers in choosing one of the systems.

2.3.1 Continuously adapting the application placement: ConDADO

If we analyze why DADO can take a long time to optimize a situation, we can arrive at the conclusion that there are two main reasons: the size of the problem and the solving method. The size of the problem refers to the number of computing devices, SDN switches, microservices, workflows, and links, that the situation has. DADO can be fast in small problems, optimizing in a few seconds [2], but does not scale well when the problem is larger. Nonetheless, if the size of the problem in large scenarios could be reduced (e.g., by having DADO consider only a part of the elements), its speed would be improved. On the other hand, DADO completely relies on the MILP-solving algorithms to make all its decisions, which are slow due to their multi-purpose and optimal nature. If a more efficient algorithm could be used to determine microservice placements, DADO could also be quicker to solve each problem.

Moreover, speed is not the only important factor. The adaptation process implies that changes are performed in the microservice placements. Making each of these changes may have a cost that should be considered by the system. In this case, we do not refer to monetary cost, but to a more general concept of *cost*. This cost is generally incurred due to *migrations*: movement of microservices from one machine to another, or creation of new microservice replicas. Migrating a microservice from one device to another may require the new device to provision this new microservice (e.g., if a virtualization platform such as Docker is used, the device must download the Docker image of the microservice). This download takes time, and, during this time, the microservice must stay in the original device, which is not the desirable placement. Moreover, it is likely that the organization may not want to migrate some microservices, e.g., databases that contain especially sensitive data should not be migrated freely to avoid security breaches. This cost due to migrations, which is not monetary but technical or organizational, is what we label as *migration cost*, and is interesting to consider in adaptive systems.

Continuous DADO or ConDADO is a solution for this adaptation problem that builds on DADO. Regarding the techniques to improve the optimization speeds, ConDADO does both: by exploiting state-of-the-art MILP solving techniques in combination with continuous reasoning techniques, ConDADO can quickly respond to environmental changes. On the other hand, these same techniques allow ConDADO to be aware of the past status of the infrastructure and mitigate migration costs.

To understand ConDADO, it is crucial to first understand its main technique: continuous reasoning. This approach is used to reduce compilation times for changes to large software repositories such as Facebook, with a tool known as *Facebook Infer* [60]. Each change to the repository, rather than triggering a complete compilation, triggered a static analysis of the codebase that detected the parts that would be affected by the change [60]. Leveraging continuous reasoning, the system builds and validates only the parts of the code that are affected by the changes, rather than the full codebase, which can speed up the compilation process. In the context of application placement, continuous reasoning is leveraged by analyzing which microservices of the application placement need to be migrated (i.e., moved between nodes) after an environmental change [25]. In a similar manner to Facebook Infer, the main idea behind continuous reasoning in application placement is to find which parts of the system have been affected by a change and must be re-computed.

Continuous reasoning reduces the size of the application placement problem, as only those elements that were meaningfully affected by environmental changes need to be migrated, which may speed up decision-making [2], and allows for the consideration of migration costs. In the case of ConDADO, the monitoring system provides updates on the status of the scenario. Each update is then sent to the *delta calculator*, whose objective is to determine the differences between the previous and the current status of the scenario. If differences exist, they are fed, along with the previous decisions taken on deployment, to the *Continuous Reasoning Engine (CRE)*. The CRE determines whether the changes affect the validity or QoS existing deployment and if they do have such effects, computes which microservices need to have their placement or traffic routing re-optimized. The CRE selects these decisions

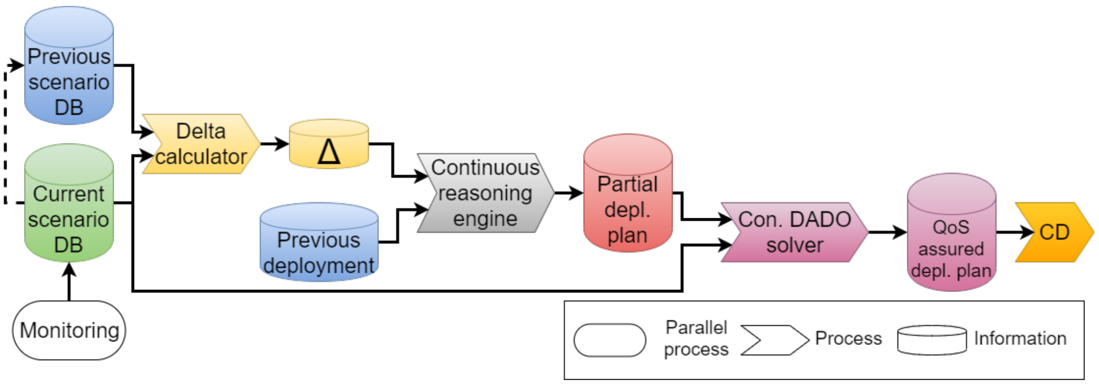


Figure 2.13: ConDADO architecture. Figure source: [**Computing 2022**]

through their migration cost as an effort to mitigate this effect. The CRE then generates a *partial deployment plan*, i.e., a list of the deployment decisions that are not required to be changed. This is fed to the ConDADO solver, which only re-optimizes the microservices and traffic routing decisions that are not in the partial deployment plan. This structure is shown in Figure 2.13.

Moreover, ConDADO is designed to be integrated into DevOps. Through a proposal that is presented together with ConDADO, named Continuous Adaptation (CA), we propose the integration of application placement adaptation with DevOps. CA proposes to add an intermediate step in Continuous Integration and Continuous Deployment (CI/CD) pipelines, able to be triggered both by application changes and infrastructural changes through monitoring. Using continuous reasoning, CA performs an adaptation of the application placement, if so required, and feeds the new placement to CD, for it to redeploy the application as necessary. A depiction of CA's role in CI/CA/CD pipelines is shown in Figure 2.14 It is important to note that, while ConDADO is a solution designed as a prototype of a CA-enabling system, it is possible for other application placement adaptation systems, such as FogBrainX [69], to be used for similar purposes.

Finally, as a demonstration of the differences between ConDADO and DADO in terms of optimization time, we present a small excerpt of ConDADO's evaluation in Figure 2.15. This figure represents the evolution of the response time of an application that cannot surpass 50 ms of response time. Initially, the application is deployed below

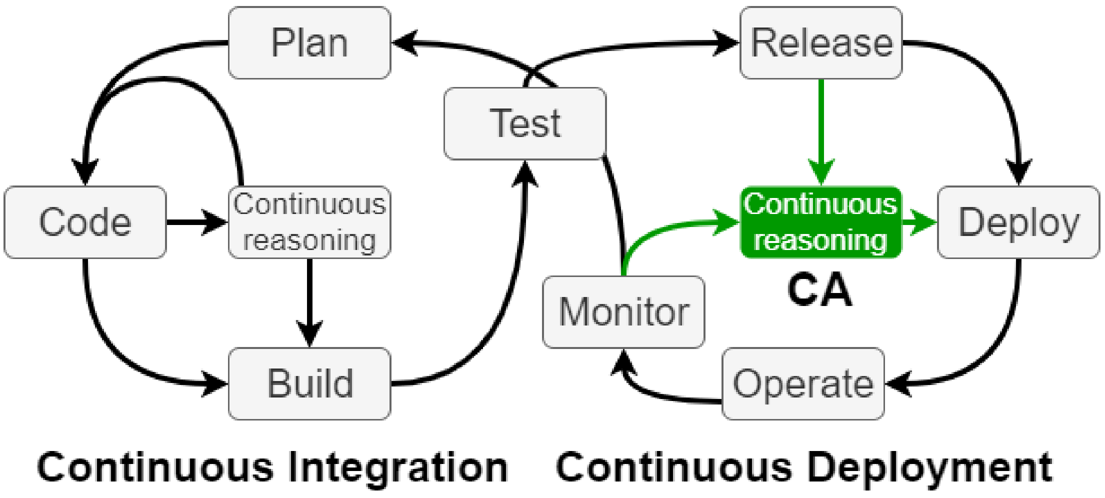


Figure 2.14: CI/CA/CD pipeline. Figure source: [**Computing 2022**]

the threshold, and thus, neither system requires optimization. Nonetheless, once t is reached, a change in the cloud provider provokes a raise in the response time above 50 ms. Once the curve reaches its peak, near $t+2$, the monitoring system feeds both DADO and ConDADO with the new situation of the environment for them to adapt the application placement. ConDADO is able to adapt it in 27.5 seconds, migrating the microservices to the fog and decreasing the response time greatly. However, DADO requires 302 seconds (approx. 5 minutes) to compute the migration. Hence, for this situation, ConDADO exhibits a speed-up of approximately $11\times$ w.r.t. DADO in this situation.

For further details on the proposal of CA, as well as on ConDADO's functioning, we refer the interested reader to the pre-print version in Appendix B. We also provide documentation on ConDADO's implementation in Appendix V.

2.3.2 Adapting to probabilistic stochastic models: S-DADO

While continuous reasoning is an interesting technique for the adaptation of deployments, there is another technique that can also be used to create adaptative systems: Lyapunov optimization [46]. This is an optimization framework meant to optimally control a dynamic system, which tries to find the *equilibrium* through its decisions. This is, a Lyapunov optimization model acknowledges that some

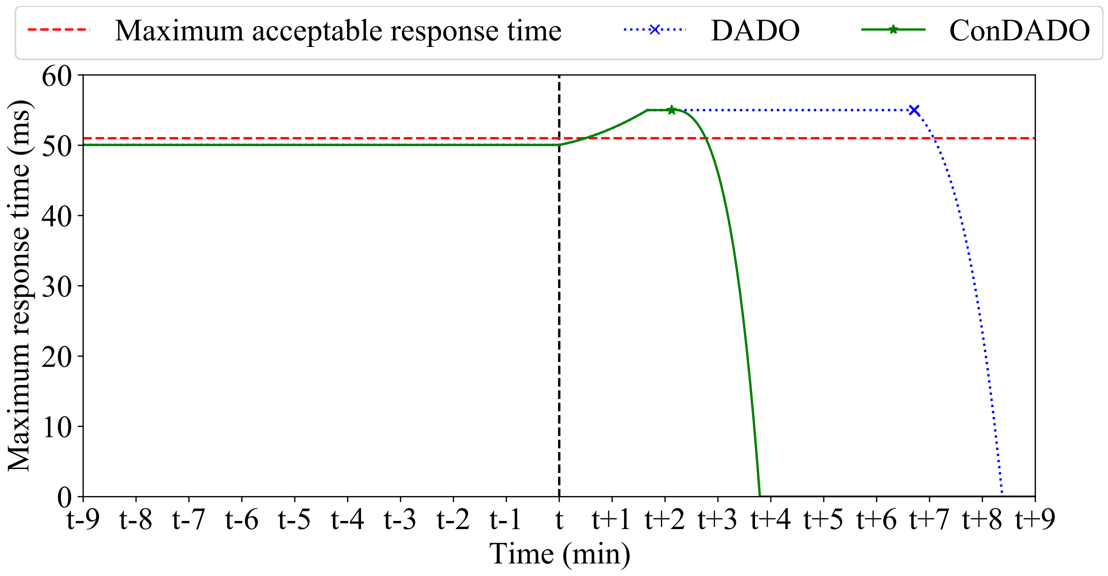


Figure 2.15: Evolution of an application's response time with DADO and ConDADO. Figure source: [**Computing 2022**]

parts of the system are dynamic, and adapts its decisions until the system reaches stability [46]. In this regard, the approach greatly differs from continuous reasoning, as it is proactive rather than reactive. Continuous reasoning reacts to changes by adapting the deployment, while Lyapunov optimization assumes changes will exist and provides a solution that will remain stable through these changes. This is especially useful for systems that do not follow a DevOps methodology, or where it is not possible to perform continuous monitoring.

In this context, we developed a Lyapunov-based version of DADO, named Stochastic DADO (S-DADO). Nonetheless, making use of Lyapunov optimization implies performing optimization in an arbitrary number of time slots, each of the slots requiring the solution of a DADO model [46]. Unlike continuous reasoning, which provides tools to speed up problem-solving, Lyapunov optimization requires fast problem-solving as the starting point, as it requires multiple problem solutions, fundamentally slowing down the optimization process. Thus, to reduce the size of the problem, the traffic routing between devices in S-DADO is performed heuristically, using shortest path-based routing. Hence, only microservice placement and replication are optimized by S-DADO. This optimization considers the same characteristics of all three dimensions as the original DADO: RAM resources, network latency, CPU cycles, etc. Nonetheless, these are not updated through monitoring, and instead, their values are drawn from probabilistic models for each time slot. Concretely, S-DADO considers this dynamic model for computing devices (RAM, CPU clock speed), microservices (RAM consumption, CPU cycles, generated traffic), and network links (capacity, latency).

To guide the optimization, the Lyapunov-based model uses *virtual queues*, which represent the stability of the system. Each of the workflow requests in S-DADO has a deadline, which is the maximum acceptable response time of the workflow. As these deadlines represent the stability of a workflow, i.e., a workflow is stable if the response time is less or equal to the deadline, each workflow also has an associated virtual queue. If the response time of a workflow is higher than its deadline (i.e., the deadline is violated), the excess response time is added to this virtual queue. The average size of the virtual queues is known as *Lyapunov drift*, as it represents the drift from the stability of the system [46]. S-DADO uses an objective function known as *Drift-Plus-Penalty* [46], which tries, at the same time, to minimize the average response time of the workflows, and to minimize the Lyapunov drift. A consequence of the use of Lyapunov optimization is that, if the Lyapunov drift is bounded, the system will be, on average, stable in an infinite time horizon [46]. This is, if there is a feasible solution that will stay stable over time, S-DADO will find it.

S-DADO is currently a work in progress, and thus, we are still working on its evaluation and the diffusion of its results. Nonetheless, the interested reader may find details on S-DADO's implementation in Appendix Y.

2.3.3 Opportunistic microservice migration: μDADO

During the development of this thesis, there has been ongoing work in privacy and microservice-based applications for mist computing devices, and more concretely, for companion devices, such as mobile phones or wearables [48, 53]. This line led us to the conclusion that it would also be desirable to optimally place and replicate microservices in purely mist scenarios. Nonetheless, there are some differences from what DADO considers. First, in DADO, the same party holds control over the optimized dimensions, but in the mist space, each device is generally in control of a different party [53]. Moreover, these devices may not have an Internet connection, and use ad-hoc networks or other wireless technologies for creating a network exclusively for these devices, in what is known as *opportunistic networks* [48, 53]. Finally, the crucial QoS metrics to optimize may differ greatly, as battery consumption, which is not critical for the edge, fog, or cloud nodes that are permanently connected to a power outlet, becomes a crucial metric.

To bring the utilities of microservice placement optimization to the mist-opportunistic space, we built a platform named Pervasive Opportunistic Delegation of Services (PODS). PODS is designed as a platform for communities of users that are physically close to run collaborative services, accessible by all the users in the community, with PODS automatically managing the service virtualization, placement, replication, and migration. PODS allows monitoring of the resources, as well as other contextual attributes, of nearby computing nodes, including those devices participating in a collaborative community. This monitoring is used to analyze which device(s) in the near surroundings should host each collaborative service based on their context, optimizing application-specific Key Performance Indicators (KPIs) to obtain a good user experience. Furthermore, PODS monitors the relevant contextual conditions for changes, e.g., a decrease of node resources running such services or movement of nodes, designating and migrating services to other nodes in response to these contextual changes. This allows for running collaborative applications in a more reliable way and where all users make their resources available to maintain the session as long as necessary.

The architecture of PODS, depicted in Figure 2.16, comprises a variety of modules. First, the *context scanner* module, represented in pink, has the task of continuously monitoring the nearby devices, including the device itself, and determining if a delegation (i.e., migration or change in the microservice placement or replication) is required, based on the KPIs provided by both the application developers and users. This information is fed to μDADO, the main element we focus on in this thesis. μDADO is an adaptive version of DADO, created to be able to support multiple KPIs with a small effort from the user and the developer, while also smaller and faster. μDADO is only able to decide on the number of replicas of each microservice and their placement, similarly to S-DADO. Nonetheless, μDADO's model is not based on workflow requests, it rather tries to make sure the service is available for all users. This is performed by knowing each device's *coverage*, i.e., which devices can reach it, as well as their KPIs. Furthermore, μDADO uses the KPIs obtained from the system information directly, such as latencies, battery percentages, or CPU speeds. Moreover, μDADO allows users and developers to declare if the KPI should be minimized, maximized, or if the device(s) chosen to run the microservices should meet a certain condition w.r.t. the KPI (e.g., the battery level of the device must be above 50%, its RAM consumption must be under 2 GB). μDADO then makes a decision, which is sent to the *service migrator*, a module that can migrate services between devices. To do so, it communicates with the *service virtualization platform*, which, in our current implementation of PODS, is Docker. Hence, the service migrator moves the Docker containers to meet the placement and replication that μDADO decided on.

PODS, and in consequence μDADO, is currently a work in progress, and thus, we are still working on its evaluation, as well as in the diffusion of its results. Nonetheless, the interested reader may find more details on μDADO, its implementation, and other technical aspects in Appendix AA.

2.3.4 Which adaptative system should a developer use?

Similarly to the holistic systems presented in the prior section, the developer or operator may wonder which system is a better fit for their problem, depending on their environment. For the same reason, we have created a flowchart to aid developers and operators to choose which system fits their needs best, displayed in Figure 2.17. The developer or operator should start on the **START** box, on the top left. Then, there

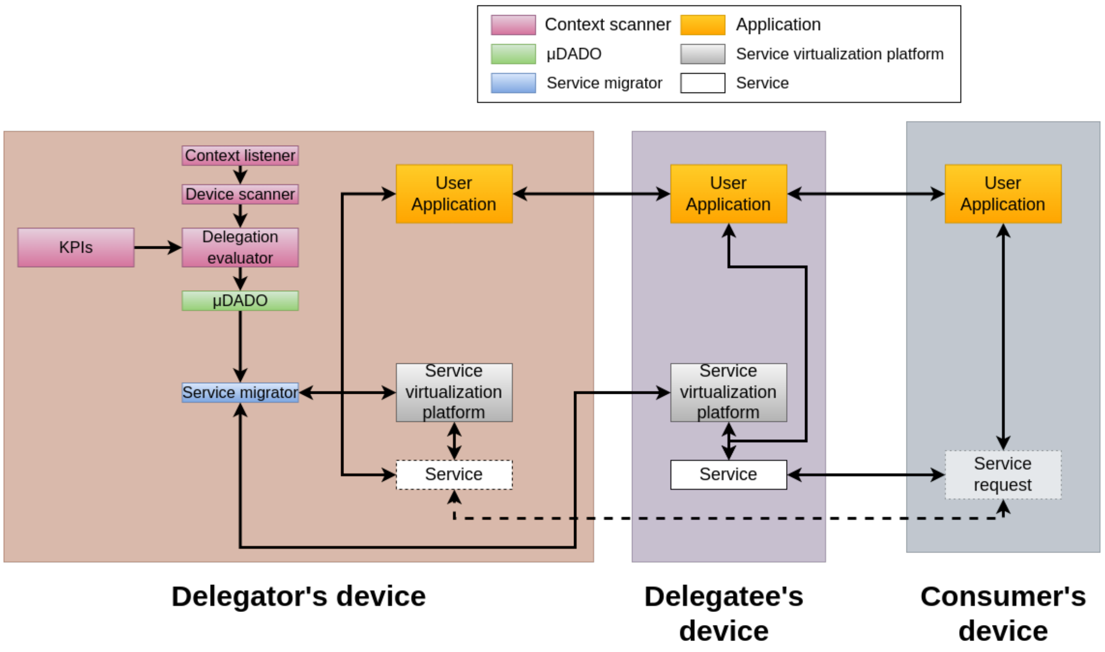


Figure 2.16: PODS architecture.

are two questions to be answered, to determine which of the three systems should be used. First, it is key to determine which type of environment should be optimized. If the environment is exclusively a mist computing environment, mainly in the case of pure mist, companion device-based infrastructures connected with opportunistic networks, the solution should give support to such environments. Hence, PODS (and, by extension, μDADO) is a suitable solution.

On the other hand, it is possible that the environment is more similar to what we defined in the previous sections: a Cloud-to-Thing continuum infrastructure connected by an SDN network. In this case, a second question must be answered: how are changes in the environment detected? It is possible that, due to the characteristics of the environment, changes are not detected, and instead, the developer or operator is looking for a system that will take dynamicity and changes for granted, finding an equilibrium of their system. In such cases, we recommend S-DADO, as its functionalities are designed for these scenarios. In other cases, we may find that the environment is continuously monitored, likely using modern development practices such as DevOps, and CI/CD pipelines. For these scenarios, ConDADO is a better

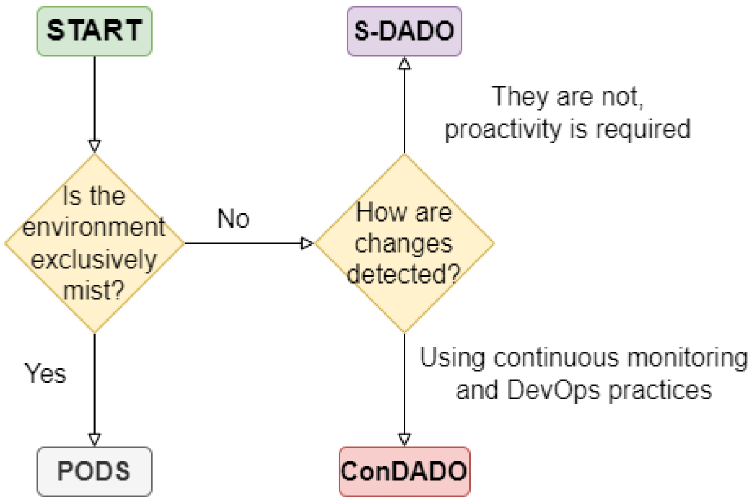

START
S-DADO
They are not, proactivity is required
Is the environment exclusively mist?
No
How are changes detected?
Yes
Using continuous monitoring and DevOps practices
PODS
ConDADO

Figure 2.17: Flowchart to decide on the adaptive system.

solution, prepared to be integrated with the DevOps cycle and be triggered by both CI and monitoring systems, and automatically migrate services through CD.

Everything in this world leads to
soul enlightenment, there is
nothing wasted about our lives.

What do you fight for?, first verse.
Daisuke Ishiwatari

Chapter 3

List of publications

This chapter serves as an overview of the publications produced throughout the PhD. First, we detail the publications that integrate the compendium of this dissertation, which are full papers published in high-level JCR journals and high-ranking international conferences, in Section 3.1. The full-text copies of these publications are provided in Chapters 4 through 7. We then detail the supporting publications of this dissertation, which include publications in international conferences and workshops, national conferences, as well as pre-prints of full papers under review in high-level JCR journals and high-ranking international conferences, in Section 3.2. The supporting publications are provided in Appendices A through S.

3.1 Compendium publications

Through the development of the PhD, many of the results of our research have been communicated in a variety of publications. This dissertation follows a modality known as *thesis by compendium of publications* (*tesis por compendio de publicaciones*), in which the dissertation is integrated by a set of publications, that must belong to relevant venues. Concretely, for the PhD Program in Computer Science of the University of Extremadura, a thesis by compendium must include, at least, three publications. Two of the publications must belong to a journal ranked in the first two quartiles (i.e., Q1, Q2) of the JCR ranking in a category belonging to Computer Science. The third and following publications must be either published in any journal indexed in at least one JCR category belonging to Computer Science, or in an international conference of class 1 or 2 at the GGS conference rating. Tab. 3.1 presents the four publications we consider to integrate the compendium. As it includes three JCR Q1 publications, along with a GGS 2 full paper, it fulfills the requirement for a publication compendium.

Title	**Key findings**	**Publication venue**	**Quality indicator**	**Chapter**
Optimizing the Response Time in SDN-Fog Environments for Time-Strict IoT Applications	DADO framework definition	IEEE Internet of Things Journal	**JCR Q1**	4
Joint Optimization of Response Time and Deployment Cost in Next-Gen IoT Applications	Definition of the NIoTO framework	IEEE Internet of Things Journal	**JCR Q1**	5
QoS-Aware Fog Node Placement for Intensive IoT Applications in SDN-Fog Scenarios	Definition of MILP and heuristic-based FNPP solvers	IEEE Internet of Things Journal	**JCR Q1**	6
Optimal Deployment of Fog Nodes, Microservices and SDN Controllers in Time-Sensitive IoT Scenarios	Fusion of DADO and FNPP solvers into Umizatou	IEEE Global Communications Conference (GLOBECOM)	**GGS Class 2**	7

Table 3.1: Compendium of publications.

3.2 Supporting publications

Aside from the publications that integrate the compendium, the research activities carried out throughout the PhD have generated a number of other publications, which we call *supporting* publications. It is important to note that the term *supporting* does not necessarily imply that these publications are of a lower quality, its meaning refers to the fact that they were not included in the compendium, due to the regulations on dissertations by compendium, because the publication has been extended or superseded by another publication, and to maintain a more theme-centered core of the dissertation. Supporting publications include full papers in international and national conferences, as well as short papers in international workshops, and articles in journals. Furthermore, pre-print versions of full papers under review at international conferences and articles under review after a major or minor revision in a journal are also included.

Table 3.2: Supporting publications of this dissertation.

Title	**Status**	**Venue**	**Rating**	**Appendix**
Context-Aware Privacy-Preserving Access Control for Mobile Computing	Published	Pervasive and Mobile Computing	**JCR Q2**	A
Continuous QoS-Aware Adaptation of Cloud-IoT Application Placements	Under review (major revision)	Computing	**JCR Q2**	B

Continued on next page

Table 3.2: Supporting publications of this dissertation. (Continued)

Improving the Energy Efficiency of Software-Defined Networks through the Prediction of Network Configurations	Published	Electronics	**JCR Q3**	C
Joint energy efficiency and load balancing optimization in hybrid IP/SDN networks	Published	Annals of Telecommunications	**JCR Q3**	D
Fog Node Placement in IoT Scenarios with Stringent QoS Requirements: Experimental Evaluation	Published	IEEE International Conference on Communications (ICC)	**GGS Class 2**	E
Multi-Objective Optimal Deployment of SDN-Fog Infrastructures and IoT Applications	Under review	IEEE International Conference on Communications (ICC)	**GGS Class 2**	F

Continued on next page

Table 3.2: Supporting publications of this dissertation. (Continued)

Privacy-Aware and Context-Sensitive Access Control for Opportunistic Data Sharing	Published	IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGrid)	**GGS Class 2**	G
Meeting Stringent QoS Requirements in IIoT-based Scenarios	Published	IEEE Global Communications Conference (GLOBECOM)	**GGS Class 2**	H
The Service Node Placement Problem in Software- Defined Fog Networks	Published	IEEE Symposium on Computers and Communications (ISCC)	**GGS Class 3**	I
Multi-Objective Genetic Algorithm for the Joint Optimization of Energy Efficiency and Rule Reduction in Software- Defined Networks	Published	International Conference on the Network of the Future (NoF)	International conference	J

Continued on next page

Table 3.2: Supporting publications of this dissertation. (Continued)

A Privacy-Aware Architecture to Share Device-to-Device Contextual Information	Published	IEEE International Conference on Smart Computing (SMARTCOMP)	International conference	K
Predicting Response Time in SDN-Fog Environments for IIoT Applications	Published	IEEE Latin-American Conference on Communications (LATINCOM)	International conference	L
On the tradeoff between load balancing and energy-efficiency in hybrid IP/SDN networks	Published	International Conference on Network of the Future (NoF)	International conference	M

Continued on next page

Table 3.2: Supporting publications of this dissertation. (Continued)

Quality of Service-Adaptive Industrial Internet of Things leveraging Edge Computing and Deep Reinforcement Learning: The Deep QoS-Adaptive Learning Environment (DeQALE) Architecture	Published	IEEE Iberian Conference on Information Systems and Technologies (CISTI)	International conference	N
Despliegue Óptimo de Aplicaciones IoT Distribuidas	Published	Jornadas de Ingeniería del Software y Bases de Datos (JISBD)	National conference	O
Optimizing the Response Time in SDN-Fog Environments for Time-Strict IoT Applications (Summary)	Published	Jornadas de Ciencia e Ingeniería de Servicios (JCIS)	National conference	P

Continued on next page

Table 3.2: Supporting publications of this dissertation. (Continued)

Optimizing the Response Time in SDN-Fog Environments for Time-Strict IoT Applications	Published	Jornadas de Ingeniería Telemática (JITEL)	National conference	Q
Joint Optimization of Response Time and Deployment Cost in Next-Gen IoT Applications (Summary)	Published	Jornadas de Ciencia e Ingeniería de Servicios (JCIS)	National conference	R
Deploying Next Generation IoT Applications Through SDN-Enabled Fog Infrastructures	Published	PhD Forum of IEEE International Conference on Pervasive Computing and Communications (PERCOM)	International workshop	S

A future is not given to you. It is something you must take for yourself.

NieR: Automata
Yoko Taro

Chapter 4

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

Authors : Juan Luis Herrera, Jaime Galán-Jiménez, Javier Berrocal, Juan Manuel Murillo

Publication type : Journal article

Journal : IEEE Internet of Things Journal

DOI : `https://doi.org/10.1109/JIOT.2021.3077992`

JCR : Q1 (Computer Science, Information Systems). IF: 10.238.

Notes : This article was sent before obtaining the grant from the Valhondo Calaff Foundation, and hence, the pre-print does not include them in the acknowledgments. Nonetheless, the edited, published version that can be consulted using the DOI includes an acknowledgment to the Valhondo Calaff Foundation.

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*—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 for correctly solving the DCDP. Software-defined networking, a paradigm that allows networks to be programmed through SDN controllers, allows for programmable routing optimiza-

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

tion [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 servers to provide commands accordingly. This application was designed using a microservice architecture (MSA) and

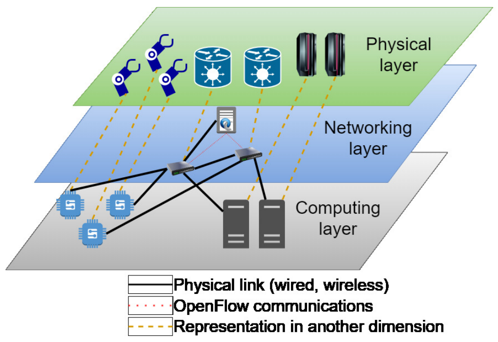


Figure 1: The presented scenario with the three involved layers.

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.

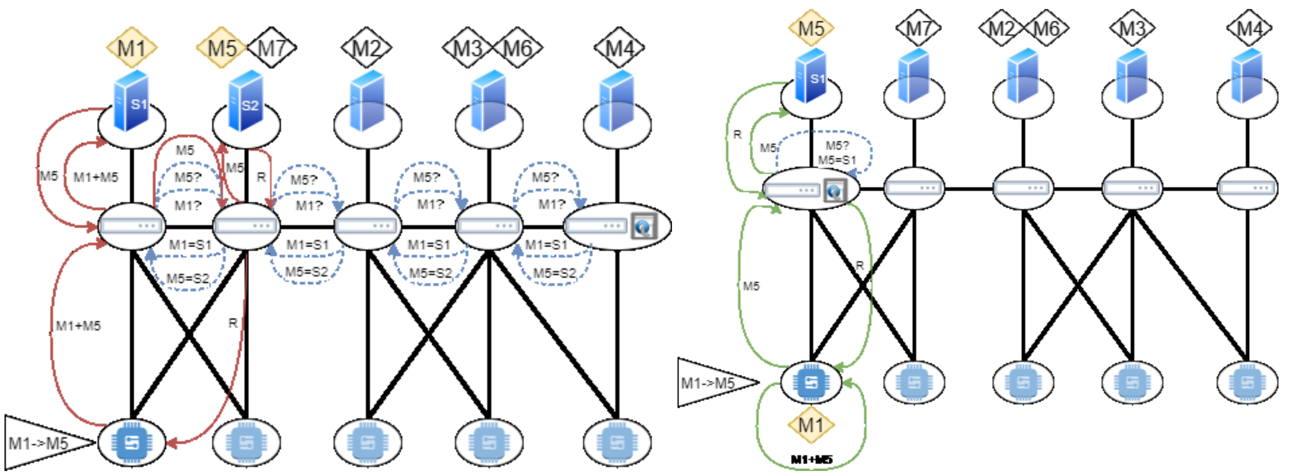
B. DCDP: Placing the microservices

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



(a) Fog deployment, suboptimal controller placement.

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

Figure 2: Workflow execution with two different deployment strategies.

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:

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

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:

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

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.

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

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: $z_{hc_a}'^{iw} = z_{hc_{a-1}}^w(1-z_{ic_a}^w)$, $z_{hc_a}''^{iw} = z_{hc_{a-1}}^w z_{ic_a}^w$. For them to have these values, they must follow these constraints:

$$-z_{hc_{a-1}}^w + z_{hc_a}'^{iw} \leq 0 \quad (9)$$

$$-1 + z_{ic_a}^w + z_{hc_a}'^{iw} \leq 0 \quad (10)$$

$$z_{hc_{a-1}}^w + 1 - z_{ic_a}^w - z_{hc_a}'^{iw} \leq 1 \quad (11)$$

$$-z_{hc_{a-1}}^w + z_{hc_a}''^{iw} \leq 0 \quad (12)$$

$$-z_{ic_a}^w + z_{hc_a}''^{iw} \leq 0 \quad (13)$$

$$z_{hc_{a-1}}^w + z_{ic_a}^w - z_{hc_a}''^{iw} \leq 1 \quad (14)$$

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

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

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.

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

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

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

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.

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

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).

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

(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 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;

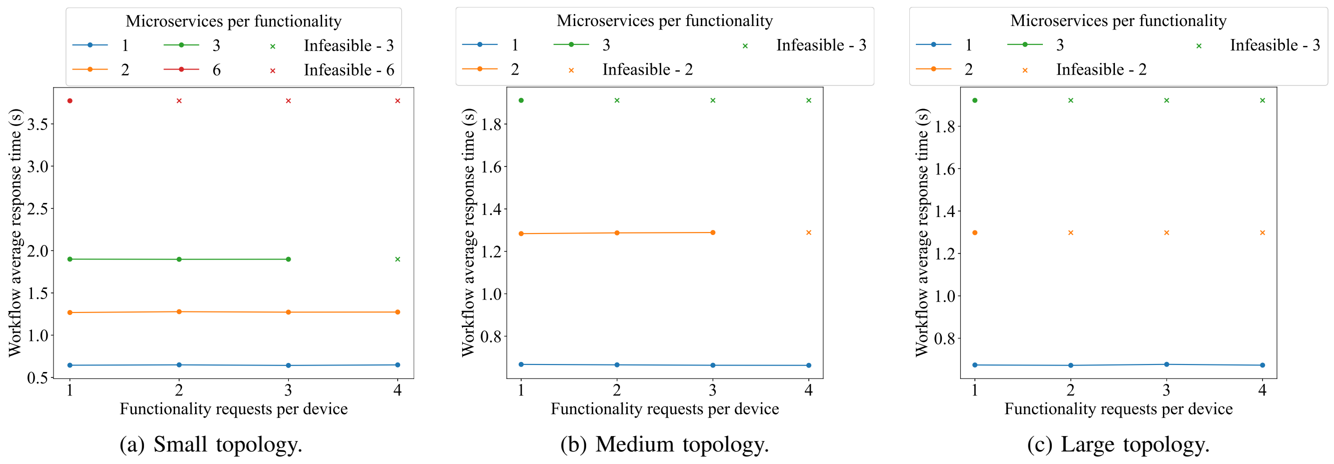


Figure 3: Request scalability analysis.

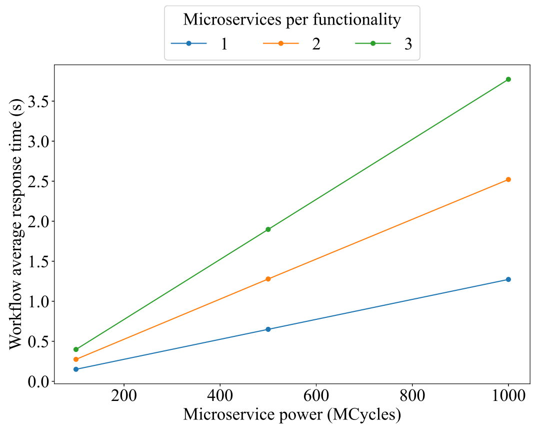


Figure 4: Computational scalability analysis in the small topology.

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 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 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

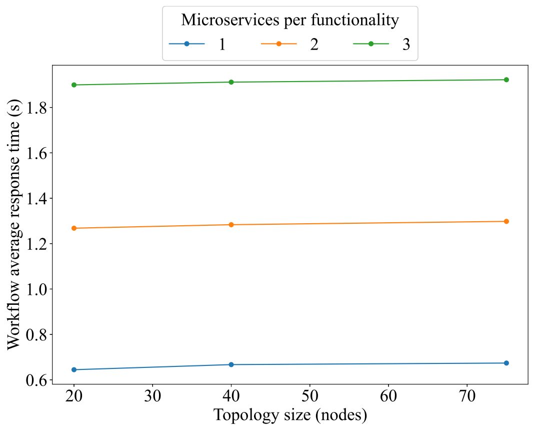


Figure 5: Comparison of response times under different topologies.

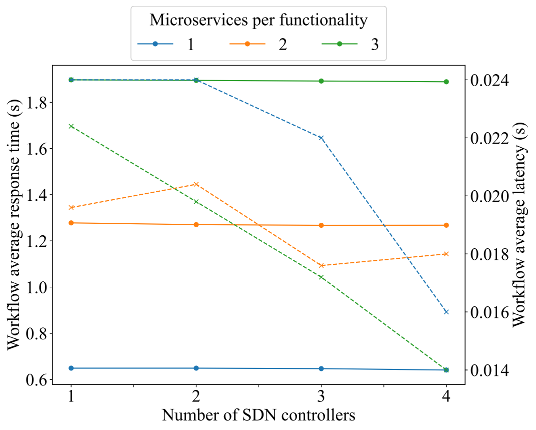


Figure 6: Latency and controller analysis in the small topology. Solid lines show the response time, and dashed lines show the latency.

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 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,

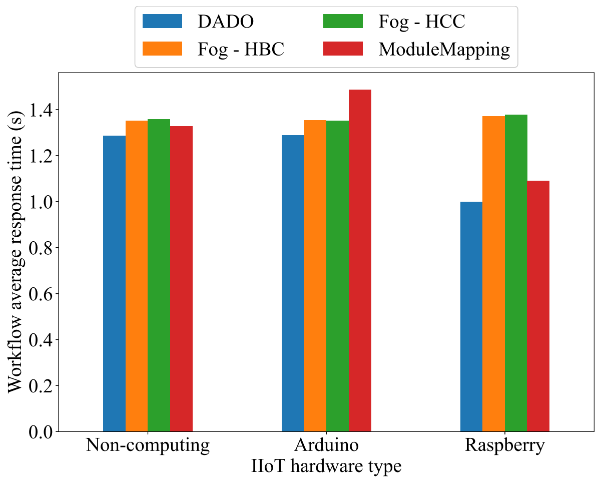


Figure 7: Performance benchmark of DADO in the medium topology.

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. 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

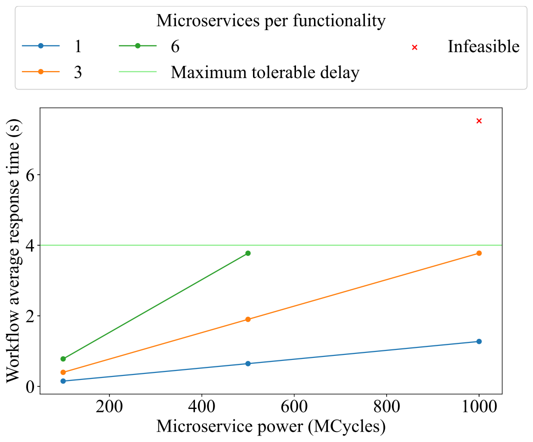


Figure 8: Maximum tolerable delay compliance of DADO in the small topology.

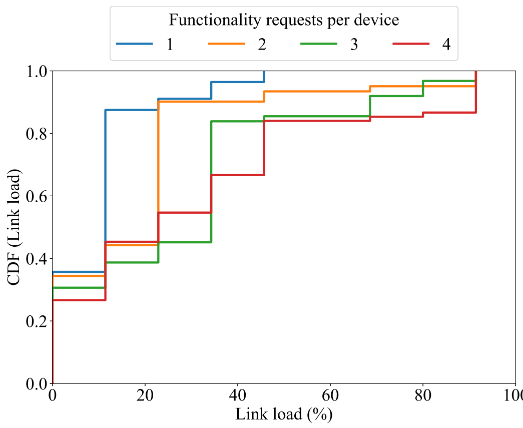


Figure 9: Empirical CDF of the link load in the medium topology.

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

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. 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 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

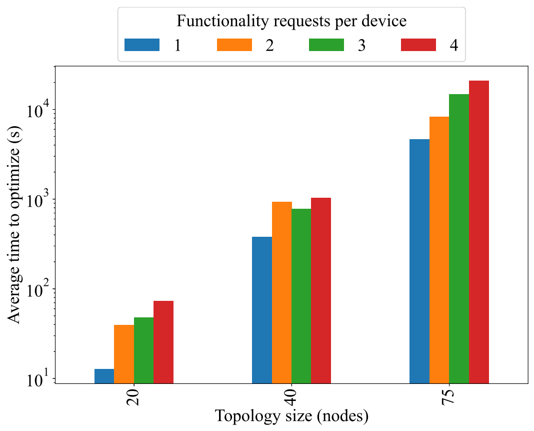


Figure 10: Average optimization times of DADO.

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: 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

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 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.

There are not more than five primary colors, yet in combination, they produce more hues than can ever be seen.

The Art of War
Sun Tzu

Chapter 5

Joint Optimization of Response Time and Deployment Cost in Next-Gen IoT Applications

Authors	: Juan Luis Herrera, Jaime Galán-Jiménez, Javier Berrocal, Juan Manuel Murillo
Publication type	: Journal article
Journal	: IEEE Internet of Things Journal
DOI	: `https://doi.org/10.1109/JIOT.2022.3165646`
JCR	: Q1 (Computer Science, Information Systems). IF: 10.238.

Joint Optimization of Response Time and Deployment Cost in Next-Gen IoT Applications

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

***Abstract*—The irruption of the Internet of Things (IoT) has attracted the interest of both the industry and academia for their application in intensive domains, such as healthcare. The strict Quality of Service (QoS) requirements of the next generation of intensive IoT applications requires the QoS to be optimized considering the interplay of three key dimensions: computing, networking and application. This optimization requirement motivates the use of paradigms that provide virtualization, flexibility and programmability to IoT applications. In the computing dimension, paradigms such as edge or fog computing, Software-Defined Networks in the networking dimension, along with micro-services architectures for the application dimension, are suitable for QoS-strict IoT scenarios. In this work, we present a framework, named Next-gen IoT Optimization (NIoTO), that considers these three dimensions and their interplay to place micro-services and networking resources over an infrastructure, optimizing the deployment in terms of average response time and deployment cost. The evaluation of NIoTO in a healthcare case study reveals a response time speed-up of up to 5.11 and a reduction in cost of up to 9% with respect to other state-of-the-art techniques.**

***Index Terms*—Internet of Things, Software-Defined Networks, Computing in the Network, Edge Computing, Fog Computing, Quality of Service**

I. Introduction

The increase in the number of Internet-connected devices in recent years, especially caused by the irruption of the Internet of Things (IoT) paradigm, has led to an exponential growth of the amount of traffic flowing through the network. In particular, it is expected that the number of connected devices will be more than three times the global population by 2023. Indeed, Machine-To-Machine (M2M) connections will conform the half of such connected devices (reaching to 14.7 billion M2M connections) in that year [1].

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 has been partially funded by the project RTI2018-094591-B-I00 (MCI/AEI/FEDER,UE), 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 (GR21133, IB18030), by the Valhondo Calaff institution, 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

The computational IoT tasks and the related data processing may be computationally intensive, which can not be often accomplished by regular IoT devices with limited resources (memory, battery, etc.) [2]. To overcome this difficulty these tasks are usually offloaded to the cloud, where they can be executed without compromising available resources. However, a penalty derived from the latency imposed by the separation of end devices and the cloud must be paid.

While this approach can be easily used with elastic applications that do not have strict Quality of Service (QoS) requirements (e.g., voice assistants), next-generation IoT applications (e.g., autonomous driving, Internet of Medical Things (IoMT) applications) require greater bandwidth and ultra low-latency constraints that are not feasible with a pure cloud-based paradigm [3]. Edge and fog computing paradigms represent a suitable solution for these intensive computational tasks that must be executed under strict latency requirements [3], [4]. By moving services from the cloud to the edge of the network some benefits are obtained: i) a reduction in the required latency; and ii) the computational load of tasks is restrained at end devices, since they are offloaded to edge servers [2].

Nonetheless, the time required to execute a request in a distributed application, or response time, has two components: latency, and execution time. While the former depends on the networking fabric and the distance between end devices and those they offload their tasks to (e.g., cloud, edge nodes), the latter depends on the computational load of these tasks and the power of the devices running them. Therefore, three dimensions are involved in the QoS of IoT applications: the computing dimension, the networking dimension and the application dimension. Furthermore, the next generation of IoT applications require for virtualization, flexibility and programmability features in these three dimensions [5]. Emerging paradigms, such as edge and fog computing in the computing dimension, Software-Defined Networking (SDN) for the networking dimension, and micro-services architectures in the application dimension, can be enablers for the QoS-optimal deployment of next-gen IoT applications [2], [6]–[9]. However, the QoS experienced by applications running in such architectures depends on the performance provided by the set of computing and networking resources. Analogously, such performance depends on the way micro-services and the SDN controller are placed [4], [7]. Moreover, micro-services may be replicated to improve the QoS, at the cost of assessing the number of replicas that must be deployed for a given scenario. Therefore, in order to optimally deploy IoT applications through all three dimensions, the problem of placing micro-services as well as the SDN controller to maximize the experienced QoS must be carefully addressed.

In this paper, we provide a framework named Next-gen IoT Optimization (NIoTO) to optimize the placement of micro-services and networking resources to optimize the QoS considering the computing, networking and application dimensions. In particular, the proposed solution provides i) the optimal number of micro-service replicas to be placed in the infrastructure, ii) the optimal placement of micro-services to be run in the computing resources of the infrastructure, iii) the optimal placement of the SDN controller to reduce the required signalling cost, and iv) the optimal routing of the traffic flows generated by micro-services, as well as control traffic flows. To do so, the problem is modeled and formulated using Mixed-Integer Linear Programming (MILP). Based on a scenario where next-gen IoT applications are considered, the framework is evaluated considering two metrics: i) the minimization of the average response time, and ii) the minimization of the costs derived from the deployment of the architecture. Finally, we remark the benefits of jointly considering the application, computing and networking dimensions to handle today IoT applications with stringent QoS requirements. The results obtained in the evaluation show that the use of NIoTO as an optimization tool is feasible at design-time. Furthermore, these results show that NIoTO is able to achieve shorter response times and lower costs compared to other, state-of-the-art benchmarks. The authors of this paper presented an initial work, the DADO framework, in [10], focused on the optimization of response time in IoT applications through optimal micro-service deployment and SDN controller placement. NIoTO extends DADO by considering deployment cost along with response time, including the possibility of multi-objective optimization, as well as trading off cost and response time. Furthermore, unlike DADO, NIoTO has been evaluated under a scenario that allows for combined fog and cloud microservice deployment. The main contributions of this work are as follows:

- The proposal of the NIoTO framework as a contribution to the optimization of the response time and deployment cost in next-gen IoT applications. NIoTO is able to optimally assess the number of micro-service replicas to deploy, where to deploy each of them, where to place the SDN controller, and the optimal routing of both application and control traffic flows. Furthermore, NIoTO supports the optimization of both, response time and deployment cost, by finding the optimal trade-off between them.
- The description and development of a problem model that considers both objectives supported by NIoTO. This model provides a holistic consideration of the optimization process, taking into the account the effects of one of NIoTO's decisions in a dimension on the rest of the dimensions. This model is developed through mathematical programming techniques, and provided as a mathematical problem formulation.
- The experimental evaluation of NIoTO in a healthcare-based next-gen IoT application, focusing on the trade-off between the two objectives supported by NIoTO and the performance differences between NIoTO and related, state-of-the-art frameworks.

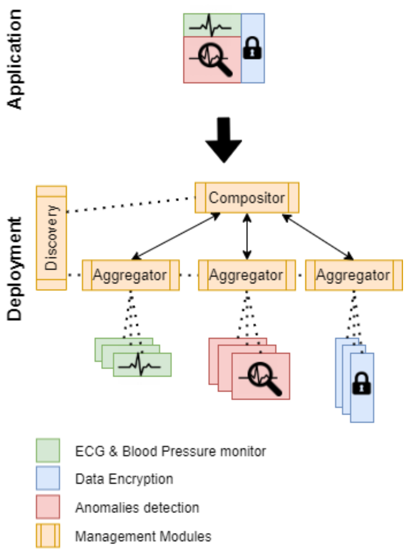
The remainder of the paper is organized as follows. Section II describes the architecture considered, remarking all the paradigms acting as enablers. Section III presents the proposed framework to optimize the application deployment, whose formulation is detailed in Section IV. Section V presents the performance evaluation of the framework over a healthcare scenario. Finally, Section VI concludes the paper.

II. Hierarchical multi-dimensional Architecture

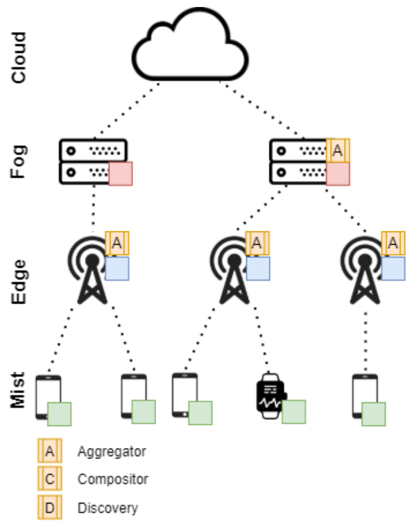
The development of QoS-stringent IoT applications requires the coordination of three highly related dimensions (Fig. 1): First, the application dimension, indicating how the software architecture of the application is modularized and coordinated. Second, the computing dimension defining the available computing resources and how these modules are deployed on them. Third, the networking dimension detailing which elements of the infrastructure will support the communications among modules. All these dimensions must be visualized in a holistic way. All of them are highly related and the configuration of one dimension impacts on the rest and, hence, the final QoS obtained.

The software architecture of next-gen IoT applications is usually based on the Service Oriented Computing paradigm (SOC) since they have to be massively distributed, interoperable and highly evolvable [8]. The Micro Service Architecture (MSA) pattern allows applications to be split into loosely coupled collaborating modules. These modules, usually called Bounded Contexts, APIs or, simply, services, contain one or more highly coupled micro-services that are usually deployed together [11]. MSAs are defined in contrast to monolithic architectures, in which the application may also be modularized, but all the modules require to be deployed together. Each of the micro-services in the MSA may be offered through different interfaces, which are normally comprised of a communication protocol and a data format. Some popular interfaces are web services (SOAP protocol and XML format) [12], gRPC services (HTTP/2 protocol and Protocol Buffers format) [13] or RESTful services (HTTP+JSON) [14]. While RESTful is currently the most popular interface and proposals for its use in IoMT exist [15], micro-services are not bound to a concrete interface.

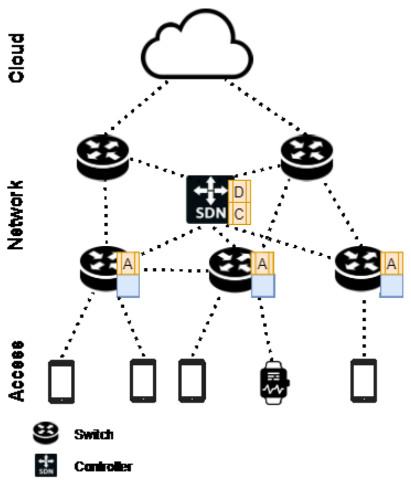
As a running example, which is depicted in Fig. 1, we base on an IoMT application to track the blood pressure of a patient and to detect anomalies in their electrocardiograms (ECG) by making use of data obtained through sensors. Each user is equipped with an IoT node that samples information for 15 seconds before sending it for further processing. The functionalities that are used in this case study are split into three services: ECG and blood pressure monitoring (green), data encryption (blue), and anomalies detection (red). Fig. 1a shows a high level architectural design of this application. This running example is based on the architecture proposed by [16].



(a) Application dimension.



(b) Computing dimension.



(c) Networking dimension.

Figure 1: Hierarchical IoMT application with the different dimensions affected to meet stringent QoS requirements.

In addition, their required data input, output and processing characteristics were obtained from [17].

These services can be deployed independently of one another, so that they can be deployed on the same or on different machines or servers, or even they can be replicated in order to balance the load and improve the QoS. Different virtualization and management technologies, such as Docker or Kubernetes, are used to reduce the effort and to automate this deployment. In addition, SOC makes use of some key technical foundations for the integration of the different services deployed [8], [18]: service discovery, for identifying the location of each service; service aggregation, for aggregating the responses; or service composition, coordinating the execution of complex functionalities by orchestrating several services.

The development of paradigms such as fog, edge or mist computing has allowed the development of intensive and QoS stringent IoT applications following a hierarchical architecture [2]. Services of MSA-based applications can be deployed closer to end-users in order to reduce network load and to improve response time [2]. The most resource-consuming services are still deployed on powerful nodes (such as cloud or, even, fog nodes) and the QoS stringent services, but less resource demanding, can be deployed on network nodes with computing capabilities –edge computing– or even on the Internet-connected devices themselves –mist computing– [2]. Therefore, a wide range of possibilities to deploy services must be evaluated, and in which nodes they are finally deployed is key to meet stringent QoS. Likewise, for complex functionalities, the deployed services may be coordinated, in order to achieve the functionalities and the workflows defined, using service discovery, aggregation and composition modules. The location of these management modules is also crucial in order to meet the required QoS.

Fig. 1b shows how the services defined on our running example are deployed on different layers of the computing hierarchical architecture depending on the required QoS and the available computing resources. For instance, the ECG and blood pressure monitoring service replicas (green service) is deployed on IoT devices, since this monitoring requires few computing resources. In addition, different aggregators (management modules) are also deployed on the upper layer (at the edge) to compute the responses of the IoT devices in order to reduce the data traffic, improving the response time. Likewise, data encryption (blue service) and anomaly detection (red service) both require some additional resources so that their replicas are deployed on more powerful nodes (edge and fog nodes, respectively). Therefore, defining an optimal computation distribution is highly dependent of the defined services and the available resources.

The deployment of an IoT application only taking into account the computing dimension does not always guarantee the optimal deployment in terms of QoS. In particular, the network configuration, such as the routing among the deployed services or the deployment of specific infrastructures, heavily impacts on the network latency and, hence, on the QoS. SDN networks allow the deployment of SDN controllers that leverage virtualization to make the network programmable based on the SOC principles [19]. Thus, Virtual Network Functions (VNFs) can be exploited for traffic engineering purposes in order to improve the network performance [20]. In this way, the SDN Controller Placement Problem (CPP) [7] identifies the deployment of SDN controllers in an optimal location to reduce the control latency and response time [21]. Fig. 1c shows that the SDN controller has been placed close to specific switches, in-between the Edge and Fog layers, in order to improve the average latency.

However, the network infrastructure has other capabilities that should also be reused. The application of virtualization and service oriented principles to the network infrastructure has led to a cloudification of both controllers and switches [19]. The Multi-Access Edge Computing (MEC) architecture embeds decentralized cloud capabilities in the network infrastructure at the network edge. This means that not only VNFs can be deployed on them, but also they can provision Virtual Computing Functions [19]. Therefore, IoT services and other management components (such as the service discovery, the aggregator or the compositor) can also be deployed in the network infrastructure. For instance, ETSI, the European Telecommunications Standard Institute, proposed the Management and Orchestration module (MANO) [22] to manage the different services and modules, their lifecycle, and to orchestrate them and chain different VNFs to support workflows. Therefore, the networking dimension is also able to perform some management tasks for the IoT applications. Fig. 1c shows that the service discovery and the service compositor modules have been deployed on the SDN controller in order to improve the QoS of the considered IoT application.

Therefore, a hierarchical multi-dimensional architecture boosts the achievement of stringent QoS requirements of next-gen IoT applications. Nevertheless, in order to achieve the optimal deployment and configuration, several dimensions must be jointly evaluated: the application dimension, the computing dimension and the networking dimension. In this paper, we base on an modularized IoT application using the principles of SOC and MSA [11], and propose a framework for the optimal placement of IoT services and SDN controllers in the hierarchical multi-dimensional architecture.

III. Next-gen IoT Optimization

Providing the best QoS requires a deployment that jointly evaluates the application, computing and networking dimensions to optimally place IoT services and SDN controllers. In this paper, we present the Next-gen IoT Optimization (NIoTO) framework, which takes all the three dimensions into account to find feasible deployments that meet the specific QoS requirements of IoT applications. Although NIoTO is an extensible framework that can support different types of QoS, in this paper, we focus on two specific requirements: response time and cost. These parameters are usually very important for IoT applications, as well as highly related —i.e, a lower response time usually requires a higher cost— and, therefore, a trade-off between them is difficult to achieve [23]. NIoTO takes as input the characteristics of the application, the network topology, the computing resources, and the QoS objectives to satisfy. These inputs are processed by the framework, which optimizes the QoS considering the three dimensions. Finally, results are reported as output, detailing the placement of the IoT services and SDN controllers to provide the optimal QoS according to the objectives.

A. Inputs

In order to know the optimal deployment design of an IoT application, different information is required for the three considered dimensions. These inputs are split into two types of information in order to improve its reusability and the extensibility of NIoTO to support other potential QoS requirements. These pieces of information are: *basic* information, and information that is *specific* to a QoS objective. The inputs required by NIoTO, split by their information type and their dimension, are depicted in a tree diagram in Fig. 2. These inputs are fed to NIoTO using open data interchange documents, such as XML or JSON, structured with a concrete schema, in which an arbitrary set of elements (e.g., micro-services, SDN switches, links, computing resources) can be defined. For each of these elements, the basic information, stored as its attributes, is mandatory, while QoS-specific information depends on the QoS to be optimized. Moreover, elements can be cross-referenced using their IDs, enhancing their reusability (e.g., a computing resource can be defined only once, and network links can refer to the definition through its ID). The use of non-proprietary data interchange formats eases the integration of NIoTO with other tools, as well as the creation of parsers to convert other formats to NIoTO inputs and vice-versa, making it easier to configure and use NIoTO. It is also possible to create support tools that assist the NIoTO user on the provision of the inputs (e.g., graphical user interfaces for NIoTO) using these data formats. Furthermore, each of the inputs are to be obtained at design-time. Some of these inputs can be obtained directly, while other inputs need to be estimated. For this estimation, the application is expected to be at the late design phase of development, in which the architectural decisions are already taken and low-level design has also been performed. Therefore, details such as the system's software architecture, the roles and functionalities of the micro-services, their complexity, or the planned network and computing infrastructure are known. Such knowledge is key, as it enables for the obtention of most parameters from the computing and networking dimensions directly. At this stage, the analysis of resource consumption, size, and performance in the application dimension, if performed with adequate techniques such as [24]–[26], can yield realistic estimates. The manner of obtention for each of the inputs, as well as its source, are detailed in Tab. I.

Basic information is always required independently from the QoS requirements to optimize. For the application dimension, the data required is the amount of RAM consumed by each service, the size of its inputs and outputs, and the requests that should be processed by each service (i.e., the workflows that are requested, and which device requests which workflow). For the computing dimension, the available computing resources (indicating their available RAM) and their location in the network topology must be provided as input. Finally, for the networking dimension, the network graph with the set of nodes and links, as well as link capacities, must be detailed. The three dimensions are used to specify the basic information in order to reduce the coupling. With this information, the feasibility of the deployment for the given IoT application is evaluated by checking if available resources are not exceeded.

QoS *specific* information are those inputs required to identify the optimal deployment for a particular type of QoS. Currently, NIoTO supports next QoS objectives: response time, deployment cost, and both of them (to find the best compromise between minimal response time and cost). Response time is calculated as the average response time for each request, which is the sum of the time needed to execute every service

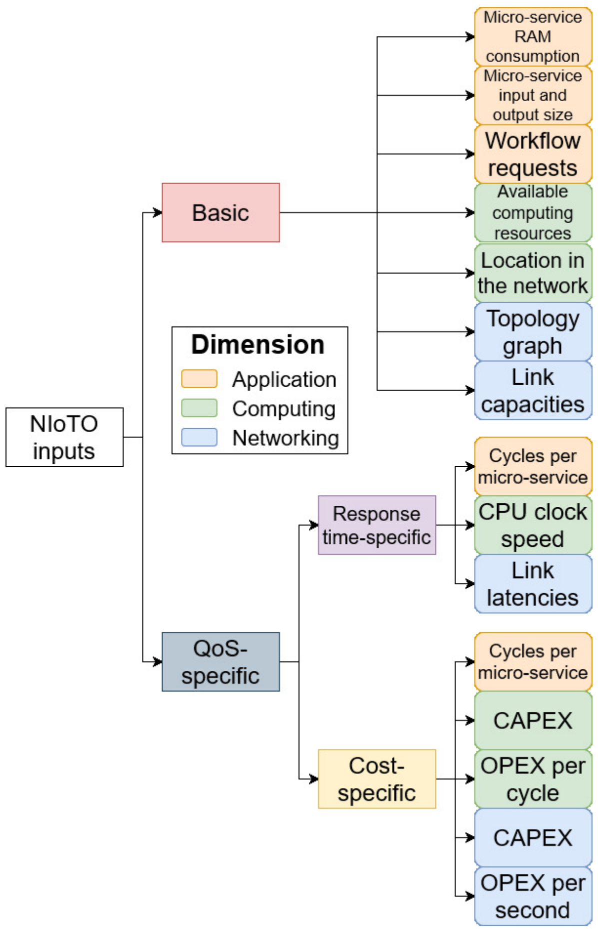


Figure 2: Inputs of the NIoTO framework per type of information and dimension.

in each request and the time needed to communicate to the devices where the services of such request are deployed. In order to evaluate this first objective, the inputs required for each of the three dimensions are: i) application dimension, the number of cycles required to execute each service; ii) computing dimension, the CPU clock speed of each computing resource; iii) networking dimension, the latency of each link. The cost objective, on the other hand, is assessed as the sum of the capital expenditures (CAPEX) (i.e., the cost of acquiring an asset) of each element used in the infrastructure plus the operational expenditures (OPEX) (i.e., the ongoing cost of maintaining an asset) derived from their use. Thus, for the application dimension, the number of cycles required by each service is required again. Instead, computing resources must include their CAPEX and OPEX per cycle. Network equipment must also report their CAPEX, as well as their OPEX per second, to optimize this objective.

To execute the NIoTO framework, to find feasible deployments, only basic information is required. If any given QoS objective must be optimized too, the QoS dependent information of said objectives has to be provided as well.

Table I: Kind of obtention and source of the input information for NIoTO at design-time.

Input information	Kind of obtention	Information source
Micro-service RAM consumption	Estimation	Memory complexity analysis of each micro-service
Micro-service input and output size	Estimation	Analysis of the expected input and output data type of each micro-service
Cycles per micro-service	Estimation	Time complexity analysis of each micro-service
Workflow requests	Estimation	Definition of the permitted workflows (use cases) for the application, estimation of the application's user base
Available computing resource	Direct	Planning of the computing devices to use, analysis of the datasheets and documentations of each planned device
Location of computing devices in the network	Direct	Network planning
CPU clock speed	Direct	Planning of the computing devices to use, analysis of the datasheets and documentations of each planned device
CAPEX of computing devices	Direct	Provided by the supplier
OPEX of computing devices	Direct (on demand)/Estimated (self-hosted)	Provided by the supplier (on demand)/planning of the computing devices to use, analysis of the datasheets and documentations of each planned device, calculation of derived cost (self-hosted)
Topology graph	Direct	Network planning
Link capacities	Direct	Planning of communication technologies, analysis of the capacity permitted by each planned technology
CAPEX of network equipment	Direct	Provided by the supplier
OPEX of network equipment	Estimated	Network planning, analysis of the datasheets and documentations of each planned networking device, calculation of derived cost

B. *NIoTO framework*

NIoTO takes all the inputs and provides the optimal deployment plan meeting the defined QoS objectives. To do so, NIoTO considers how services need to be executed in the computing resources, how executing these services in certain computing resources generates network traffic, and how the network manages such traffic. Following the example defined in Sec. II: given an example request for detecting anomalies in an ECG, if both services are deployed in the same machine, traffic will be sent back and forth between the IoMT device and said machine. However, if each service is deployed in a different machine, an additional traffic flow between both machines is required as well. The best deployment depends on (according to the QoS objective that has been set) the QoS the network is able to provide depending on the SDN controller placement and the QoS provided by each machine. NIoTO automatically selects the deployment that provides the best QoS considering all the three dimensions and their interplay. Nevertheless, a deployment plan that is optimal at some specific moment could be sub-optimal when the application context changes. Therefore, the IoT application deployment plan should be continuously evaluated and modified to ensure that the defined QoS is always met. There are two key phases with different time constraints to find the optimal deployment: design-time, and run-time. During the design phase, the application and infrastructure barely change their estimations, and there is no strict time limitation for obtaining the optimal deployment plan. Instead, during the execution, the infrastructure is dynamic and the monitored data often changes, and thus deployment plans are periodically needed. These periods cannot be very long, especially in cases in which a sudden spike in requests may require a deployment adaptation to maintain the QoS. The version of NIoTO presented in this paper is only suitable for design time optimization, due to the techniques used on its implementation. Nonetheless, the development and implementation of a version suitable for execution time, making use of other techniques that allow for faster optimizations, is one of our main future works.

To identify the optimal deployment plan at design-time, NIoTO makes use of MILP. This technique guarantees that the solution given as output meets all the constraints and is an optimal solution considering the given objective function. In order to support the three considered objectives (response time, cost and both), three different objective functions are defined. The MILP formulation of NIoTO tries to find an optimal deployment that places services and SDN controllers in a feasible way, respecting next constraints: i) RAM resources are not exceeded; ii) each traffic flow, regardless if it is control traffic or not, has a single source, a single destination and it is fully routed through a set of links (i.e., it is not divisible); and iii) link capacities are not exceeded. Then, the QoS objective function is applied so that the deployment is both feasible and optimal. In the case that both objectives are considered at the same time, i.e., response time and cost, NIoTO tries to find the optimal trade-off between them. While using or adding more resources to the architecture provides better response time, it is also costly, and vice-versa , thus NIoTO considers both to find the best compromise.

C. Outputs

NIoTO's output is a set of the placement decisions, i.e., a deployment plan for both IoT services and SDN controllers. The deployment plan follows certain patterns. For instance, each service replica needs to be deployed on the machine (or machines) it will be executed, the SDN controllers must be co-located with switches [7], each SDN switch is under the control of a single given SDN controller and traffic must be steered from the device requesting a service (or a set of them) to the machine where it is executed.

At design time, the deployment plan is a guide for the application's operation engineer, system administrator and network administrator. Information about which elements and how they should be used, as well as how to make an initial IoT application and SDN controller deployment are provided.

IV. Problem formulation

In this section, the MILP formulation that the NIoTO framework uses at design time is detailed. This version holistically optimizes the application, computing and networking dimensions. This formulation optimizes the number of microservice replicas required in the application dimension, where in the computing dimensions are each of the replicas deployed, as well as SDN controller placement and traffic routing in the networking dimension. This formulation is structured into four tightly coupled elements: parameters, decision variables, objective function, and constraints, that will be explained in the same order. Moreover, Table II provides a summary and quick reference of the notations used throughout the formulation.

A. Parameters

The parameters of a MILP formulation are its inputs: values that are provided to the formulation at run-time and stay fixed during the MILP solving process. Thus, the parameters of the formulation are the inputs of the NIoTO framework.

Let the deployment infrastructure be represented as a graph $G = \{V, L\}$. Let V be the set of vertices, which comprises both the computing and networking dimensions. The term *vertices* is used instead of *nodes* to avoid confusions with the term *fog nodes*. Let C be the set of computing devices in the infrastructure, and let S be the set of SDN switches, so that $V = C \cup S; C \cap S = \emptyset$. Furthermore, let L be the links that connect the infrastructure's vertices, i.e., $l_{ij} \in L; i, j \in V$.

The basic information of each computing device $c \in C$ comprises its amount of available RAM, r_c, measured in bytes. For the response time objective, QoS-specific information includes the device's CPU clock speed in Hz, P_c. On the other hand, deployment cost-specific information comprises the CAPEX of the device, $CAPEX_c$, and the device's OPEX per cycle, $OPEX_c^{\Omega}$.

For each SDN switch $s \in S$, its basic information is related to its position within the infrastructure, which is already represented in G. Thus, only cost-specific information is required: its CAPEX $CAPEX_s$, the CAPEX of an SDN controller placed in the switch $CAPEX_s^{CNT}$, its OPEX per second $OPEX_s$, and the analoguous OPEX of a controller $OPEX_s^{CNT}$. Response time QoS-specific information, such as control latency, depends on controller placement, and thus cannot be known a priori. Therefore, control latency is not an input, it is an internal calculation of the formulation instead. Finally, the size of the control packets used on the SDN

Table II: List of formulation notations.

Parameter	Meaning
G	Graph that represents the infrastructure
V	Set of vertices of G
C	Subset of V that are computing devices.
S	Subset of V that are SDN switches
L	Set of links of G
W	Set of workflows of the IoT application
r_c	RAM memory of the computing device c
P_c	CPU clock speed of the computing device c
$CAPEX_c$	CAPEX of the computing device c
$OPEX_c^{\Omega}$	OPEX per cycle of the computing device c
$CAPEX_s$	CAPEX of the SDN switch s
$OPEX_s$	OPEX of the SDN switch s
$CAPEX_s^{CNT}$	CAPEX of placing a controller on the SDN switch s
$OPEX_s^{CNT}$	OPEX of the controller placed on the SDN switch s
σ	Size of the SDN control packets
θ_{ij}	Capacity of the link l_{ij}
δ_{ij}	Latency of the link l_{ij}
$WS(w,c)$	Boolean function that indicates whether the workflow w is requested by the device c or not
I_m	Size of the input of the micro-service m
O_m	Size of the output of the micro-service m
Ω_m	CPU cycles of the micro-service m
ϵ_{OBJ}	Boolean that indicates whether the QoS objective OBJ is enabled or not
u_c	Boolean that indicates whether the computing device c is in use or not
u_s	Boolean that indicates whether the SDN switch s is in use or not
Decision variable	**Meaning**
$z_{cm_a}^{w}$	Boolean to determine if the micro-service m_a of the workflow w is deployed to the computing device c
$f_{ij}^{cwm_a}$	Boolean to determine if the traffic generated by the computing device c as a consequence of the micro-service m_a of the workflow w is routed through the link l_{ij}
x_s	Boolean to determine if a controller is placed on the SDN switch s
$y_{ss'}$	Boolean to determine if the SDN switch s is mapped to the controller placed on SDN switch s'
cf_{ij}^{s}	Boolean to determine if the control traffic generated by the SDN switch s is routed through the link l_{ij}

network, σ, is required as basic information. Nonetheless, σ does not depend on the SDN controller's deployment, but on the version of the OpenFlow protocol used [27].

The links that bind the infrastructure's vertices together have a limited available capacity θ_{ij}, which is part of their basic information. For the response time objective, each link's latency δ_{ij} must be known.

On the application dimension, NIoTO models execution as *functionalities*. A functionality, or *workflow*, is a request for the execution of a micro-service, or the execution of a set of micro-services in an ordered, pipelined manner. Following the example from Sec. II, a functionality may request for the execution of the ECG analyzer, the anomaly detector and the encryption service, so that the commented ECG outputted by the first micro-service serves as input to the anomaly detector, and the anomaly information is later encrypted for its safe storage. Thus, let W be the set of workflows requested for the application. Each workflow $w \in W$ is requested by a certain computing device, therefore, let $WS(w,c)$ be a binary function that evaluates to 1 if c requests workflow w and 0 otherwise. Moreover, let each workflow $w \in W$ be an ordered set of micro-services $w = \{m_1, m_2, ..., m_{|w|}\}$.

Each of these micro-services requires basic information, namely, the size of its input and output data, I_m and O_m respectively, and the amount of RAM it consumes, r_m. The QoS-specific information for response time also includes the number of CPU cycles that the micro-service requires to fully execute, Ω_m. Micro-services do not need any cost-specific information, as the costs derived from micro-services are a consequence of their deployment and execution, and thus, depend on the resources from the networking and computing dimensions used to execute them.

Finally, in order to enable and weigh the importance the QoS objectives, we add a parameter named ϵ_{OBJ}, which is a positive real number, or 0. ϵ_{OBJ} represents the weight of the objective OBJ in the optimization. It is important to note that the sum of all the values of ϵ_{OBJ} through all objectives must be exactly 1.

B. Decision variables

Similar to how parameters are the inputs of the MILP formulation, decision variables can be seen as its output: a set of values that the formulation must manipulate in order to find the optimal combination of values. Nonetheless, MILP is only able to manipulate integer decision variables, thus conditioning the information representation of the output. Concretely, NIoTO makes use of binary decision variables to represent information.

Firstly, the micro-service replication deployment must be modeled. To do so, binary variables are used: let $z_{cm_a}^{w}$ be a binary variable that takes a value of 1 if the micro-service m_a of the workflow w is deployed to the computing device c, and 0 otherwise. These variables automatically account for the number of replicas: if different workflows have the same micro-service deployed to the same computing device, they are sharing a single replica. Otherwise, multiple replicas exist. Secondly, the traffic flows generated by the communications between computing devices (i.e., to request the execution of the next micro-service in the workflow to another computing device) must also be modeled. Thus, let $f_{ij}^{cwm_a}$ be a binary variable that takes the value of 1 if the traffic flow generated by the computing device c as a consequence of the execution of the micro-service m_a of the workflow w is routed through the link l_{ij}, and 0 otherwise. With these decision variables, NIoTO is able to plan the replication of micro-services at the application dimension, the deployment of the replicas at the computing dimension, and the routing of the application's traffic through the networking dimension.

Nonetheless, NIoTO must also take the control of the networking dimension into account, and thus, place SDN controllers accordingly. To do so, let x_s be a binary variable, that is 1 if there is an SDN controller placed in the switch s and 0 otherwise. Nonetheless, if multiple controllers are placed, the mapping between SDN controllers and switches (i.e., which controller is on charge of which switches) must also be found. Hence, let $y_{ss'}$ be a binary variable, which takes the value of 1 if switch s is mapped to controller s' and 0 otherwise. Finally, NIoTO needs to account for the control traffic flows generated by the communications between switches and controllers to optimally route it. Let cf_{ij}^{s} be a binary variable that takes the value of 1 if the control traffic generated by switch s is routed through link l_{ij} and 0 otherwise.

C. Objective function

While MILP formulations manipulate the values of their decision variables to provide optimal outputs, they require a manner to calculate the optimality of the output. This manner is the objective function, which is a function of the decision variables. It is important to note that MILP only supports linear functions, i.e., it is strictly forbidden to multiply or non-linear operations over decision variables (e.g., multiplying two decision variables).

To better understand the objective function of NIoTO, we first define different calculations that will later be integrated. First, in order to calculate the execution time of a micro-service, the number of cycles of the micro-service need to be divided by the CPU clock speed of the computing device it is deployed to. Thus, if the micro-service m_a is deployed to the computing device c, then its execution time is $\frac{\Omega_{m_a}}{P_c}$. However, it is not possible to know *a priori* where each micro-service is executed. What can be known is that, given that m_a is part of workflow w, the variable $z_{cm_a}^{w}$ will have the value 1 if it deployed to c and 0 otherwise. Based on this knowledge, we can calculate the total execution time of a micro-service in a workflow by calculating the sum of all possible execution times in all computing devices, multiplied by $z_{cm_a}^{w}$:

$$EXEC_{m_a}^{w} = \sum_{c\in C} \frac{\Omega_{m_a}}{P_c} z_{cm_a}^{w}$$

Therefore, we define the execution time of a workflow as the sum of the execution times of its micro-services:

$$EXEC_{w} = \sum_{a=1}^{|w|} EXEC_{m_a}^{w}$$

Next, workflow latency needs to be calculated. In the case of a single flow that traverses a single link l_{ij}, latency is defined as the latency of the link, i.e., δ_{ij}. In a similar manner to the z variable of micro-services, the binary variable that contains the information of whether a flow traverses a link or not is $f_{ij}^{cwm_a}$. Nonetheless, there is an exception in the case of latency: if the flow reaches an SDN switch, i.e., if j is an SDN switch, we need to calculate the control latency of the switch, as it may need to communicate with the SDN controller. In this case, the binary variables for SDN control traffic flows is cf_{ij}^{s}. Thus, we can define the SDN control latency of a switch as:

$$CNTLAT_{s} = \sum_{l_{ij}\in L} cf_{ij}^{s}\delta_{ij}$$

Thus, with a known control latency, the latency of a traffic flow from a micro-service to another in the context of a workflow is is: i) its own latency if j is not an SDN switch, or ii) its own latency plus the control latency of j otherwise. To define it with more ease, let $SW(i)$ be a function which yields 1 if $i \in S$ and 0 otherwise. Formally, it is possible to denote this latency as:

$$LAT_{m_a}^{w} = \sum_{c\in C}\sum_{l_{ij}\in L} (f_{ij}^{cwm_a}\delta_{ij} + SW(j)CNTLAT_j)$$

Hence, to calculate the total latency of a workflow is the sum of the latencies of its micro-services:

$$LAT_{w} = \sum_{a=1}^{|w|} LAT_{m_a}^{w}$$

Thus, we define the average response time of the deployment as the average response times of all the workflows requested, each of them being the sum of the execution time and latency of the workflow:

$$RT = \frac{1}{|W|}\sum_{w\in W} EXEC_w + LAT_w$$

For the deployment cost objective, we need to split it in CAPEX and OPEX. In both cases, we assume that, since NIoTO operates at design time, any equipment that is not used will not be acquired, and hence, only the CAPEX of the used equipment should be considered. Thus, there is a need to know whether an element is in use or not. In the case of computing devices, we assume that one is used if it either requests at least one workflow, or runs at least one micro-service:

$$u_c = \max(\max_{w\in W, a\in[1,|w|]}(z_{cm_a}^{w}), \max_{w\in W}(WS(w,c)))$$

In the case of switches, a switch is used if it belongs to the route of at least one traffic flow, be it an application or a control flow:

$$u_s = \max(\max_{l_{is}\in L, c\in C, w\in W, a\in[1,|w|]}(f_{is}^{cwm_a}), \max_{l_{is}\in L, s'\in S}(cf_{is}^{s'}))$$

Thus, to obtain the CAPEX, we must multiply the CAPEX of each of the elements by 0 if they are unused, and by 1 if they are in use, and then sum the results. Formally:

$$\begin{aligned} CAPEX = &\sum_{c\in C}(CAPEX_c u_c) + \\ &\sum_{s\in S}(CAPEX_s u_s + CAPEX_s^{CNT} x_s) \end{aligned}$$

For the OPEX, we also need to account for the usage in cycles of the computing devices. Formally:

$$\begin{aligned} OPEX = &\sum_{c\in C}(\sum_{w\in W}\sum_{a=1}^{|w|} OPEX_c^{\Omega}\Omega_{m_a} z_{cm_a}^{w}) + \\ &\sum_{s\in S}(OPEX_s u_s + OPEX_s^{CNT} x_s) \end{aligned}$$

For the final objective function, we should take into account whether each of the QoS objectives is or is not enabled. Moreover, it is recommended to normalize each of the objectives so the final values on each side are within the same range (e.g., between 0 and 1), especially if both objectives are enabled. The final objective function is shown in Equation (1).

$$\min \epsilon_{RT} RT + \epsilon_{COST}(CAPEX + OPEX) \quad (1)$$

D. Constraints

With only parameters, decision variables and an objective function, an MILP formulation will find the combination of values for the variable that optimizes the objective function. However, this may lead to illegal situations in our problem: the limited capacities of computing devices and links could be surpassed, micro-services could have zero replicas, the SDN controller could be undeployed, switches may not be related to SDN controllers... In order to make the solution legal within the problem, we enforce these rules through constraints.

The constraints of NIoTO are as follows:

$$\sum_{c\in C} z^{w}_{cm_a} = 1 \forall w \in W, a \in [1, |w|] \tag{2}$$

$$\sum_{w\in W}\sum_{a=1}^{|w|} z^{w}_{cm_a} r_{ma} \leq r_c \forall c \in C \tag{3}$$

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

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

$$\sum_{j\in V} f^{cwm_1}_{ij} - f^{cwm_1}_{ji} = \begin{cases} 0 & \text{if } i \in S \\ WS(w,c)(1 - z^{w}_{im_1}) & \text{if } i = c \\ -WS(w,c) z^{w}_{im_1} & \text{otherwise.} \end{cases} \quad \forall i \in V, c \in C, w \in W \tag{6}$$

$$- z^{w}_{cm_{a-1}} + z'^{iw}_{cm_a} \leq 0 \tag{7}$$

$$- 1 + z^{w}_{im_a} + z'^{iw}_{cm_a} \leq 0 \tag{8}$$

$$z^{w}_{cm_{a-1}} + 1 - z^{w}_{im_a} - z'^{iw}_{cm_a} \leq 1 \tag{9}$$

$$- z^{w}_{cm_{a-1}} + z''^{iw}_{cm_a} \leq 0 \tag{10}$$

$$- z^{w}_{im_a} + z''^{iw}_{cm_a} \leq 0 \tag{11}$$

$$z^{w}_{cm_{a-1}} + z^{w}_{im_a} - z''^{iw}_{cm_a} \leq 1 \tag{12}$$

$$\sum_{j\in V} f^{cwm_a}_{ij} - f^{cwm_a}_{ji} = \begin{cases} 0 & \text{if } i \in S \\ z'^{iw}_{cm_a} & \text{if } i = c \\ -z''^{iw}_{cm_a} & \text{otherwise.} \end{cases} \quad \forall i \in V, c \in C, w \in W, a \in [2, |w|] \tag{13}$$

$$\sum_{j\in V} cf^{s}_{ij} - cf^{s}_{ji} = \begin{cases} 0 & \text{if } i \in C \\ 1 - y_{si} & \text{if } i = s \\ -y_{si} & \text{otherwise} \end{cases} \quad \forall i \in V, s \in S \tag{14}$$

$$\sum_{c\in C}\sum_{w\in W} [(\sum_{a=1}^{|w|} f^{cwm_a}_{ij} I_{m_a})] + \sum_{s\in S}[cf^{s}_{ij}\sigma] <= \theta_{ij} \quad \forall l_{ij} \in L \tag{15}$$

Equation (2) guarantees that each micro-service in a workflow is instantiated exactly once. Equation (3) enforces the RAM limit on the computing devices. Equation (4) makes sure each switch is controller by a single SDN controller, which must first be deployed as of Equation (5). Equation (6) guarantees that the traffic flow of starting a workflow has the workflow's requester as its source and the computing device that has deployed the first micro-service of the workflow as its destination, which is an special case of the general flow constraint from Equation (13). However, for micro-services beyond the first of a workflow, it is required to know if the machine that executed the previous micro-service is the same as the one hosting the current one, which would require variable multiplication. Since only linear constraints can be used in MILP, Equations (7-12) are used to *linearize* the problem by exploiting the properties of binary variables, which guarantee that $z'^{iw}_{cm_a} = z^{w}_{cm_a}(1 - z^{w}_{im_a})$ and $z''^{iw}_{cm_a} = z^{w}_{cm_{a-1}} z^{w}_{im_a}$. For control flow, Equation (14) has the same role. Finally, Equation (15) enforces link capacity.

By setting the parameters of the MILP formulation, an user can apply the NIoTO framework to hierarchical multi-dimensional architectures, making it possible to optimize the deployment of arbitrary next-gen IoT applications as long as they follow the NIoTO model.

V. Performance Evaluation

In this section, the possible benefits achieved when NIoTO is applied are evaluated: shorter response times and lower costs. These benefits are the result of NIoTO's consideration of the application, computing and networking dimensions by assessing the number of micro-service replicas, deploying the micro-services and placing SDN controllers, respectively. Moreover, the potential drawbacks of NIoTO are also analyzed. At first, the simulation environment is described. Then, three different sets of experiments are proposed. In the first one, the MILP solver is run over four topologies of different sizes with different optimization objectives to evaluate the trade-off between the deployment cost and the response time. The second analysis aims at evaluating the computational complexity of NIoTO by assessing the time required to optimize different scenarios, while the third one is devoted to comparing NIoTO's performance in both, response time and cost, with that of state-of-the-art benchmarks. This last analysis is carried out by comparing the response time and costs yielded by NIoTO with those obtained with the benchmarks.

A. Simulation Environment

In order to evaluate the performance of NIoTO framework, four different scenarios based on the example described in Sec. II have been considered. In each scenario, we vary the parameters of the three considered dimensions. In the application dimension, we vary the number of users (and thus, of workflow requests). In the computing dimension, we vary the number of fog nodes where such services can be deployed, as well as the number of gateway nodes connecting the network fabric to the cloud. The number of SDN nodes of the network is varied in the networking dimension. To evaluate the scalability of the proposed solution, values for previous parameters are increased in each scenario, deriving in the simulation set-up shown in Tab. III. The different network topologies leveraged for evaluation were generated using the Erdös-Rényi model for network generation [28], applying the parameters for the simulation set-up as reported by Tab. III. The specific model of each element in the scenario, as well as their CAPEX and OPEX (retrieved from [29]), are detailed in Tab. IV. The specific technical characteristics of these devices, such as their available RAM or CPU clock speed, were retrieved from the official specifications of each device. Thus, to retrieve this information, we refer the reader to the official datasheets and documentations of the computational resources. Regarding the type of wireless connection used by IoT devices to connect with the SDN network, both Wi-Fi and Bluetooth technologies are exploited by Arduino devices, while 6LoWPAN and ZigBee technologies are used by Texas Instruments ones. The access layer makes use of these wireless technologies, while the network core is connected through Gigabit Ethernet links. Link capacities were adjusted according to the capacity of the link's technology (e.g., Gigabit Ethernet links have 1 Gbps capacity, Wi-Fi links have 300 Mbps). To calculate the length of the links, the hospital in [30] is used as a reference for size. Thus, each of these links are, at most, 90 meters long, and their lengths were obtained from a normal distribution of mean $\mu = 45$ and standard deviation $\sigma = 18$. Their transport latencies were calculated based on these lengths. Finally, the number of micro-service replicas that are deployed ranges between 18 and 125, depending on the number of users in the simulation, since more recurrent services are replicated as more users need them to maintain the QoS. The technical details for each of the microservices can be found in Tab. V

Table III: Parameter setting.

Scenario	SDN Nodes	Users	Fog Nodes	Gateways
7-node	7	5	1	1
20-node	20	15	3	2
50-node	50	40	10	5
150-node	150	50	20	10

B. Response Time-Deployment Cost Trade-off

The first analysis we propose aims at evaluating the performance of the proposed framework over two different metrics: i) average response time; and ii) deployment cost. At first, each single objective is independently evaluated. Then, both metrics are compared to find a suitable trade-off representing the best deployment cost-response time compromise, giving the same weight to both metrics.

Fig. 3 shows the outcomes of this analysis in terms of deployment cost, whereas Fig. 4 depicts the results in the workflows' average response time. These two figures present the analysis for three groups of simulations: i) the objective function aims at minimizing the average response time; ii) the objective function is defined as the minimization of the deployment cost; and iii) a multi-objective function where the weight given to both metrics is the same. By inspecting

Table IV: Models and costs considered for each infrastructure element.

Element	Model	CAPEX	OPEX
IoT device	Arduino UNO	20 €	$6.10 \cdot 10^{-16}$ €/cycle
IoT device	Texas Instruments CC2538	4.65 €	$1.59 \cdot 10^{-16}$ €/cycle
Fog node	Pandaboard	215 €	$1.48 \cdot 10^{-16}$ €/cycle
Fog node	edge.network instance	0 €	$3.47 \cdot 10^{-16}$ €/cycle
Cloud node	Amazon Web Services m5.xlarge instance	0 €	$5.56 \cdot 10^{-15}$ €/cycle
Cloud node	Google Cloud Platform E2-Standard-4 instance	0 €	$4.17 \cdot 10^{-15}$ €/cycle
SDN switch	Ruijie Networks RG-S5310-24GT4XS	641 €	$1.15 \cdot 10^{-6}$ €/s
SDN controller	Raspberry Pi 3 A+	22.20 €	$2.12 \cdot 10^{-7}$ €/s
Wi-Fi base station	Pulse Electronics TWR0083	21 €	$3.39 \cdot 10^{-8}$ €/s
Bluetooth base station	Pulse Electronics TWR0083	21 €	$1.06 \cdot 10^{-10}$ €/s
ZigBee base station	DIGI XB3-24Z8UM-J	13.73 €	$2.06 \cdot 10^{-8}$ €/s
6LoWPAN base station	Renesas Electronics ZWIR4532-U	28.80 €	$3.82 \cdot 10^{-7}$ €/s

Table V: Input information for micro-services, as reported in [17]

Micro-service	RAM required	Input size	Output size	Execution cycles
ECG and blood pressure monitor	393 MB	8 Kbps	10 Kbps (ECG)/1 Kbps (blood pressure)	$24.44 \cdot 10^9$ cycles
Compression	136 MB	10 Kbps	2.27 Kbps	$9.95 \cdot 10^9$ cycles
Encryption	79 MB	2.27 Kbps (ECG)/1 Kbps (blood pressure)	2.30 Kbps (ECG)/1.02 Kbps (blood pressure)	$6.18 \cdot 10^9$ cycles

Fig. 3, it is clear that the size of the network highly impacts the cost of the associated deployment. However, there are no big differences in the reported cost when the objective of the optimization is varied, i.e., similar results are obtained for the optimization of single metrics as well as for the joint average response time-deployment cost objective. The difference between the joint objective and cost is negligible, while the average response time objectives yields deployments between 467 and 8610€ more expensive. Nonetheless, if we move our attention to the results on Fig. 4, several considerations emerge. At first, the network size negatively impacts the obtained average response time when the optimization objective is the deployment cost, being up to 16.28 times longer in large scenarios compared with smaller ones. Conversely, a longer response time is experienced in smaller scenarios when response time is the unique metric to optimize. The main reason behind this behavior is that smaller scenarios have fewer resources, specially fog nodes, and thus more services need to be deployed to the cloud. Finally, if both metrics are equally balanced, the difference in the experienced response time is negligible.

As a summary, from the previous evaluation , next remarks are extracted. First, the network size is the parameter that impacts the cost of infrastructure deployment, regardless the type of optimization performed. Second, there is a direct proportional relationship between the network size and the average response time when the objective of the optimization is the deployment cost. However, such relationship is inverse when the metric to optimize is the response time. Finally, if a joint optimization is performed, no clear impact is experienced.

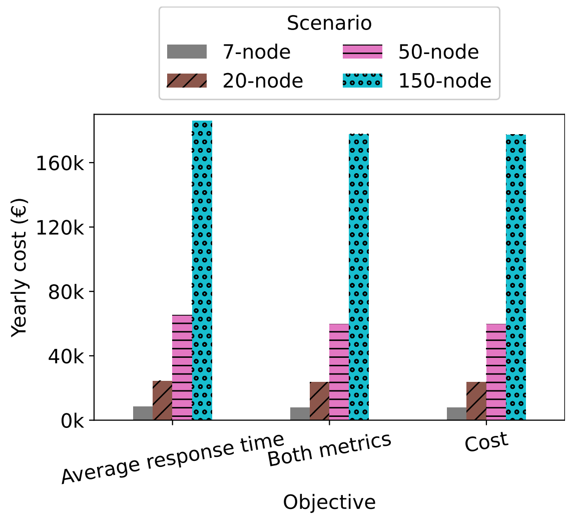


Figure 3: Cost as a function of the optimization objective for the 4 considered scenarios.

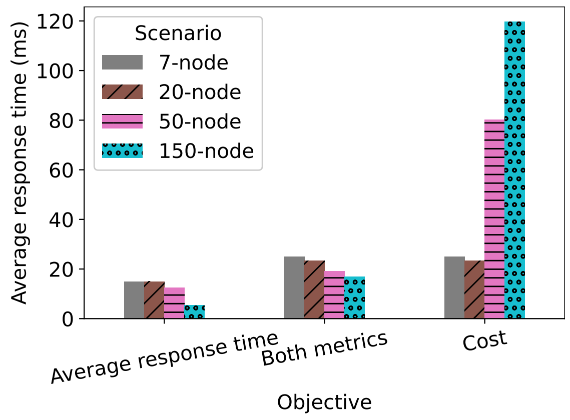


Figure 4: Average response time as a function of the optimization objective for the 4 considered scenarios.

C. Computational Analysis

In the following, an analysis of the computation time required by NIoTO to obtain a solution is performed. NIoTO has been implemented using Python's MIP library [31], solved using Gurobi, and run on a quad-core Intel-based machine at 2 GHz with 16 GB of RAM. Fig. 5 shows the outcomes of the time required to provide an optimal deployment solution as a function of the topology size for the three proposed optimizations.

The first aspect to remark is the exponential increase of the computation time (note the logarithmic y axis) w.r.t. the topology size, for each of the three optimizations. As expected, the joint optimization of response time and cost is the one that requires more time to provide a solution, taking between 2 and 3.5 times more time than the optimization of a single objective. Finally, the metric that in general requires less time to converge is the deployment cost, lasting, on average, 502.55 seconds less than the response time as a single optimization objective. This analysis clearly shows that the MILP version of NIoTO, especially in big scenarios, is mainly suitable for design time optimization. Nonetheless, the development of a faster version of NIoTO, able to optimize and adapt the deployment during execution time, is one of the key future works.

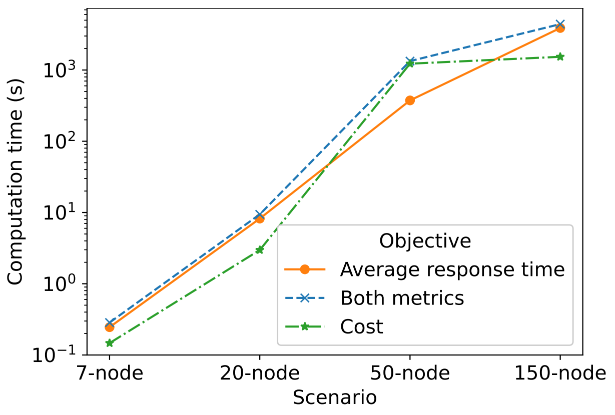


Figure 5: Computation time required to provide an optimal deployment solution as a function of the topology size.

D. Benchmark comparison

The objective of the final analysis is to compare the results obtained by NIoTO with other similar, state-of-the-art techniques. Concretely, two benchmarks are considered for comparison: ModuleMapping [32], as a benchmark aimed at response time and resource usage optimization; and FogPart [33], which focuses on optimizing the financial costs of the deployment. To the best of our knowledge, no benchmarks that jointly optimize response time and cost have been found: ModuleMapping focuses exclusively on response time and ignores cost, while FogPart focuses on cost and ignores response time optimization. Moreover, it is important to note that, out of these three techniques, only NIoTO has a holistic view of the hierarchical multi-dimensional architecture. Nonetheless, the input information was adapted to each of the benchmarks in order to maintain a fair comparison (e.g., FogPart considers the cost of sending data from a computing device to the cloud as a single, aggregated cost, rather than a multitude of CAPEX and OPEX items [33]; and thus, it was provided as an aggregate of the CAPEX and OPEX of the network equipment used to send the data).

The results of the comparison are depicted in Fig. 6 for average response time and in Fig. 7 for deployment cost, while they are detailed in Table VI for response time and Table VII for cost. Moreover, the comparison includes two categories for NIoTO (one for the appropriate objective and one for the joint approach), and a single one for the benchmark that optimizes the appropriate objective (i.e., ModuleMapping for response time and FogPart for cost). Starting the comparison with response times (Fig. 6, Table VI), we find NIoTO with the average response time objective as the best technique for all the analyzed scenarios, which we consider to be optimal. Its performance is closely followed by NIoTO optimizing both metrics, with an average optimality gap of approximately 9.16 ms. ModuleMapping is the last one, with an average optimality gap of 40.22 ms. The response time of ModuleMapping is constant through all the topology sizes due to the fact it only considers two of the three dimensions: application and computing. ModuleMapping uses the computational resources of a device, as well as the resources required by the micro-service, as the main metric to select where to deploy a micro-service to [32]. Since the cloud is consistently the most powerful and resourceful computing device, ModuleMapping will deploy as many micro-services as possible there. Due to the usage of fog nodes to reduce the experienced latency, NIoTO achieves an average speed-up of 5.11 w.r.t. ModuleMapping.

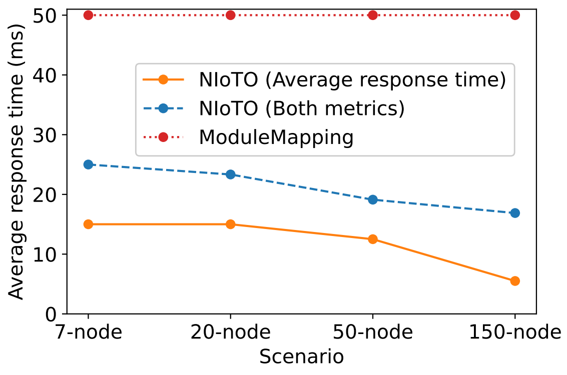


Figure 6: Average response time comparison of NIoTO and ModuleMapping.

Table VI: Average response times yielded by NIoTO and ModuleMapping.

Scenario	NIoTO (average response time)	NIoTO (both metrics)	ModuleMapping
7-node	15.005 ms	25.004 ms	50.004 ms
20-node	15.005 ms	23.338 ms	50.004 ms
50-node	12.506 ms	19.120 ms	50.004 ms
150-node	5.508 ms	16.882 ms	50.004 ms

Continuing with cost, as depicted in Fig. 7 and detailed in Table VII, NIoTO with the cost objective is the optimal solution for the analyzed scenarios. Once again, we find the next best solution to be NIoTO optimizing both metrics, with an optimality gap of approximately 50€ (0.08%). FogPart is the third best technique, with an optimality gap of 3954€ (6.22%). It is important to note that the gap between FogPart and NIoTO increases with topology size, and thus, using FogPart in larger topologies may lead to larger optimality gaps. The difference in costs responds to FogPart's partial view on the scenario, not considering the computing dimension. To choose the node to deploy a micro-service to, FogPart compares the cost of communicating the previous micro-service in the workflow with the cloud, and the cost of communicating with a fog node instead, choosing the most cost-effective communication [33]. Thus, the costs of the computing dimension, such as the CAPEX of the computing devices used or their OPEX per cycle, may not be optimal. This is precisely the difference we find between NIoTO and FogPart. In a general conclusion, we find NIoTO, with the according objective, as the optimal solution in terms of response time and cost. Moreover, the version of NIoTO that optimizes both metrics at the same time is able to perform better at both dimensions than state-of-the-art techniques aimed at their optimization.

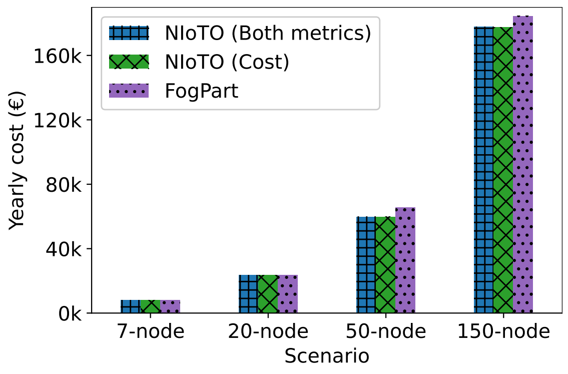


Figure 7: Deployment cost comparison of NIoTO and FogPart.

Table VII: Yearly costs of deploying the scenario using NIoTO and FogPart.

Scenario	NIoTO (cost)	NIoTO (both metrics)	FogPart
7-node	8,070€	8,070€	8,233€
20-node	23,672€	23,673€	23,673€
50-node	59,801€	59,801€	65,644€
150-node	177,508€	177,772€	184,502€

VI. Conclusions and Future Work

As the number of IoT devices grows every year, the demand for QoS-strict IoT applications, hardly suitable for a cloud-based deployment, does as well. A hierarchical multi-dimensional architecture is an enabler for this kind of applications, but optimizing the QoS requires the consideration of the interplay between the computing, networking and application dimensions. In this work, we present NIoTO, a framework to optimally deploy next-gen IoT applications in a hierarchical multi-dimensional architecture, considering all the three aforementioned dimensions. NIoTO is able to optimally assess the number of micro-service replicas in the application dimension, their deployment in the computing dimension, and the placement of the SDN controller in the networking dimension, along with optimizing the routing of the generated traffic flows. The joint consideration of all three dimensions, and the optimization of the response time and cost, including the assessment of the optimal trade-off between them, allows it to optimize each of the metrics further than related, state-of-the-art frameworks. In the future, we expect to support run-time, dynamic optimizations of the deployment through the development of a faster solver for NIoTO, enabling it to adapt the deployment to environmental changes. Moreover, we expect to evaluate NIoTO's performance over real or emulated network test-beds.

REFERENCES

[1] Cisco. (2020, March) Cisco Annual Internet Report (2018–2023). (visited on Jan. 29, 2021). [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.pdf

[2] 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.

[3] B. Farahani, F. Firouzi, V. Chang, M. Badaroglu, N. Constant, and K. Mankodiya, "Towards fog-driven iot ehealth: Promises and challenges of iot in medicine and healthcare," *Future Generation Computer Systems*, vol. 78, pp. 659–676, 2018.

[4] 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.

[5] S. Forti and A. Brogi, "Continuous reasoning for managing next-gen distributed applications," *arXiv preprint arXiv:2009.10245*, 2020.

[6] L. Nkenyereye, J. Y. Hwang, Q.-V. Pham, and J. S. Song, "Virtual iot service slice functions for multi-access edge computing platform," *IEEE Internet of Things Journal*, 2021.

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

[8] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: State of the art and research challenges," *Computer*, vol. 40, no. 11, pp. 38–45, 2007.

[9] D. Kutscher, T. Karkkainen, and J. Ott, "Directions for Computing in the Network," Internet Engineering Task Force, Internet-Draft draft-kutscher-coinrg-dir-02, Jul. 2020, (visited on Jan. 29, 2021). [Online]. Available: https://datatracker.ietf.org/doc/html/draft-kutscher-coinrg-dir-02

[10] J. L. Herrera, J. Galán-Jiménez, J. Berrocal, and J. M. Murillo, "Optimizing the response time in sdn-fog environments for time-strict iot applications," *IEEE Internet of Things Journal*, 2021.

[11] K. Indrasiri. Microservices in practice - key architectural concepts of an MSA. (visited on Jan. 14, 2021). [Online]. Available: https://wso2.com/whitepapers/microservices-in-practice-key-architectural-concepts-of-an-msa/

[12] H. Haas, D. Orchard, F. McCabe, C. Ferris, E. Newcomer, D. Booth, and M. Champion, "Web services architecture," W3C, W3C Note, Feb. 2004, https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.

[13] GRPC, "Documentation — gRPC," 2021. [Online]. Available: https://grpc.io/docs/

[14] R. T. Fielding, *Architectural styles and the design of network-based software architectures*. University of California, Irvine, 2000.

[15] R. Priya, K. H. Prabha, R. Hemalatha, P. Vanmathi, and M. Ilakiya, "A secured remote health monitoring system based on iot," *Annals of the Romanian Society for Cell Biology*, pp. 17 759–17 765, 2021.

[16] T. N. Gia, M. Jiang, A.-M. Rahmani, T. Westerlund, P. Liljeberg, and H. Tenhunen, "Fog computing in healthcare internet of things: A case study on ecg feature extraction," in *2015 IEEE international conference on computer and information technology; ubiquitous computing and communications; dependable, autonomic and secure computing; pervasive intelligence and computing*. IEEE, 2015, pp. 356–363.

[17] A. Limaye and T. Adegbija, "A workload characterization for the internet of medical things (iomt)," in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2017, pp. 302–307.

[18] M. Huang, W. Liu, T. Wang, H. Song, X. Li, and A. Liu, "A queuing delay utilization scheme for on-path service aggregation in services-oriented computing networks," *IEEE Access*, vol. 7, pp. 23 816–23 833, 2019.

[19] Q. Duan, S. Wang, and N. Ansari, "Convergence of networking and cloud/edge computing: Status, challenges, and opportunities," *IEEE Network*, vol. 34, no. 6, pp. 148–155, 2020.

[20] 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.

[21] P. Wang, H. Xu, L. Huang, C. Qian, S. Wang, and Y. Sun, "Minimizing controller response time through flow redirecting in sdns," *IEEE/ACM Transactions on Networking*, vol. 26, no. 1, pp. 562–575, 2018.

[22] ETSI. Open Source NFV Management and Orchestration (MANO). (visited on Jan. 14, 2021). [Online]. Available: https://www.etsi.org/technologies/open-source-mano

[23] M. M. Badawy, Z. H. Ali, and H. A. Ali, "Qos provisioning framework for service-oriented internet of things (iot)," *Cluster Computing*, pp. 1–17, 2019.

[24] J. Berrocal, J. Garcia-Alonso, C. Vicente-Chicote, J. Hernández, T. Mikkonen, C. Canal, and J. M. Murillo, "Early analysis of resource consumption patterns in mobile applications," *Pervasive and Mobile Computing*, vol. 35, pp. 32 – 50, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1574119216300797

[25] M. Ronchetti, G. Succi, W. Pedrycz, and B. Russo, "Early estimation of software size in object-oriented environments a case study in a cmm level 3 software firm," *Information Sciences*, vol. 176, no. 5, pp. 475–489, 2006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025505000083

[26] M. Bichier and K.-J. Lin, "Service-oriented computing," *Computer*, vol. 39, no. 3, pp. 99–101, 2006.

[27] Open Networking Foundation, "OpenFlow Switch Specification 1.3.0," pp. 1–3205, 2013. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf

[28] P. Erdös and A. Rényi, "On random graphs i," *Publ. math. debrecen*, vol. 6, no. 290-297, p. 18, 1959.

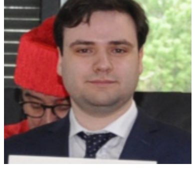
[29] J. Herrera, J. Galán-Jiménez, J. García-Alonso, J. Berrocal, and J. Murillo. (2021) Joint optimization of response time and deployment cost in next-gen iot applications. (visited on Oct. 14, 2021). [Online]. Available: https://tinyurl.com/nioto-capex-opex

[30] G. Maps, "Hospital provincial nuestra sra. de la montaña," 2021. [Online]. Available: https://goo.gl/maps/NqsPvQbw7JhaSW8k6

[31] T. A. Toffolo and H. G. Santos, "Python-MIP," 2021. [Online]. Available: https://www.python-mip.com/

[32] 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.

[33] Z. Á. Mann, A. Metzger, J. Prade, and R. Seidl, "Optimized application deployment in the fog," in *International Conference on Service-Oriented Computing*. Springer, 2019, pp. 283–298.



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

Jaime Galán-Jiménez received the 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 Software-Defined Networks, 5G network planning and design, and mobile ad-hoc networks.

Jose Garcia-Alonso (IEEE Memeber) is an Associate Professor at the University of Extremadura. His research interests include software engineering, mobile computing, pervasive computing, eHealth, gerontechnology.

Javier Berrocal (IEEE Memeber) is a cofounder of Gloin. His main research interests are software architectures, mobile computing and edge and fog computing. Berrocal has a PhD 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.

Critical thinking is the key to success.

Professor Layton
Akihiro Hino

Chapter 6

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

Authors : Juan Luis Herrera, Jaime Galán-Jiménez, Luca Foschini, Paolo Bellavista, Javier Berrocal, Juan Manuel Murillo

Publication type : Journal article

Journal : IEEE Internet of Things Journal

DOI : `https://doi.org/10.1109/JIOT.2022.3143948`

JCR : Q1 (Computer Science, Information Systems). IF: 10.238.

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: [jlherrerag, jaime, juanmamu, jberolm]@unex.es).

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

Digital Object Identifier 00.000/JIOT.0000.0000000

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.

Considered 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 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.

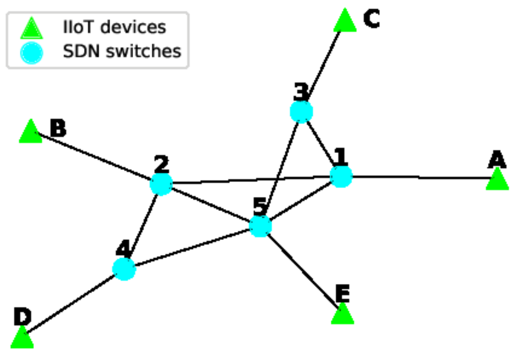


Figure 1: Topology for the example model: IIoT factory automation.

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 vertices[1] 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: $V = H \cup S; H \cap S = \emptyset$: H. S contains all SDN switches, whereas H contains all the hosts.

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

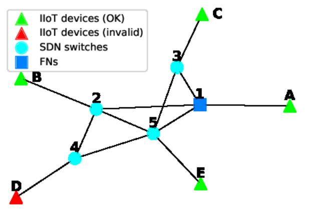
Table I: List of formulation notations.

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^h_{ij}	Boolean to determine if traffic generated by host h is routed through link l_{ij}

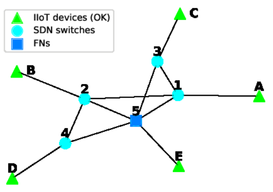
Starting with S, the objective of the FNPP is to replace a given number of SDN switches with FNs. We call this number of FNs θ. Furthermore, each SDN switch $s \in S$ is a potential location for a FN. We assume that all FNs to be set in a network have a capacity α for processing traffic. Moreover, switches 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 $f^h_{ij} \forall h \in H; l_{ij} \in L$ 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, $L(h) = (\sum_{l_{ij} \in L} f^h_{ij}(\beta_{ij} + \beta_S)) - \beta_S$. 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

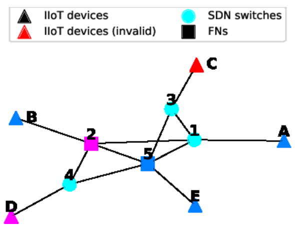


(a) FN placement in 1

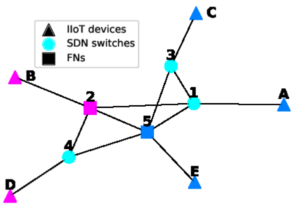


(b) FN placement in 5

Figure 2: First scenario.



(a) First assignment option.



(b) Second assignment option.

Figure 3: Second 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):

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

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

subject to:

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

$$\forall l_{ij}\in L: \sum_{h\in H} f_{ij}^h \phi_h \le C_{ij} \tag{4}$$

$$\sum_{s\in S} X_s \le \theta \tag{5}$$

$$\forall s\in S: \sum_{h\in H} \phi_h Y_{hs} \le \alpha X_s \tag{6}$$

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

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

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 constrains, 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). 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 := $S$[first: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

```
Input: $G$: topology graph;
$I$: set of initial centroids;
Output: $F$: placement for FNs;
begin
    $\mathcal{C}$ := dictionary();
    for $i \in I$ do
        $\mathcal{C}_i := \emptyset$;
    end
    for $v \in G.V$ do
        minDist := $\infty$;
        for $i \in I$ do
            centDist := shortestPathDistance($v$, $i$, $G$);
            if centDist < minDist then
                minDist := centDist;
                minCentroid := $i$;
            end
        end
        $\mathcal{C}_{minCentroid} := \mathcal{C}_{minCentroid} \cup v$;
    end
    $F := \emptyset$;
    changes := 0;
    for $i \in I$ do
        minDist := $\infty$;
        for $v_1 \in \mathcal{C}_i$ do
            totalDist := 0;
            for $v_2 \in \mathcal{C}_i - \{v_1\}$ do
                totalDist := totalDist +
                  shortestPathDistance($v_1$, $v_2$, $G$);
            end
            if totalDist < minDist then
                totalDist := minDist;
                newCentroid := $v_1$;
            end
        end
        if newCentroid $\neq i$ then
            changes := changes + 1;
        end
        $F := F \cup$ {newCentroid};
    end
    if changes > 0 then
        $F$ := modifiedKMedoids($G$, $F$);
    end
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

```
Input: $G$: topology graph;
$F$: placement for FNs;
$\alpha$: FN capacity;
$\phi$: vector of traffic flows. $\phi_h$ is the size of the traffic
  flow of host $h$;
Output: $A$: assignments, in dictionary form. $A_h$ is the
  FN assigned to host $h$;
begin
    $A$ := dictionary();
    remainingCap = dictionary(keys=$F$, values=$\alpha$);
    $sortedH$ := ascendingSort($H$, criteria=$\phi$);
    for $h \in sortedH$ do
        FNCandidates := ascendingSort($F$,
          criteria=shortestPathDistance($h$, $G$));
        for $f \in$ FNCandidates do
            if $\phi_h \leq remainingCap_f$ then
                $A_h := f$;
                $remainingCap_f :=$
                  $remainingCap_f - \phi_h$;
                break;
            end
        end
    end
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

```
Input: $G$: topology graph;
$\phi$: vector of traffic flows. $\phi_h$ is the size of the traffic
    flow of host $h$;
$A$: assignments, in dictionary form. $A_h$ is the FN
    assigned to host $h$;
Output: $R$: routes, in dictionary form. $R_h$ is the path
    from $h$ to its assigned FN;
begin
    $sortedH$ := descendingSort($H$, criteria=$\phi$);
    $R$ := dictionary();
    for $h \in sortedH$ do
        finalPath := 0;
        $G'$ := copy($G$);
        while $finalPath \leq 0$ do
            route := shortestPath($h$, $A_h$, $G'$);
            finalPath:= 1;
            for $l_{ij} \in route$ do
                cap = capacity($l_{ij}$, $G'$);
                if $cap < \phi_h$ then
                    finalPath := 0;
                    removeLink($l_{ij}$, $G'$);
                end
            end
        end
        $R_h$ := route;
    end
end
```

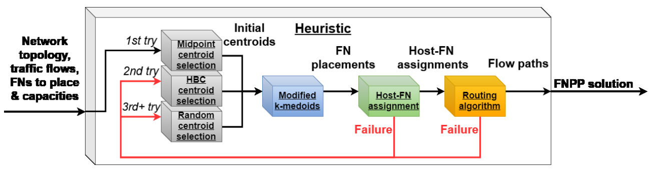


Figure 4: Heuristic behavior diagram.

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, 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 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.

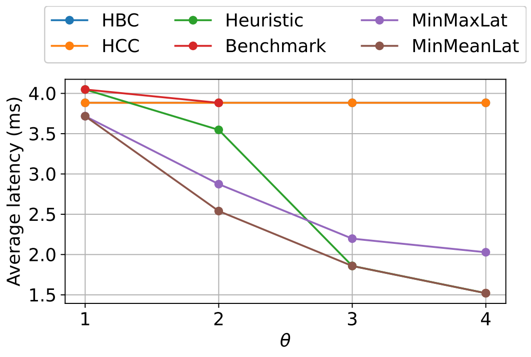


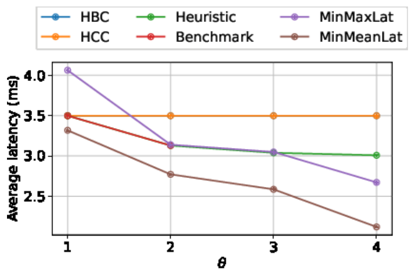
Figure 5: Average latencies in Abilene.

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.

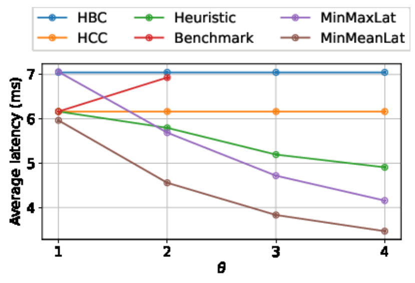
Fig. 6 shows the average latencies in larger topologies: GEANT (Fig. 6a), Germany-50 (Fig. 6b) and Brain (Fig. 6c). Performance-wise, we can see in general that HBC and HCC yield the highest latencies, followed by MinMaxLat, which highly varies in this performance metric, raising from the 4th place to 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.

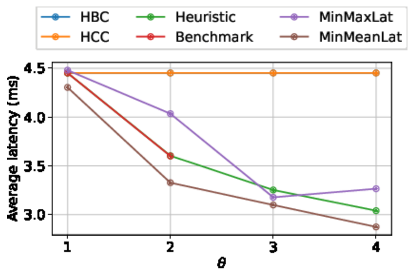
The execution time analysis is depicted in Fig. 8. This analysis makes use of the largest θ value to maintain fairness and compare all solutions in a worst-case scenario for execution time, since their execution time is directly related to the value of θ. This analysis allows for a study of the scalability 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



(a) GEANT (22 switches, 36 links).



(b) Germany-50 (50 switches, 88 links).



(c) Brain (161 switches, 166 links).

Figure 6: Average latencies in other topologies

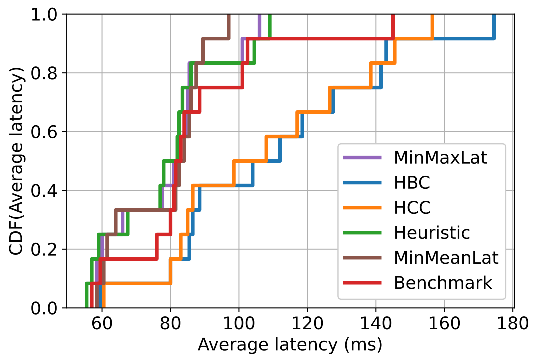


Figure 7: CDF of emulated latency in Abilene, $\theta = 4$

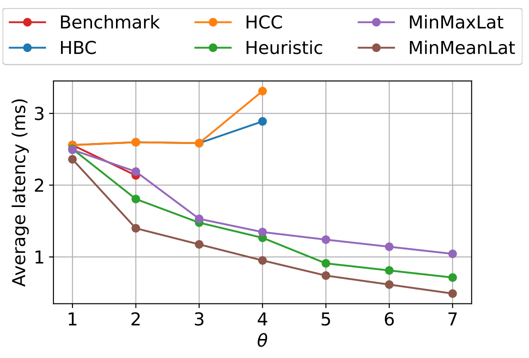


Figure 9: Average latencies in LCL-scaled.

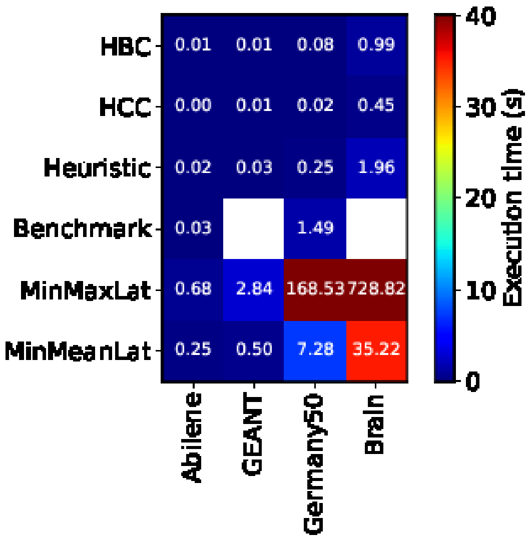


Figure 8: Optimization times for all topologies, $\theta = 4$.

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

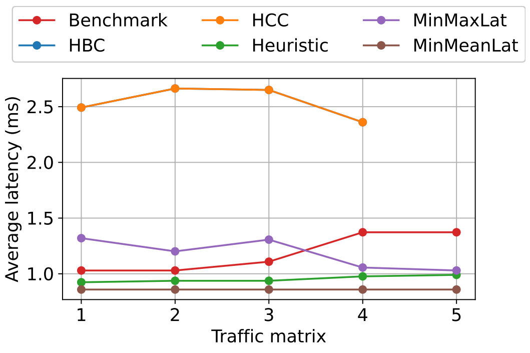


Figure 10: Average latencies in HCL, $\theta = 4$.

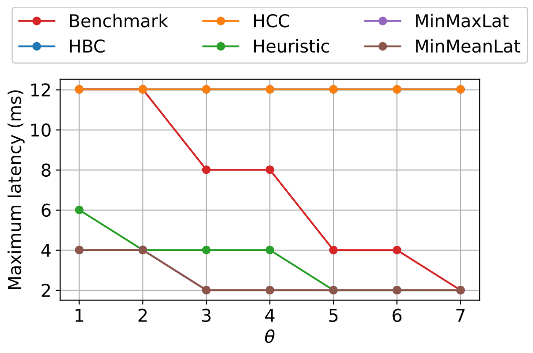


Figure 11: Maximum latencies in HCL.

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

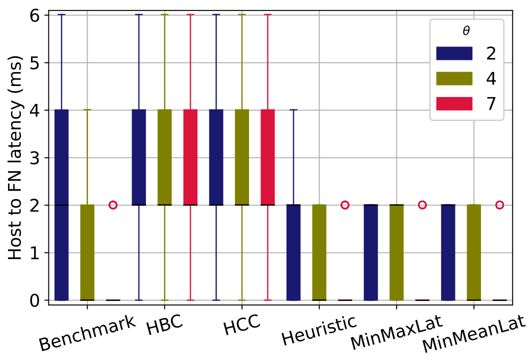


Figure 12: Latency box plot for LCL-unscaled.

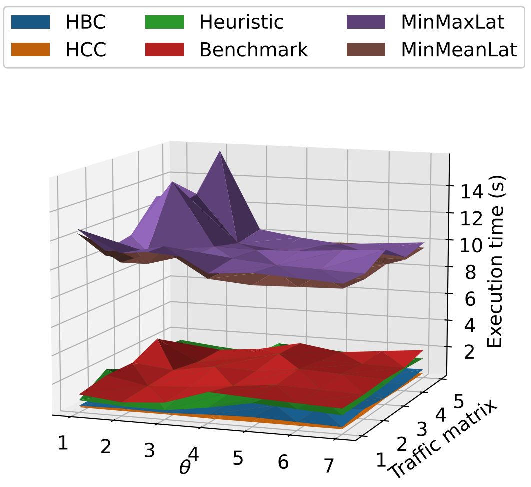


Figure 13: Execution times for LCL-unscaled.

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 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

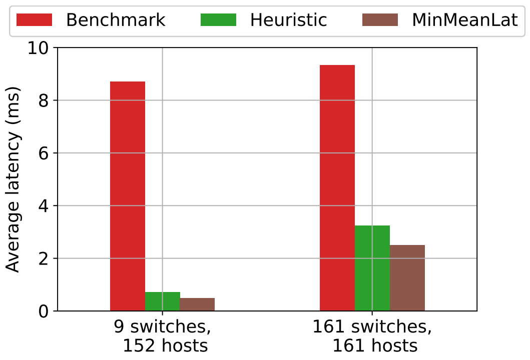


Figure 14: Average latencies in different topology sizes, $\theta = 7$.

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. Eyers, "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. Kassar, and T. Aguili, "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. Murillo, 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. Linderoth 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.

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, where he is now an Assistant Professor. His main research interests are SDNs, 5G network planning and design, and mobile ad hoc networks.

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.

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.

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.

One is All, All is One.

Fullmetal Alchemist

Hiromu Arakawa

Chapter 7

Optimal Deployment of Fog Nodes, Microservices and SDN Controllers in Time-Sensitive IoT Scenarios

Authors	: Juan Luis Herrera, Jaime Galán-Jiménez, Luca Foschini, Paolo Bellavista, José García-Alonso, Javier Berrocal, Juan Manuel Murillo
Publication type	: Conference paper
Conference	: IEEE Global Communications Conference (GLOBECOM)
DOI	: `https://doi.org/10.1109/GLOBECOM46510.2021.9685995`
Rating	: GGS Class 2

Optimal Deployment of Fog Nodes, Microservices and SDN Controllers in Time-Sensitive IoT Scenarios

Juan Luis Herrera*, Jaime Galán-Jiménez*, Paolo Bellavista†, Luca Foschini†, Jose Garcia-Alonso*, Juan M. Murillo* and Javier Berrocal*
*Department of Computer Systems and Telematics Engineering, University of Extremadura, Spain.
†Dipartamento di Informatica-Scienza e Ingegneria, University of Bologna, Italy.
[jlherrerag, jaime, jgaralo, juanmamu, jberolm]@unex.es, [paolo.bellavista, luca.foschini]@unibo.it

***Abstract*—The application of Internet of Things (IoT)-based solutions to intensive domains has enabled the automation of real-world processes. The critical nature of these domains requires for very high Quality of Service (QoS) to work properly. These applications often use computing paradigms such as fog computing and software architectures such as the Microservices Architecture (MSA). Moreover, the need for transparent service discovery in MSAs, combined with the need for network scalability and flexibility, motivates the use of Software-Defined Networking (SDN) in these infrastructures. However, optimizing QoS in these scenarios implies an optimal deployment of microservices, fog nodes, and SDN controllers. Moreover, the deployment of each of the different elements affects the optimality of the others, which calls for a joint solution. In this paper, we motivate the joining of these three optimization problems into a single effort and we present Umizatou, a holistic deployment optimization solution that makes use of Mixed Integer Linear Programming. Finally, we evaluate Umizatou over a healthcare case study, showing its scalability in topologies of different sizes.**

I. Introduction

The Internet of Things (IoT) paradigm has brought the potential of real-world interaction to computer applications. This potential has drawn attention to the IoT from intensive domains aside from general-purpose one, such as the Industrial Internet of Things (IIoT) [1] or the Internet of Medical Things (IoMT) [2]. Due to the critical nature of the processes in intensive domains, they often require a high Quality of Service (QoS), such as extremely short response times [1], [2]. This QoS is motivated by the QoS of the application's execution, as well as by the QoS of its communications. Thus, the three dimensions that affect QoS must be considered: application, computing and networking.

In the application dimension, current IoT applications are often based on the Service-Oriented Computing paradigm [3]. The Microservices Architecture (MSA) design pattern allows IoT applications to be divided into loosely coupled, collaborating *microservices* [4]. Nonetheless, microservices are independent from each other, and can be leveraged both separately or combined [4]. For instance, microservices can be combined by having the output of a microservice be the input of another one, in a pipeline manner. When the application is deployed, each of the microservices can be executed by a different device. The decision of where microservice is executed should be motivated by the obtained QoS.

Focusing on response time, the computational complexity of the microservice and the computational power of the device are the main factors. Since microservices require a certain amount of RAM to run, and devices have a limited amount of RAM, the optimal placement of microservices in devices can become complex in large infrastructures. The problem that involves optimal deployment of microservices in an infrastructure is known in literature as the Decentralized Computation Distribution Problem (DCDP) [5].

In the computing dimension, the most popular computing paradigm for IoT is cloud computing [6]. In this paradigm, a server far away from the IoT devices provides the application's services to them. However, the distance between IoT devices and cloud servers is so large that guaranteeing a suitable QoS in intensive domains may be very complicated [1], [2]. This problem has motivated the rise of new computing paradigms, such as fog computing, that bring some of the computational resources of the cloud closer to IoT devices [7]. This closer placement allows IoT devices to obtain enhanced QoS (e.g., shorter latencies).

However, there is a key factor to be considered: the placement of fog nodes has an effect on QoS [8]. Due to the fact that microservice communication occurs between the devices running them, the placement of said devices affects the QoS of their communication [8]. Therefore, they should be placed optimally in the infrastructure. The formal problem of optimally placing fog nodes is known as the Fog Node Placement Problem (FNPP) [8].

Finally, in the networking dimension, the communications of the underlying infrastructure must be supported. In a fog infrastructure, IoT devices must know where microservices are deployed (*service discovery*) to communicate with the corresponding fog nodes [9]. Since this is unrelated to the application's logic, it would be desirable to perform service discovery transparently, separating the concerns of application and deployment [9]. Furthermore, the infrastructures for intensive-domain IoT applications require the network to be scalable and flexible while maintaining the strict QoS required [1], [8]. All these requirements can be met by making use of the Software-Defined Networking (SDN) paradigm [9], [10]

SDN allows for network programmability by decoupling the control and data planes, embodying the control plane in the figure of the SDN controller. However, since SDN switches are purely data plane equipment, they require to communicate with the SDN controller to operate. Due to these communication requirements, the switch-controller communications QoS are key. Considering response time, latency is a crucial QoS metric. Every time a flow setup is required, the overall network latency is affected by the latency between SDN switches and controllers [11]. Thus, the placement of SDN controllers should be optimized. This optimization problem is named the SDN Controller Placement Problem (CPP) [11].

Each of these optimization efforts are aimed at the same objective. Moreover, due to the coupling between all three dimensions, they are not independent, as each decision taken in one effort affects the rest of them. For instance, if the latency of communicating with a device is higher than the time saved by executing a microservice in said device, microservices should not be deployed in it. Thus, latency needs to be considered in the DCDP [5], which depends on the FNPP and the CPP. To optimally place fog nodes, it is important to know which other nodes of the infrastructure it must communicate with [8]. Placing two fog nodes that must communicate far from each other will have a very negative impact on latency. Placing them closely, but far from the IoT devices that communicate with them, will have a similar impact as well. To trade them off, the FNPP must consider whether or not two nodes interact, and the latency between different potential placements. These factors depend on the DCDP and the CPP, respectively. In the case of the CPP, controller placement is affected by routing [11]. For instance, it is desirable to have lower controller latency in switches that are heavily used, as more flows will obtain a better QoS [11]. The steering of these flows depends on the communications between microservices (DCDP) and the devices that communicate (FNPP). Therefore, all three problems are linked together.

Therefore, and due to the mutual influences between all three dimensions, there is a need for a complete deployment solution. Such solution needs to solve the DCDP, the FNPP and the CPP, taking into account how each of them influence the overall QoS. In this work, we present Umizatou (named after the legendary creature Umizatou), a framework that supports deployment optimization in all three dimensions to minimize response time. To the best of our knowledge, no other works in literature address a combined deployment problem that considers controller placement, fog node placement and microservice placement.

The main contributions of this work are:

- The formalization of a problem that combines DCDP, FNPP and CPP, aimed at minimizing response time.
- The formulation of a response time optimization solution using Mixed Integer Linear Programming (MILP).
- An evaluation of Umizatou over an IoMT case study.

The remainder of this paper is structured as follows. Sec. II explains the system model followed in Umizatou. Sec. III includes the formulation used by Umizatou to optimize response time. Sec. IV presents the evaluation over an IoMT case study. Finally, Sec. V concludes the paper and highlights future challenges.

II. System model

To better understand the model used by Umizatou, the problem is explained using an IoMT case study. In this case study, extracted from [12], an IoMT application, along with the fog infrastructure, is to be deployed on a hospital. The application allows for two functionalities: electrocardiogram (ECG) anomaly analysis and blood pressure (BP) analysis. Similar applications for cardiovascular disease detection have also been proposed in the field of edge intelligence for the IoMT domain [13]. Since the application follows an MSA, it is divided into four microservices: an ECG monitor, a BP monitor, an encryption microservice and a data compression microservice. When ECG analysis is requested, the IoMT device gathers the raw ECG information and sends it to the ECG monitor, which outputs a commented ECG. Since the size of a commented ECG is large [12], it is compressed after that. Finally, the compressed, commented ECG is encrypted to maintain confidentiality before being stored. BP analysis is similar, except for the fact that the output of the BP monitor is not large enough to need compression. This IoMT application is depicted in Fig. 1.

The application is to be deployed in an SDN network that is yet to have SDN controllers or fog nodes placed. Therefore, there is a need to jointly solve the FNPP, the CPP and the DCDP for this new application. The kind of fog node that will be used is the one detailed in [14], the same kind as the original model of the FNPP [8]. To tell it apart from generic fog nodes, it will be denoted as *FN*. The main feature from these FNs is that they integrate both an SDN switch and a computing device, enabling them to have both SDN and computing capabilities [14]. Moreover, it is possible to replace a given SDN switch by a FN without loss of functionalities. As for SDN controllers, the classic SDN control model is used, in which controllers are co-located with SDN switches [11].

Application-wise, two possible workflows are considered: the workflow for ECG analysis, and the workflow for BP analysis. Each of these workflows can be instantiated as many times as it is requested. In the case depicted in Fig. 1, two IoMT devices request ECG analysis, while the other two request BP analysis. Each workflow instance, or request, implies that each microservice in the workflow needs to be executed once, and thus, potentially deployed once. If the same microservice from two different workflow requests is executed in the same FN, the microservice is deployed once and executed twice. Nonetheless, if each workflow request executes the microservice in two FNs, the microservice is deployed twice, and executed once per deployment. With said application model in mind, the objective of the DCDP is to find the optimal FNs to execute each microservice.

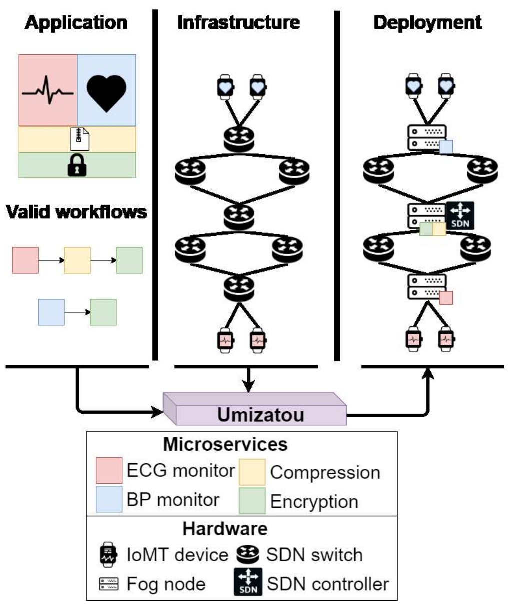


Figure 1: Example operation of Umizatou.

If Umizatou was to be executed to deploy 3 FNs, a single SDN controller and the example IoMT application, the result would be similar to the deployment depicted in Fig. 1. Starting with the infrastructure, FNs are deployed on the top and bottom the network for heavy processing, while the third one is placed in the middle for more lightweight tasks. Such placement is related to the type of communications: raw ECG and BP data is heavier, and thus is placed the closest to the devices. On the other hand, lighter microservices such as compression or encryption are placed in the middle FN. This placement is also due to the fact that they are shared among all workflow requests in the infrastructure, and thus, most traffic flows reach the middle switch. Thus, the SDN controller is placed in it, which boosts the network QoS in this zone.

It is important to note that all these insights for deployment exist because of the joint approach. To place FNs for lighter and heavier microservices, as well as co-located with SDN controllers, it is key to know how heavier and lighter microservices are deployed and where SDN controllers are. Similarly, deploying workflow-specific microservices closer to their requesting IoMT devices and shared microservices closer to SDN controllers implies knowing where FNs and SDN controllers are deployed. Finally, deploying controllers co-located with FNs that execute shared microservices requires knowledge about FN placement and microservice deployment. Hence, Umizatou's joint approach to the FNPP, DCDP and CPP is crucial for this solution.

III. Problem formulation

In order to solve the DCDP, the FNPP and the CPP jointly, we present a MILP formulation. This formulation serves two purposes. Firstly, it formalizes the problem mathematically, thus serving as an abstraction of the problem. And secondly, it allows the problem to be solved optimally during design-time through the use of automatic MILP solvers.

To begin, let $G = \{V, L\}$ be a directed graph that represents the infrastructure. V represents the vertices of the infrastructure, and L represents the links. A link $l_{ij} \in L; i \neq j; i, j \in V$ links together vertices i and j. Each link l_{ij} has a propagation latency δ_{ij} and a maximum capacity θ_{ij}.

We define a subset $H \subset V$, which contains the vertices that are hosts. We also define $S \subset V$ as the subset of vertices that are switches, so that $H \cap S = \emptyset; H \cup S = V$. Similar to links, each switch $s \in S$ has a processing latency δ_s. For simplicity's sake, we define the binary function $SW(v) \forall v \in V$, that is 1 if $v \in S$ and 0 otherwise.

With respect to FNs, let MAX_{FN} be the maximum number of FNs that can be placed in the network. As all FNs are equal, let r be the total RAM memory of a FN. While the CPU clock speed and the number of cycles of a microservice are relevant to the execution time, and thus considered in Sec. IV, they do not need to be considered in the formulation, as we assume that all FNs are equal. Thus, execution time of a microservice is constant, as it will take the same amount of time to run in any given FN. Considering SDN controllers, let MAX_C be the maximum number of controllers to be placed in the network. Moreover, the size of the control packets used to communicate SDN switches and controllers also need to be considered. Thus, let σ be the size of said control packets. In this work, the size is taken from the OpenFlow specification [15].

In the IoT application model considered, logic is executed in workflows. Thus, let M be the set of microservices the application is divided into, and let W be the set of workflows requested in the infrastructure. Each microservice $m \in M$ requires a certain amount of RAM r_m to be executed. Moreover, its input data has a certain size I_m, and its output data has a size O_m. Each workflow $w \in W$ is an ordered set of microservice instances $w = \{c_1, c_2, ..., c_{|w|}\} \subseteq M$, so that each element is a defined microservice: $c_i \in w \Rightarrow c_i \in M$. Furthermore, let $WR(w, h) \forall w \in W, h \in H$ be a function that is equal to 1 if the workflow w is requested by the host h and 0 otherwise.

Next, we define the decision variables for the problem. First, FNs need to be placed in SDN switches. Thus, let $n_s \forall s \in S$ be a binary variable that will be 1 if a FN is placed in SDN switch s. Moreover, each microservice in a workflow must be executed in a FN. Thus, let $z^w_{sc_a} \forall w \in W, s \in S, a \in [1, |w|]$ be a binary variable that will be 1 if the microservice c_a of workflow w is executed in the FN placed in SDN switch s. The traffic flows generated by executing workflows in a distributed manner must also be routed. Therefore, let $f^{vwc_a}_{ij} \forall l_{ij} \in L, v \in V, w \in w, a \in [1, |w|]$ be a binary variable that is 1 if the traffic generated by vertex v as a consequence of requesting microservice c_a of workflow w is routed through the link l_ij. Furthermore, responses must be routed in a similar manner: let $f'^{vw}_{ij} \forall l_{ij} \in L, v \in V, w \in w$ be a binary variable that will be 1 if the traffic generated by vertex v as a response for workflow w is routed through the link l_{ij}. Similarly, for controllers, let x_s be a binary variable that will be 1 if an SDN controller is placed in SDN switch s. In the case of SDN switches, each switch is assigned to a single controller that it communicates with. Thus, let $y_{ss'}$ be a binary variable that will be 1 if SDN switch s communicates with the controller placed in SDN switch s'. Communication flows must also be routed, and therefore, let $cf_{ij}^{s} \forall l_{ij} \in L, s \in S$ be a binary variable that will be 1 if the control flow of SDN switch s is routed through the link l_{ij}.

Then, the problem can be formalized as follows:

$$\begin{aligned}\min \frac{1}{|W|} \sum_{w \in W} \sum_{s \in S} \sum_{l_{ij} \in L} (\sum_{a=1}^{|w|} (f_{ij}^{swc_a}) + f_{ij}'^{sw})(\delta_{ij} + SW(j)\delta_i) \\ + SW(j) \sum_{l_{km} \in L} cf_{km}^{j} (\delta_{km} + SW(m)\delta_k)\end{aligned} \tag{1}$$

subject to:

$$\sum_{s \in S} n_s \leq MAX_{FN} \tag{2}$$

$$\forall w \in W, a \in [1, |w|] : \sum_{s \in S} z_{sc_a}^{w} = 1 \tag{3}$$

$$\forall w \in W, a \in [1, |w|], s \in S : z_{sc_a}^{w} \leq n_s \tag{4}$$

$$\forall s \in S : \sum_{w \in W} \sum_{a=1}^{|w|} z_{sc_a}^{w} r_{c_a} \leq r \tag{5}$$

$$\begin{aligned}\forall l_{ij} \in L : \sum_{s \in S} [cf_{ij}^{s} \sigma + \sum_{w \in W} [(\sum_{a=1}^{|w|} f_{ij}^{swc_a} I_{c_a}) + (f_{ij}'^{sw} O_{c_{|w|}})]] \\ \leq \theta_{ij}\end{aligned} \tag{6}$$

$$\forall i, v \in V, w \in W : \quad \sum_{j \in V} f_{ij}^{vwc_1} - f_{ji}^{vwc_1} = \begin{cases} WR(w,v) & \text{if } i = v \\ -WR(w,v) z_{ic_1}^{w} & \text{if } v \in H \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

$$\forall i, v \in V, w \in W : \quad \sum_{j \in V} f_{ij}'^{vw} - f_{ji}'^{vw} = \begin{cases} z_{vc_{|w|}}^{w} & \text{if } i = v \\ -WR(w,i) & \text{otherwise.} \end{cases} \tag{8}$$

$$\forall i, v \in V, w \in W, a \in [0, |w|] : -z_{vc_{a-1}}^{w} + z_{vc_a}'^{iw} \leq 0 \tag{9}$$

$$\forall i, v \in V, w \in W, a \in [0, |w|] : -1 + z_{ic_a}^{w} + z_{vc_a}'^{iw} \leq 0 \tag{10}$$

$$\forall i, v \in V, w \in W, a \in [0, |w|] : z_{vc_{a-1}}^{w} + 1 - z_{ic_a}^{w} - z_{vc_a}'^{iw} \leq 1 \tag{11}$$

$$\forall i, v \in V, w \in W, a \in [0, |w|] : -z_{ic_a}^{w} + z_{vc_a}''^{iw} \leq 0 \tag{12}$$

$$\forall i, v \in V, w \in W, a \in [0, |w|] : -z_{vc_{a-1}}^{w} + z_{vc_a}''^{iw} \leq 0 \tag{13}$$

$$\forall i, v \in V, w \in W, a \in [0, |w|] : z_{vc_{a-1}}^{w} + z_{ic_a}^{w} - z_{vc_a}{''}iw \leq 1 \tag{14}$$

$$\forall i, v \in V, w \in W, a \in [2, |w|] : \quad \sum_{j \in V} f_{ij}^{vwc_a} - f_{ji}^{vwc_a} = \begin{cases} z_{vc_a}'^{iw} & \text{if } i = v \\ 0 & \text{if } v \in H \\ -z_{vc_a}''^{iw} & \text{otherwise.} \end{cases} \tag{15}$$

$$\sum_{s \in S} x_s \leq MAX_C \tag{16}$$

$$\forall s \in S : \sum_{s' \in S} y_{ss'} = 1 \tag{17}$$

$$\forall s, s' \in S : y_{ss'} \leq x_{s'} \tag{18}$$

$$\forall i \in V, s \in S : \quad \sum_{j \in V} f_{ij}^{vwc_a} - f_{ji}^{vwc_a} = \begin{cases} 0 & \text{if } i \in H \\ 1 - y_{si} & \text{if } i = s \\ -y_{si} & \text{otherwise.} \end{cases} \tag{19}$$

Eq. 1 expresses the optimization objective: to minimize the average response time of workflows. As previously stated, since each microservice has a constant execution time, the objective is to minimize workflow latency. Workflow latency includes both the latency of application traffic flows (i.e., traffic generated by requesting microservice execution) and the control latency of switches traversed by the application flows related to the workflow. Eq. 2 limits the number of FNs that can be placed. Eq. 3 states that each microservice request in a workflow can only be fulfilled once. Eq. 4 ensures that only FNs that are actually placed can execute microservices. Eq. 5 states that the total RAM consumed by the microservices executed in an FN cannot be higher than the available RAM of the FN. Eq. 6 enforces link capacity. Eq. 7 states that traffic flows for the first microservice request of each workflow are generated at the requesting host and consumed at the FN that executes the first microservice. Eq. 8 is similar, as it states that responses are generated by the FN that executes the last microservice requested in a workflow and consumed by the host that requested the workflow. Eqs. 9-14 define the value of new variables, that are created in order to maintain the linearization. Eq. 15 adapts the classic flow constraints to the generic case (i.e., microservices that are not the first ones in a workflow). Eq. 16 limits the number of controllers to be placed. Eq. 17 makes sure only one controller is assigned to each switch. Eq. 18 states that only placed controllers can be assigned to switches. Finally, Eq. 19 adapts the classic flow constraints to control flows.

By giving this formulation as an input to an automatic MILP solver, it is possible to obtain optimal FN, SDN controller and microservice placement, along with optimal routing.

IV. Performance evaluation

In this section, we evaluate Umizatou's performance over an IoMT case study, using a variety of scenarios with different parameters.

A. Evaluation environment

To evaluate Umizatou, the same IoMT case study presented in Sec. II is used: ECG and BP analysis in a smart hospital. The overall software and hardware architecture was taken from [12], while the technical details of microservices (e.g., RAM consumption, CPU cycles) are detailed in [16]. The formulation has been evaluated over topologies of different sizes, generated using the Erdös-Rényi model [17]. Users are the nodes of the topology that have a degree of one, and they are not considered SDN switches, but IoMT devices. Therefore, they are able to request workflows and generate traffic, although they are unable to route traffic, execute microservices, or host FNs or SDN controllers. The rest of the nodes, however, are considered to be SDN switches, and thus, potential FN and controller placements. An additional difference between switches and hosts are their connection technologies. Switches are connected to one another with Gigabit Ethernet links, as in [12]. On the other hand, IoMT devices can connect to the infrastructure making use of different wireless technologies. Following the case study from [12], four wireless technologies are considered: Wi-Fi, Bluetooth, ZigBee and 6LoWPAN. Each user requests a randomly chosen workflow, which can be either ECG or BP analysis. With the aim of evaluating the performance of the proposed solution with a minimal-cost fog deployment, we deploy the minimal number of required FNs in order to satisfy the set of requested workflows. The minimal number of fog nodes is assessed by considering the total RAM consumed by the microservice instances and the RAM of each fog node.

Umizatou makes use of the Gurobi MILP solver to apply the formulation presented in Sec. III to the different scenarios evaluated. These tests are performed in a computer with an Intel i7-8565U CPU and 16 GB of RAM. For each of the twelve scenarios detailed in Table I, different analyses have been performed. First, an analysis to evaluate the effectiveness of microservice deployment is performed. For this purpose, we evaluate both the microservice balancing and the load of the set of FNs. Moreover, an analysis on the response time obtained is also made to assess both the results obtained by Umizatou and the scalability of the deployment in larger topologies. All the parameters used for the tested scenarios are detailed in Table I.

B. Performance analysis

In Fig. 2, the response times for both functionalities, BP monitoring and ECG monitoring, are depicted in the three topologies. The first conclusion that can be drawn from this figure is that ECG monitoring consistently has a higher response time than BP monitoring. This increase comes from the fact that ECG monitoring requires for an additional microservice (compression), which adds both the execution time of said microservice as well as the possible latency from its execution in a different node to the overall response time of the workflow. However, the main conclusion in these results is that response time stays mostly stable over all three topologies, with a subtle decrease in topologies of larger sizes. Moreover, the standard deviation is extremely low, being in the order of 10^-5 ms. Therefore, we conclude that the proposed solution is scalable and response time is minimized.

In Fig. 3, the average share of microservices deployed per fog node and the average load of each fog node is depicted. Starting with the share of microservices, there are interesting results in the error bars. As depicted, the larger a topology is, the lower the standard deviation of this share is. Thus, the larger a topology is, the more evenly distributed microservices are. The even distribution comes from the fact that larger topologies have more fog nodes, and is thus easier to distribute microservices evenly while maintaining an optimal QoS. On the other hand, the average load of fog nodes is high in every topology, ranging from 86% (7 nodes topology) to 95% (50 nodes topology). This result is to be expected, as only the minimal number of FNs required was deployed. Moreover, larger topologies have a higher load. The standard deviation is larger different in the 7-node topology (14.43%) than in the other two (1.91% and 2.2%, respectively). This result implies that, in the same manner microservices are distributed more evenly in larger topologies, load is also distributed in such an even manner.

Fig. 4 depicts the CDF of the latency of all workflows in all three topologies. The main result extracted from this graph is, in essence, that larger topologies have larger latencies. This is depicted in two different ways. First, we find that larger

Table I: Parameter settings of the performed tests.

Scenario ID	Switches	Users	Micro-service instances	Fog Nodes	Wireless protocol (IoMT devices)
1	7	5	13	3	Wi-Fi
2	7	5	13	3	Bluetooth
3	7	5	13	3	ZigBee
4	7	5	13	3	6LoWPAN
5	20	15	38	8	Wi-Fi
6	20	15	38	8	Bluetooth
7	20	15	38	8	ZigBee
8	20	15	38	8	6LoWPAN
9	50	40	100	21	Wi-Fi
10	50	40	100	21	Bluetooth
11	50	40	100	21	ZigBee
12	50	40	100	21	6LoWPAN

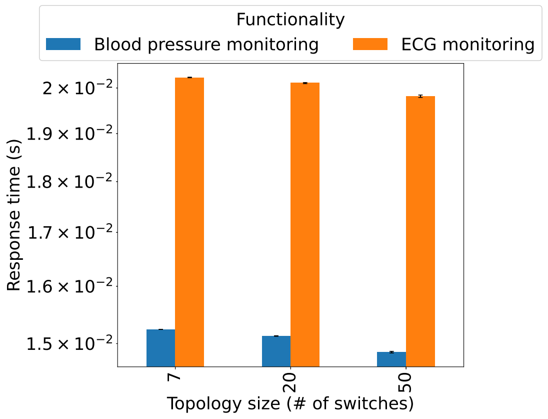


Figure 2: Response time.

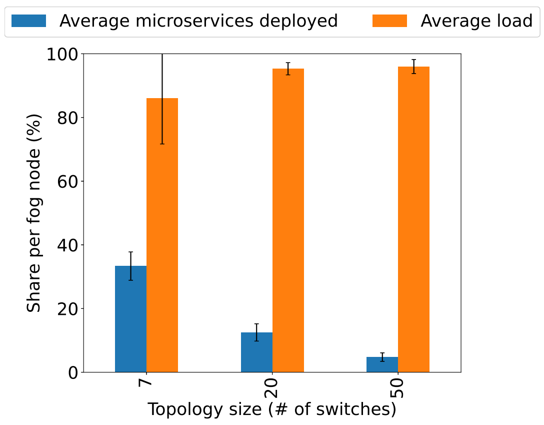


Figure 3: Deployment load and share.

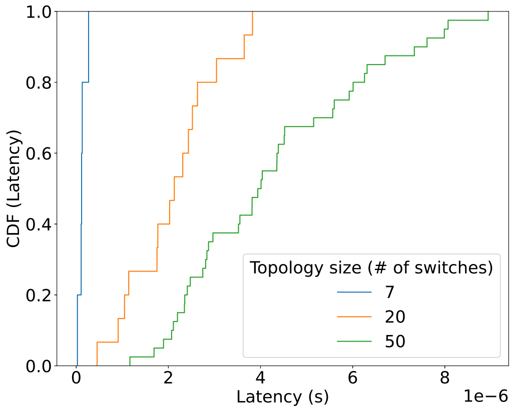


Figure 4: Latency empirical CDF.

topologies have their CDF placed more to the right. While not all workflows of larger topologies have higher latencies than the ones of smaller ones (e.g., some workflows of the 20-node topology have lower latencies than others from the 7-node topology), it can be generally stated that larger topologies have larger latencies. Nonetheless, due to the generally small latencies achieved by optimal FN placement, this does not affect the average response time in a significant manner, as Fig. 2 depicts.

V. Conclusions and Future Work

Applying IoT solutions to intensive domains requires a high QoS. This QoS can be achieved by optimally deploying the application's microservices, placing the fog nodes that run them, and placing the network controllers. However, the tight coupling between the application, computing and networking dimensions creates a relationship between their decisions. Thus, a joint, holistic solution that optimally places controllers, fog nodes and microservices is required. In this paper, we presented Umizatou, a MILP-based solution to this joint problem. Umizatou has been evaluated over an IoMT case study, obtaining response times under 20 ms for ECG monitoring and under 16 ms for blood pressure monitoring, with good scalability in increasingly large topologies. In the future, we expect to extend Umizatou by allowing it to use alternative optimization algorithms, such as heuristics, to improve the optimization times. Finally, we also expect to test Umizatou over real or emulated network test-beds.

Acknowledgements

This work has been partially funded by the project RTI2018-094591-B-I00 (MCI/AEI/FEDER,UE), 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 and Infrastructure of the Government of Extremadura (GR18112, IB18030), by the Valhondo Calaff institution, and by the European Regional Development Fund.

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] 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.

[3] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: State of the art and research challenges," *Computer*, vol. 40, no. 11, pp. 38–45, 2007.

[4] K. Indrasiri. Microservices in practice - key architectural concepts of an MSA. (visited on Jan. 14, 2021). [Online]. Available: https://wso2.com/whitepapers/microservices-in-practice-key-architectural-concepts-of-an-msa/

[5] 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, dec 2019.

[6] 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.

[7] 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.

[8] J. L. Herrera, P. Bellavista, L. Foschini, J. Galán-Jiménez, J. M. Murillo, and J. Berrocal, "Meeting stringent qos requirements in iiot-based scenarios," in *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE, 2020, pp. 1–6.

[9] 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 and Tutorials*, vol. 19, no. 4, pp. 2359–2391, oct 2017.

[10] V. Balasubramanian, M. Aloqaily, and M. Reisslein, "An sdn architecture for time sensitive industrial iot," *Computer Networks*, vol. 186, p. 107739, 2021.

[11] T. Das, V. Sridharan, and M. Gurusamy, "A Survey on Controller Placement in SDN," *IEEE Communications Surveys & Tutorials*, pp. 472–503, 2019.

[12] T. N. Gia, M. Jiang, A.-M. Rahmani, T. Westerlund, P. Liljeberg, and H. Tenhunen, "Fog computing in healthcare internet of things: A case study on ecg feature extraction," in *2015 IEEE international conference on computer and information technology; ubiquitous computing and communications; dependable, autonomic and secure computing; pervasive intelligence and computing*. IEEE, 2015, pp. 356–363.

[13] V. Hayyolalam, M. Aloqaily, O. Ozkasap, and M. Guizani, "Edge intelligence for empowering iot-based healthcare systems," *arXiv preprint arXiv:2103.12144*, 2021.

[14] I. Bedhief, L. Foschini, P. Bellavista, M. Kassar, and T. Aguili, "Toward self-adaptive software defined fog networking architecture for IIoT and industry 4.0," in *Proceedings of IEEE CAMAD 2019*, 2019.

[15] Open Networking Foundation, "OpenFlow Switch Specification 1.3.0," pp. 1–3205, 2013. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf

[16] A. Limaye and T. Adegbija, "A workload characterization for the internet of medical things (iomt)," in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2017, pp. 302–307.

[17] P. Erdös and A. Rényi, "On random graphs i," *Publ. math. debrecen*, vol. 6, no. 290-297, p. 18, 1959.

Everything that lives is designed to end.

NieR: Automata

Yoko Taro

Chapter 8

Conclusions and future work

This chapter concludes this thesis dissertation. In Section 8.1, we draw some general conclusions from the results obtained through the development of the PhD. A discussion of these results from the perspective of the stated research questions is given in Section 8.2. On the other hand, the main limitations of the proposed solutions are discussed in Section 8.3. Section 8.4 presents the future research lines. Finally, Section 8.5 concludes the thesis with the author's personal remarks.

8.1 Conclusion

The next generation of IoT applications has the potential of bridging the gap between computer systems and the real world, even for intensive domains, like industry or healthcare. Indeed, current paradigms for application design, computing, and networking control, such as MSAs, the Cloud-to-Thing continuum, or SDN, respectively, pave the road for next-gen IoT. However, an important challenge for this next generation is the optimization of the application's QoS, especially considering the characteristics and peculiarities of each of these paradigms. Hence, optimally placing the application's microservices, computing devices, and network controllers is key to enabling the next generation of IoT applications. Furthermore, the requirements may differ between companies developing IoT applications, and hence, optimizing QoS is not something that can be done using a single metric: multiple metrics on QoS must be possible to consider. Finally, once the application is deployed, the placements must be continuously adapted, integrating the adaptation task into existing continuous practices and methodologies like DevOps.

In this thesis, we have presented a set of approaches, models, tools, techniques, and architectures for tackling these problems. The main result of this dissertation is DADO [2], a framework for optimizing response time through the optimal deployment of IoT applications and SDN controllers. This result includes a multi-objective version of DADO, code-named NIoTO [5], which extends DADO's holistic approach with the possibility of optimizing the cost of the deployment, separately or jointly with response time. In parallel, we also defined the problem of placing the computing devices or FNPP, as well as developed an optimal and a heuristic solution to it [3]. These works converged in Umizatou [42], an all-in-one framework for optimizing all three dimensions holistically. Along with these 5 artifacts, we also presented 9 additional supporting artifacts that are directly related to DADO. Out of them, we would like to highlight ConDADO, a version of DADO meant to dynamically adapt the application placement to environmental changes making use of continuous reasoning, as well as integrating with DevOps CI/CD pipelines.

8.2 Discussion

This section summarizes the contributions of this thesis w.r.t. the RQs detailed in Section 1.3. Concretely, we defined 5 RQs:

RQ1 **Are the decisions taken by QoS optimizations performed in each separate dimension different from those taken with a holistic approach?** As of this thesis, the answer to this question is **Yes, depending on the scenario**. Holistic optimization allows the system to have information from its own decisions in other dimensions to decide on the optimal solution. This is especially evident when the system provides a counter-intuitive solution, which takes decisions that sacrifice QoS in one dimension because those decisions *unlock* the possibility of making other decisions that improve QoS further. One of these cases is depicted by Figure 6 in [2] (Chapter 4): DADO decides to place SDN controllers so that the overall traffic latency increases, because doing so allows for the placement of microservices in more powerful nodes that are further away, hence decreasing the total response time. If the SDN controller placement system did not know of the microservice placement system, it would be illogical and sub-optimal to place the controller in that manner. As a consequence, a separate microservice placement system would be unable to reach such nodes with a low enough latency to decrease the response time. The more complex the scenario is, the more likely it is that these situations may arise. Hence, although it may not be the case for the smaller and simpler scenarios, we conclude that, if the possibility for further optimization exists due to the complexity of the scenario, the response to RQ1 is positive.

RQ2 **Is it possible to jointly consider multiple points of view on QoS?** Our works in the context of this thesis conclude that the question to RQ2 is **Yes**. In the case of holistic IoT application deployment optimization, it is possible to consider multiple QoS metrics, even if they come from multiple points of view and must be traded off. Based on this idea, artifacts such as NIoTO [5] or MO-SFO implement multi-objective optimization, considering two objectives from different points of view: response time from the technical point of view, and economic cost from the business point of view. Both of them can consider each objective separately and both objectives jointly to meet the developer or operator's requirements.

RQ3 **Which models and techniques are useful for solving the joint optimization problem?** This is, possibly, the most extensive question of all five RQs, as it requires a list of the proposed solutions. In summary, we could say that **We have proposed multiple models that allow state-of-the-art optimization techniques to be used**. Concretely, starting with the modeling of the optimization problem, we have developed DADO's optimization model [2], the FNPP's model [3], Umizatou's all-in-one model [2], S-DADO's dynamic model, and the simplified and expressive model for μDADO. Regarding optimization techniques, we have eminently used MILP for solving the optimization problems [2–5,37,42,44]. Nonetheless, we have also made used of additional state-of-the-art techniques, both to support MILP and as completely standalone solutions, such as unsupervised machine learning ([3]), Lyapunov dynamic optimization [46] (S-DADO), continuous reasoning [60] (ConDADO), or Adaptive Kernel Search [59] (Vernier).

RQ4 Can the decisions taken during optimization adapt to changes in the dimensions' environments? The answer to this question is also **Yes**. In the context of this thesis, we have proposed multiple artifacts that adapt the optimization decisions to the changes that arise in the environment, and more concretely, in each of the decisions. Concretely, ConDADO, S-DADO, μDADO, and Faustum are artifacts that perform adaptative optimization, proactively in the case of S-DADO, and reactively in the rest.

RQ5 How does this optimization fit in current development lifecycles and practices? During the development of this thesis, we have concluded that the answer to RQ5 involves two key points in the application development lifecycle. **The optimization presented in this thesis fits both in the late design phase and in the operation phase of the lifecycle, and the latter can be integrated into DevOps practices**. Purely optimal frameworks (e.g., DADO, NIoTO, Umizatou) are expected to be used at the late design phase, to ensure the application is initially deployed in optimal conditions of QoS. During runtime, in the operation phase, we proposed CA as a manner to integrate the adaptation of the optimization decisions into the existing DevOps practices. Concretely, ConDADO is our main proposal for its use at operation time.

8.3 Limitations

During the development of the PhD, we also found some limitations to the works presented in this thesis:

- *Integration in an orchestrator.* Although some of the artifacts we have developed, like PODS, enable the migration of microservices, the ConDADO framework, as well as the future multi-layered continuous reasoning system, is yet to be integrated with microservice orchestrators like Kubernetes. This limitation is mainly a threat to the user-friendliness of the developed artifacts. Due to our work in PODS and Pascal, we know it is possible to integrate ConDADO into such orchestrators. However, as of now, the developer must be the one to write the appropriate parsers to create this integration, both from the monitoring system to DADOJSON and from DADOSIM to placement in Kubernetes.

- *Pareto front generation.* Due to the MILP model used in our multi-objective systems (NIoTO, MO-SFO, μDADO), each of the artifacts output a single solution, which trades off the multiple objectives as the developer or operator decides. However, other models and multi-objective systems, such as genetic algorithms, can provide a set of Pareto-optimal solutions known as the Pareto front, where each solution can be considered *optimal* because an improvement in an objective involves the worsening of at least another objective (e.g., a shorter response time also involves a higher cost). In some cases, some developers or operators may find useful a feature that extracts a Pareto front that can be evaluated by a human, rather than a single solution. Rather than a threat or weakness of the existing work, this simply represents a limitation in terms of optional, useful functional features.

- *User-friendliness of the DADOJSON format.* The final user of the optimization artifacts developed through this thesis, generally, a developer or an operator is expected to describe their planned scenario in a concrete format to make use of the existing artifacts. The family of formats that are used for the different artifacts can be described as modifications of the original format we created for DADO, the DADOJSON format. This format is a JSON schema that describes each of the characteristics of the scenario in a simple manner. However, we have yet to test if the DADOJSON format is user-friendly enough for our target userbase to directly describe their scenarios in it or if, on the contrary, the developers or operators would greatly benefit from a concrete UI (e.g., a DADOJSON builder GUI).

- *Integration of artifacts*. Many of the artifacts that have been presented in this thesis could be used together to perform specific, more complex tasks. For example, NIoTO and MO-SFO can be used in conjunction with ConDADO to obtain an initial placement that ConDADO will adapt over time, or Pascal can be used instead of ConDADO to more accurately emulate the effects of the adaptation on the environment. However, as of now, although most of the artifacts make use of very similar formats, as they come from a similar baseline, they are not directly integrated. As they are all developed and packaged as standalone artifacts, the developer is on charge of this integration, either by the creation of automated processes or by manually performing the appropriate changes to data in order to make use of the integrated artifacts.

8.4 Future work

A part of the future, as well as ongoing work, is to address the limitations we have described in the previous section. Furthermore, we consider that this PhD has opened many new lines of research and engineering work:

- *Communication of existing and work-in-progress artifacts*. Out of the 14 presented artifacts, 5 are still in the Design Cycle of our methodology, and hence, we are yet to have a publication related to them. Concretely, PODS (and by extension, μDADO), Pascal, S-DADO, Faustum (including the multi-layered continuous reasoning stack), and Vernier are yet to be published. In the near future, we expect to publish journal articles or conference papers related to these artifacts and their advances w.r.t. the state of the art. We are currently collaborating with the University of Cantabria (Spain), the University of Pisa (Italy), the University of Bologna (Italy), and The University of Texas at Austin (United States of America) in this line of work.

- *Human-defined constraints to candidate solutions*. In some cases, the developers or operators may wish for a microservice placement solution where the replication and placement of some microservices are already fixed for privacy reasons. For example, microservices that process confidential data of the organization should be deployed on-premises, in the organization's own edge or fog nodes, regardless of the underlying infrastructure. It would be desirable for the developer or operator to be able to express this sort of additional constraints simply, using models they are familiar with. We are currently collaborating with the LoUISE research group of the University of Castilla-La Mancha (Spain) in this research line.

- *Holistic heuristics for multi-objective optimization*. One of the caveats of the MILP-based optimization frameworks is their high memory consumption. Our models are exclusively made out of binary variables, which mitigates this problem, but up to a certain limit. Concretely, we have found that our MILP solver, Gurobi[1], is too memory-intensive if the infrastructure has more than 300 nodes [2]. For very large infrastructures, it could be desirable to have a heuristic solver, especially if it is less memory-intensive. Heuristics could also allow the frameworks to obtain the Pareto front in multi-objective optimization scenarios. Thus, we believe this research line can be a very interesting future work.

- *Study on the user-friendliness of the developed artifacts*. As we have mentioned in the limitations section, the user-friendliness of our tools is yet to be studied,

[1]`https://www.gurobi.com/`

analyzed, and possibly improved. While we believe that the use of the artifacts themselves should be fairly simple for technical users, as they provide a self-documented, straightforward, command-based CLI, it is still necessary to study the user-friendliness of the DADOJSON format, as well as its related formats, the output CSV format, and the DADOSIM format used by Pascal. Furthermore, this research line can include the development of DADOJSON editors and visualizing tools to aid developers and operators in their tasks.

- *Development of distributed optimization models*. Another interesting line of work is to develop distributed algorithms that can optimize the QoS with a partial view of the system, instead of the global view used by our artifacts. While the global view is an acceptable assumption, as it can be obtained through the SDN controller(s) and the monitoring system, some use cases of the framework can benefit from a distributed approach. Concretely, this would be interesting from the point of view of a multi-tenant infrastructure. Our existing frameworks can optimize the QoS for an MSA, which may include more than one application, as the microservice and workflow model allows for the possibility of having workflows tied to multiple applications. However, in a scenario where multiple devices are controlled by multiple parties, where each of the parties wants to deploy one or more applications, and a *fair* deployment must be found, the frameworks would require all parties to freely share their scenario information, which may not be desirable if the parties do not trust each other. For these situations, we want to develop distributed optimization models in the future.

8.5 Personal remarks

As the final conclusion, I would like to add my personal remarks. This PhD thesis is the result of 3.5 years of work, from October 3rd, 2019, until the defense in 2023. Becoming a PhD has been one of my targets in life for a very long time, and hence, in a way, this feels like a major milestone. However, if I had to describe the thesis with one word, it would be *learning*. Learning about the current environments of IoT applications, learning from others about state-of-the-art techniques, learning the foundations of optimization techniques, models, systems, and automatic solvers, learning about the gaps in the literature, learning about the details of the system's behavior, and learning about other techniques and considerations that could be integrated with our work. And, quoting the martial artist Bruce Lee, *"Learning is definitely not mere imitation, nor is it the ability to accumulate and regurgitate fixed knowledge. Learning is a constant process of discovery - a process without end."*

Appendix A

Context-Aware Privacy-Preserving Access Control for Mobile Computing

ID : **PMC 2022**

Authors : Juan Luis Herrera, Hsiao-Yuan Chen, Javier Berrocal, Juan M. Murillo, Christine Julien

Publication type : Journal article

Journal : Pervasive and Mobile Computing

DOI : `https://doi.org/10.1016/j.pmcj.2022.101725`

JCR : Q2 (Computer Science, Information Systems). IF: 3.848.

Context-Aware Privacy-Preserving Access Control for Mobile Computing

Juan Luis Herrera[a], Hsiao-Yuan Chen[b], Javier Berrocal[a], Juan M. Murillo[a], Christine Julien[b]

[a] *University of Extremadura, Spain*
[b] *University of Texas, Austin, United States*

Abstract

In mobile and pervasive computing applications, opportunistic connections allow co-located devices to exchange data directly. Keeping data sharing *local* enables large-scale cooperative applications and empowers individual users to control what and how information is shared. Supporting such applications requires runtime frameworks that allow them to manage the who, what, when, and how of access to resources. Existing frameworks have limited expressiveness and do not allow data owners to modulate the granularity of information released. In addition, these frameworks focus exclusively on security and privacy concerns of data *providers* and do not consider the privacy of data *consumers*. We present PADEC, a context-sensitive, privacy-aware framework that allows users to define rich access control rules over their resources and to attach levels of granularity to each rule. PADEC is also characterized by its expresiveness, allowing users to decide under which conditions should which information be shared. We provide a formal definition of PADEC and an implementation based on private function evaluation. Our evaluation shows that PADEC is more expressive than other mechanisms, protecting privacy of both consumers and providers.

Keywords: privacy, device-to-device, mobile computing, access control

1. Introduction

During the last several years, there has been a massive increase in the deployment of mobile devices [1]. Similar trends exist for Internet of Things (IoT) and wearable devices [2]. These devices carry a large number of on-board sensors that provide a rich view of the surrounding context and support a wide array of applications. In addition, these devices commonly accompany the user, providing an interface between their owner and a wider networked community [3].

This astounding increase in *companion devices* enables new systems and applications. Mobile crowd sensing recognizes that a user's sensing needs can often be fulfilled by others nearby [4]. Opportunistic data sharing [5], which can be combined with mobile crowd sensing, relies on transient wireless connections [6] to distribute and fulfill a data sharing task using information from local networked devices. Both domains rely on communities formed based on human contact [7]. As interactions become more opportunistic, users need mechanisms to control the release of data according to their privacy needs. In decentralized and opportunistic scenarios, trust among peers and the security of shared information are crucial [8]. Nevertheless, it is also essential to implement privacy management so that users can manage who, when, and how their personal information is accessed [9].

As a running example, we describe an opportunistic data sharing application for the city of New York. This application leverages users' sensors to support both tourists and residents. Concretely, one function of the application is to store points of interest (POIs) identified by residents and then offer this information to tourists or other residents via a microservice. Obviously, information about an individual's visits to particular POIs may be sensitive, and residents should be able to decide with whom their POI information (e.g., date, purpose, people they visited with, review, etc.) can be shared and under what conditions. For instance, some residents may share this information only with other users that belong to similar social groups, or with tourists that have a specific context (e.g., are physically nearby).

In the opportunistic space, contextual information becomes very important, as the user will often interact with strangers. Contextual information is thus defined as any kind of information that allows for the characterization of a user in a given situation, e.g., their identity, location, time of the day, sound level, interests or mood [10]. The context of a user is, therefore, a complete set of contextual information that characterizes the user's current situation. In opportunistic environments, context-aware access controls constrain access to data or services based on contextual conditions [11]. In contrast, basic access control, such as Role-Based Access Control (RBAC) [12] or Dynamic Sharing and Privacy-aware Role-Based Access Control (DySP-RBAC) [13], control access to information based solely on the identity of a consumer. These mechanisms are not sufficiently expressive to support opportunistic applications in which the identities of most peers are not known *a priori*. In addition, existing techniques do not allow data and service providers to consider dynamic and contextual conditions when constraining access. For instance, residents in our example would not be able to limit access to POI information to tourists who are nearby. Attribute-Based Access Control (ABAC) [14] and the access control mechanism from [15] do allow users to define access rules based on contextual conditions beyond identity. However, these mechanisms cannot modulate the precision of information released to each peer as a function of the shared context, which prevents providers from exerting fine-grained control over the access. For example, these techniques would not allow residents to release only a partial set of POIs depending on the contextual situation of each tourist. Furthermore, these systems are designed for centralized (cloud-based) data sharing in which a coordinator mediates access. This is not the case in our scenarios, which are often completely decentralized and rely on the provider and consumer coordinating directly.

Finally, attribute-based or context-aware techniques usually require consumers to share their whole context to evaluate if they can access the offered services. In a cloud-supported system, a trusted third-party has an omniscient view of this information. However, in opportunistic environments, the exchange of context to support access control is peer-to-peer. This could lead to an increase in the exposure of consumers' private contextual information.In our running example, tourists would have to release details about their social groups, noise level, location, etc. to get access to POI information of a provider, regardless of the granularity of the information they desire. There is thus a need for access control mechanisms designed for opportunistic environments, that focus on expresiveness to allow users to precisely describe the circumstances under which they will share each specific piece of information, while maintaining crucial access control features such as privacy, as well as considering the context-aware nature of opportunistic exchanges.

We present a context-aware, privacy-preserving access control mechanism designed for completely decentralized data sharing (PADEC, Privacy-Aware DEvice Communication). PADEC empowers users by increasing the expressiveness of access rules and protecting the privacy of both providers and consumers. PADEC allows data and resource owners to define rich access rules based on widely varying contextual information. Providers attach filters to rules to provide access at different levels of granularity based on a consumer's provided context. At the same time, PADEC protects consumers' privacy by minimizing the amount of contextual information they need to share in their access requests. This paper's novel contributions are:

- We provide a formal specification of PADEC to allow for exact replication, implementation, and verification of PADEC.

- We use this formalization to show how to employ PADEC.
- We provide details about PADEC's implementation using private function evaluation [16, 17] to increase privacy and security.
- We apply PADEC in a large case study.
- We evaluate PADEC using a threat model and assess how PADEC mitigates each threat while also reducing consumers' and providers' exposure.
- We evaluate the probability of type I and type II errors that exist in PADEC's access control evaluation.
- We compare PADEC with state-of-the-art approaches for access control.
- We perform a user survey to evaluate PADEC's usefulness and the effectiveness of using context for access control.

Section 2 introduces related context-sensitive and privacy-aware access control mechanisms. Section 3 describes PADEC's threat model. Each threat is dealt with in the PADEC model, as detailed in Section 4. Section 5 provides PADEC implementation detail, and a case study and user survey are presented and evaluated in Section 6. Finally, Section 7 concludes.

2. Related Work

During the last few years, different context-sensitive, decentralized or privacy-aware access control mechanisms have been developed.

Context-sensitive access control. A variety of recent efforts use personal context information to authenticate users. Hayashi *et al.* present CASA [18], which uses context information such as nearby familiar devices or locations to apply an access control policy that adapts the authentication method of mobile phones to contextual factors. CASA changes the information required for authentication, such as fingerprint or password, to make it more convenient to unlock a device in familiar and safe situations, while strengthening security in less familiar ones. Similarly, BANDANA [19] relies on users' movements, like gait, to validate identities. BANDANA uses different devices worn on the users' body to obtain fresh secrets for ad-hoc communication between these devices, which are able to recognize the person wearing them. While these approaches demonstrate the interest of using contextual attributes in pervasive and mobile computing environments, these systems require an *a priori* model of each user, which is not feasible for large-scale applications, especially those that need to grant access to unknown users depending on their context.

Decentralized access control. Other existing frameworks control access to shared data without using a centralized point or database. For instance, Penumbra [20] is a decentralized access control system that could be used in opportunistic scenarios like those we seek to support. Penumbra proposes a decentralized filesystem where users can use tags on files to define the access control policy. However, only the identity of the requester can be used to define these policies, omitting other contextual information. Shafeeq *et al.* developed an approach based on storing access control policies in a blockchain [21]. Concretely, their approach stores access rights and policies in a distributed ledger, encrypting them to ensure their privacy and decrypting them only when they are applied. Moreover, the approach also supports the use of multiple contextual attributes. This approach guarantees the access control is distributed and can be audited. Although this mechanism is also privacy-aware, privacy is only preserved if access decisions are made by honest participants, which is difficult to ensure in opportunistic scenarios.

Privacy-aware access control. Opportunistic and pervasive computing require decentralized and context-sensitive access control in which individual users can modulate the specificity of responses based on a consumer's context. Role-Based Access Control (RBAC) [12] is a NIST-standard [22] that authenticates users based on identity by grouping them in roles. RBAC is a default access control mechanism in modern technologies, such as commonly used cloud platforms [23–25]. However, RBAC is not context-sensitive nor privacy-aware. DySP-RBAC [13] is a state-of-the-art extension of RBAC, adding context-sensitivity and privacy-awareness. However, DySP-RBAC is implemented by applying a centralized architecture, in which all users are known in advance and do not retain ownership of their data. Moreover, context is based on users' relationships and is therefore fully identity-based.

Attribute-Based Access Control (ABAC) [14] is also a NIST-standard [26] model supported by platforms such as Amazon Web Services [23]. ABAC is potentially applicable for context-sensitive access control. However, it is not privacy-aware, as it does not allow providers to set different levels of detail for information shared with different consumers. The work in [15] proposes an alternative for pervasive computing in which users can define access control policies by providing conditions over context. However, this mechanism also requires a centralized knowledge base with context information from all users, which takes ownership of contextual information away from users and is unsuitable for opportunistic scenarios. Furthermore, it does not allow users to control access at different levels of granularity.

Semantic-Based Access Control (SAC) [27] uses semantic web technologies to derive new contextual attributes and relationships from ontologies of existing attributes, using an ABAC-based mechanism to perform access control. Hence, SAC enriches ABAC by automating the generation of additional contextual attributes. While SAC can be useful from an user-friendliness perspective, SAC employs ABAC as a back-end mechanism, and thus, in terms of expressiveness, privacy preservation and access control, it is functionally equivalent to ABAC.

Finally, Mhetre *et al.* introduce Experience-Based Access Control (EBAC) [28]. EBAC is tailored to provide access control in situations where a high number of devices interact frequently with each other in groups. EBAC simplifies the complexities of manually managing access control policies by converting policies into a rating between 0 and 1 of how good past experiences with a device need to be for access to be granted. This experience rating is calculated based on the properties of the requester device, such as its reliability or its interaction history, and can be propagated using transitive relationships. However, the requirement for frequent interactions within a closed group is quite different from our challenge, where access control must be performed for unknown devices that will not interact frequently.

While privacy-aware and context-sensitive access control mechanisms exist, none are designed for opportunistic scenarios, making them unsuitable for pervasive computing environments. Efforts in privacy-aware access control are not often coupled with context-sensitivity and *vice versa*, and those that address both do not consider the needs of opportunistic data sharing.

3. Threat Model

We next define a threat model that we use to present the remainder of our contribution. In our system model, a *provider* voluntarily shares data or resources with *consumers*, as long as certain conditions set by the provider are met. These conditions may constrain the provider's own context, the consumer's context, or a combination of the two. The provider can also filter or obfuscate information it provides based on the context. We assume an attacker whose objective is to obtain unauthorized information by exploiting any weakness of the system. Attackers can have up to three roles:

- They can masquerade as legitimate consumers and try to obtain information from providers.
- They can masquerade as legitimate providers, trying to obtain information from consumers
- They can be third-party attackers, trying to obtain information from messages exchanged by providers and consumers.

General Threats		
UA	*Unauthorized Access*	attacker poses as a consumer to try to obtain some data that the provider desires to deny access to.
CC	*Circumventing context constraints*	attacker poses as a consumer and attempts to use identity to access data when that access would be denied based on other context.
Individuals' Privacy Threats		
CE	*Consumer over-exposure*	an attacker using a modified PADEC poses as a provider to obtain information from a consumer using context in a way that is incongruent with the consumer's intent.
PE	*Provider over-exposure*	an attacker posing as a consumer obtains information with a finer granularity or higher precision than a provider desires to grant access to
IA	*Insider attack*	an attacker posing as a consumer obtains finer-grained information than a provider allows by correlating and aggregating results of repeated allowed requests.
Threats from Third Party Attackers		
EA	*Eavesdrop attack*	a third-party attacker obtains private messages shared between providers and consumers.
RA	*Replay attack*	a third party attacker obtains a legitimate message and replays it to incorrectly obtain access.

Table 1: Threats identified and addressed in PADEC.

We build our threat model incrementally, first considering common general threats and then moving to threats specific to a context-aware and opportunistic approach (Table 1). We start with two general threats: *Unauthorized access* (UA) and *Circumventing context constraints.* Because we aim to create a privacy-aware access control model, we also identify three threats to individuals' privacy: *Consumer over-exposure* (CE), *Provider over-exposure* (PE), and *Insider attack* (IA). Finally, we also identify two third-party threats: *Eavesdrop attack* (EA) and *Replay attack* (RA).

We assume that an anti-tampering mechanism such as remote attestation [29] exists, and thus, contextual information cannot be forged. Our approach directly addresses the remaining threats, as described next.

4. The PADEC Model

We next present the model of our context-aware, privacy-sensitive access control mechanism, PADEC. Section 4.1 presents the general architecture and its key concepts, while Section 4.2 presents the formal model. Finally, Section 4.3 applies PADEC in a mobile computing case study.

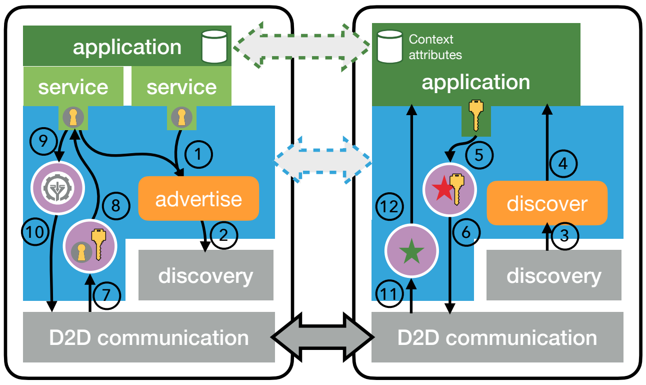


Figure 1: A PADEC interaction is initiated when a service provider (left) advertises a service (1). The advertised service is shared with nearby devices via an underlying discovery mechanism (2-3) and becomes available to nearby consumers (4). The advertisement contains the keyholes for the service, which a consumer uses to construct a key (5) that it places in an application request (6) and sends to the provider. The provider checks the key against the service's keyholes (7) and invokes the service (8). A filtered or obfuscated response (10) is returned to the consumer (11) and delivered to the application (12). As the horizontal arrows indicate, abstractly, the provider and consumer applications communicate with one another without considering the underlying mechanics. PADEC (the blue box in the figure) holds the semantics of keys and keyholes and the nature of requests and filtering, while the D2D communication is concerned with networking details.

4.1. Architecture and key concepts

PADEC is designed as a layer placed *on top* of the endpoints exposed by an application. One of the tenets is a separation of concerns: the application endpoints should not be aware of how access is controlled, and, in turn, PADEC should be aware of the specifics of the application such as data semantics. The focus of PADEC is to offer users an expressive manner to decide which information should be shared and under which conditions. Figure 1 depicts PADEC's architecture. Its key concepts are as follows:

1. **Lock**: locks are the elements of PADEC that protect endpoints. Each lock protects a single endpoint, and each endpoint has only one lock. A lock is a collection of *access levels*, sorted in non-decreasing order of QoI degradation (defined below).
2. **Access level**: access levels encapsulate a given granularity for the information provided by an endpoint, as well as the circumstances under which it can be accessed. Each access level contains a *filter* and a *rule*.
3. **Filter**: a filter is an abstraction of a mechanism to degrade or alter the granularity or the level of detail of the information returned from

an endpoint. PADEC's built-in filters can be tailored by users by setting parameters. New filters can be added by injecting small pieces of code. Each filter comprises the filtering technique, user-tailored parameters, and the expected Quality of Information (QoI) degradation. As information passes through a filter, its quality is degraded to fit the provider's desired QoI. Filters effectively address *PE* threats, and filtering techniques must be idempotent in order to avoid *IA* threats.

4. **Rule**: a rule in PADEC represents the conditions under which access is granted to a given access level. Rules can be defined over one or more contextual *attributes* of the provider, the consumer, or combinations of attributes (e.g., relative distance from the consumer to the provider) and define values for each attribute where access is granted. Rules tackle *UA* threats by controlling access to information and allow attributes other than identity to be used to control access, thwarting *CC* threats.
5. **Attribute**: an attribute is a type of contextual information, such as location, interest or social group. The context of a user comprises the values of all of their attributes at a given point in time.
6. **Keyhole**: a keyhole is the set of contextual attributes a rule requires from the consumer in order to be evaluated. Keyholes are how consumers perceive the rules associated to access levels. Producers advertise locks as collections of keyholes, each associated with an expected QoI degradation. Keyholes enhance PADEC's efficiency and decrease *CE* threats by minimizing the amount of exposed contextual information.
7. **Key**: a key is the set of values for the attributes defined in one or more keyholes. When consumers discover a lock, they select the access level(s) they want to try and build a key that fits those keyholes. On the provider, keys are used to evaluate the rules of the lock's access levels. To ensure privacy, keys are never shared from PADEC to the application.

There is an additional feature that can be enabled in PADEC, but has no effects over the architecture: private function evaluation (PFE). PFE [16] allows PADEC's key evaluation to be performed in a completely private manner: consumers provide an encrypted version of their key that can only be used a single time, while providers can evaluate whether their rule holds without decrypting the key or revealing their rule logic. The use of PFE enables PADEC to completely mitigate the *CE* threat. Keys do not change conceptually, although their communication is performed in a different manner. The details of these communications are detailed in Section 5.2.

Finally, with regards to the device-to-device communication, PADEC ensures that all device-to-device communications are encrypted, avoiding both third-party attacks in Table 1. To encrypt communications, the provider and consumer follow an ephemeral Diffie-Hellman key-agreement protocol [30], in a manner similar to SSL/TLS. The consumer generates an ad-hoc public key and shares it with the provider, including some of the parameters used to generate the public key. This key is *public* because it is shared with the other party. A new key is generated for each PADEC exchange, used exclusively for key agreement within the communication [30]. The provider, using the settings received from the consumer, generates its own public key, which is shared with the consumer. Finally, based on the shared public keys, both consumer and provider generate a secret. This secret is guaranteed to be the same for both parties and is used as a symmetric encryption key to protect the rest of the communications. Since messages are encrypted, eavesdropping is not a concern, since the attacker cannot examine message content. *RA* threats are also thwarted, since any answer the attacker might get by replaying messages will be as unreadable as any message they overhear.

Each data exchange is supported by a continuously executing endpoint discovery process [31]. The workflow that is used to communicate back and forth in a PADEC interaction is depicted by Figure 2. The ephemeral Diffie-Hellman protocol can be piggybacked onto this discovery process: the consumer can send its public key with its discovery query (step 1 in Figure 2), and, as part of returning the endpoint, the provider can share their public key. The agreed-upon symmetric key can be used to encrypt subsequent communications. Furthermore, the complexity of the main steps of the framework (2, 4 and 6 in Figure 2) is the same, which depends on the number of access levels the lock of the requested endpoint has, which we denote α, and the amount of attributes each of the rules of the access levels are defined over, which we denote τ. Hence, the interactions using PADEC have a complexity of $O(\alpha\tau)$ in Big O notation. In terms of a generic problem size n, PADEC can be considered to be $O(n^2)$.

4.2. The PADEC Formal Model

This section provides a formal description of how access is controlled, making PADEC easier to re-implement or replicate.

Users and endpoints. The base components are users ($\mathcal{U}$), who participate in a PADEC interaction. Users can be humans, devices, or applications and each user can act as a provider, a consumer, or both. A provider exposes a set of endpoints ($\mathcal{E}$) belonging to application programming interfaces (APIs) that consumers can discover and access to leverage some data or service. Formally, an endpoint $e \in \mathcal{E}$ is a function that maps an input to an output: $e : I \to O$, and a given provider's endpoints can be referenced

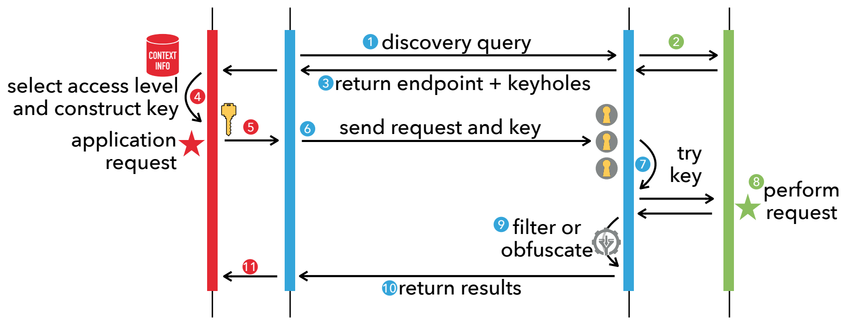


Figure 2: PADEC Sequence Diagram: A consumer (on the left side) discovers an endpoint and its access levels in the surroundings (1-3). The application selects an access level via a *keyhole* (4) and makes an application request (5). The request is sent to the provider (on the right side).The provider checks the consumer's key against the set of available keyholes (7) and, if access is granted, the provider performs the request (8) and returns the results (10-11), after applying a filter (9).

as $\mathcal{E}^u = \{e\}$ for provider $u \in \mathcal{U}$. An endpoint is identified by its *interface*, and therefore the same endpoint may be provided by multiple providers.

Attributes. In PADEC, attributes ($\mathcal{A}$) describe the contextual state of users. Each user is associated with a set of attributes that describes their situation; a particular user's attributes are defined by:

$$attributes : \mathcal{U} \rightarrow \mathcal{P}(\mathcal{A}) \tag{1}$$

where $\mathcal{P}(\mathcal{A})$ is the power set of $\mathcal{A}$. For a given user u, $attributes(u) \subseteq \mathcal{A}$.

Each attribute has a primitive type, which constrains what a legal value is (e.g., locations are pairs of longitude and latitude). A particular application may provide additional constraints on the value. For instance, locations of points of interest may be constrained to be within a particular city's boundaries, and ratings, while integers, must be between 0 and 5. For the formal model, we refer generically to the universe of possible values as $\mathcal{V}$, with the understanding that, for a given attribute a, the set of possible values is a subset of $\mathcal{V}$.

Context state. The *context* is defined as a function that maps each user $u \in \mathcal{U}$ to a set of values for each attribute:

$$context : \mathcal{U} \nrightarrow \mathcal{A} \times \mathcal{V} \tag{2}$$

The *context* function is a *partial function*; the attributes that are relevant for a particular user may be only a subset of the full attribute set $\mathcal{A}$. It can be more convenient to reference the context state of a user $u \in \mathcal{U}$ as a mapping of the user's contextual attributes to concrete values:

$$\mathit{context}(u) : \mathit{attributes}(u) \rightarrow \mathcal{P}(\mathcal{V}) \tag{3}$$

Rules. PADEC defines *rules* ($r \in \mathcal{R}$) that allow providers to restrict access to resources based on a consumer's context. A rule comprises one or more *clauses* that constrain the values of context attributes. PADEC defines set operators for discrete attributes and comparison operators for attributes whose values are ordered or partially ordered. In total, PADEC provides six operators($\mathcal{OP}$): $\in$, $=$, $\neq$, $>$, $<$, and $<>$ (which tests whether a value is within a specified range). To combine clauses over multiple attributes, PADEC supports combinations using $\wedge$, $\vee$, and $\neg$.

To fully specify rules and clauses, we use a context-free grammar [32], $G = (N, \Sigma, P, S)$. The sentence symbol of G is a rule, i.e., $S = r$. Because a PADEC rule is composed of clauses that combine attributes, operators, and values; the clause is the only nonterminal symbol, i.e., $N = \{c\}$. The terminal symbols are $\Sigma = \{a, op, v, \vee, \wedge, \neg\}$. These symbols connect directly to the PADEC formal model: $a \in \mathcal{A}$, $op \in \mathcal{OP}$, $v \in \mathcal{V}$ and $r \in \mathcal{R}$. The grammar captures the creation of rules through clauses, attributes, values, and operators via its production rules:

$$\begin{array}{lll} r \rightarrow r \vee c, & r \rightarrow r \wedge c, & r \rightarrow \neg r, \\ r \rightarrow c, & c \rightarrow a \ op \ v, & c \rightarrow v \ op \ a \end{array}$$

Although the final two production rules are similar, they allow for asymmetric operators (e.g., $\in$). Moreover, the first two production rules allow one to combine multiple clauses, thus also allowing for combined rules.

At a higher level, a PADEC rule can be viewed as a function, defined by its clauses, that maps a context state or a subset of a context state to a Boolean value:

$$\mathit{rule} : \mathcal{R} \times \mathcal{C} \rightarrow \{0, 1\} \tag{4}$$

Generically, we refer to the universe of rules as $\mathcal{R}$.

Keyhole. Consumers need to know what context information they need to provide to access an endpoint. The *keyhole* captures the set of contextual attributes required by the rule, limiting the state the consumer needs to provide. Given a rule (r) that combines clauses, each of which references an attribute of $\mathcal{A}$, the *keyhole* for rule r is simply the union of the set of attributes referenced by all of the clauses:

$$\mathit{keyhole}(r) = \{a : a \in r\} \tag{5}$$

Given that $\mathcal{R}$ is the set of rules in a PADEC system, we define $\mathcal{H}$ to be the complete set of possible keyholes:

$$\mathcal{H} = \{keyhole(r) \forall r \in \mathcal{R}\} \tag{6}$$

Key. A key is a set of pairs of attributes and values, $\mathcal{K} \subseteq \mathcal{A} \times \mathcal{V}$, obtained from the context state of the consumer and used to evaluate a rule. Given the set of required attributes for a keyhole, h, a consumer can define a *key* that contains only attributes listed in h. This key is a *restriction* of the user's complete context state function:

$$keyFromKeyhole(u, h) = context(u) \mid_h \tag{7}$$

It is important to note that each key is a subset of the complete context state of the user. Thus, with a slight abuse of notation, the *rule* function can operate over a key, given that it is a subset of a complete context state.

Filter. A PADEC filter is any technique used to degrade the QoI of an endpoint's output. Formally, a filter is a function whose input is the result returned by an endpoint (i.e., an output, O) and whose output is a QoI-degraded (or *filtered*) version of the endpoint's output. Filters must be idempotent: for a given input to the filter, its output should always be the same. Moreover, the QoI degradation can be expressed as a real number:

$$f : O \rightarrow O \times \mathbb{R} \tag{8}$$

Each filter comprises a method used to degrade the QoI, as well as a set of parameters tailored by the provider, that specify how much the QoI is degraded or which information should be obfuscated. Furthermore, each filter also reports the expected QoI degradation of the filtered output:

$$degradation(f) = \mathbb{R}, f \in \mathcal{F} \tag{9}$$

where $\mathcal{F}$ is the set of all filters. This *degradation* value provides a means for the provider and consumer to communicate about the QoI of the result in a datatype- and application-specific way, as the QoI degradation metric is tied to the datatype the filter is able to be used on (e.g., the euclidean distance may be a degradation metric for a filter that works on coordinates, but not for one that works on strings).

Access level. Formally, an access level combines a rule and a filter:

$$\mathcal{AL} \subseteq \mathcal{R} \times \mathcal{F} \tag{10}$$

Each access level also has a public interface. That is, for $al \in \mathcal{AL}$, if r_{al} is the access level's rule and f_{al} is the access level's filter, then the public interface for al is:

$$al_{pub} = (keyhole(r_{al}), degradation(f_{al})) \tag{11}$$

Through this public interface, consumers can select a desired access level based on the combination of the context information they are required to provide (i.e., the keyhole) *and* the QoI they can achieve in return.

Lock. A lock in PADEC is the set of all access levels that a provider defines over an endpoint. Conceptually, locks are the access control entities in PADEC: endpoints are directly protected by user-defined locks, which can contain as many access levels as the provider desires. Formally, the existence of locks is represented as the definition of PADEC's access control function, relating each endpoint to a lock (i.e., multiple access levels):

$$\mathcal{AC} : \mathcal{E} \times \mathcal{U} \nrightarrow \mathcal{AL} \tag{12}$$

Finally, we consider that consumers can construct keys for multiple endpoints. To do so, we define $\mathcal{AL}_{e_u t}$ as the subset of $\mathcal{AC}(e_u)$ a given consumer t is interested in. The consumer can use multiple criteria, based on both QoI degradation and keyhole information, to select some access levels: those with low enough QoI degradation to enable their use case, those that do not contain attributes they are unwilling to share, or even all of them. Based on this, they can build a single key designed to fit all the keyholes in the public interfaces of the access levels in $\mathcal{AL}_{e_u t}$ they are interested in at once:

$$key(t, \mathcal{AL}_{e_u t}) = \bigcup_{al \in \mathcal{AL}_{e_u t}} keyFromKeyhole(t, keyhole(r_{al})) \tag{13}$$

To evaluate a key, the provider must test the key in their access levels in their non-decreasing order by their filter's QoI degradation. Formally, to test a key in a given access level, the provider evaluates the *rule* function associated with the access level. If the function evaluates to 1, access is granted, the provider evaluates the filter function f associated to the access level, and returns the filtered output to the consumer, which finishes the evaluation. However, if the rule evaluates to 0, access at that level is denied, and the next access level should be tested. If no further access levels remain to be tested, access is effectively denied.

4.3. Case study: Mobile Crowd Sensing

To better understand PADEC, we present a case study based on a mobile crowd sensing (MCS) application for a smart city. The MCS application leverages users' sensors and computing power to, first, promote social running among locals, and, second, help tourists identify Points of Interest (POIs) in the city. Concretely, residents can share contextual information, such as the routes they use for running, the time they take to run through it or their presence in one of them, with fellow residents. This information is exposed from the users' smartphone by three endpoints: *getRoutes*, *getBestTimes*, and *getPresence*. On the other hand, to promote tourism, it exposes from residents the POIs they frequent, along with their rating with two endpoints: (*nearByPOIs*), and (*getRatings*). Moreover, PADEC also manages other kinds of contextual information for the consumers of these endpoints, such as their social groups, location or interests.

When users share personal information, privacy is crucial. A malicious party could potentially use the history of a user to track their habits, learn their routine, etc., so this information must be protected. The privacy exposure of the above detailed endpoints can be modulated with PADEC by each user. In this sense, residents can be willing to share their presence with their friends or family, but not with unknown users. Therefore, as Figure 3 shows, a resident can define two access levels: one providing the exact location in the route and another one where it is only shared during the daytime (i.e., it has a degradation value of 0.5). These two access levels have a lock associated detailing two rules: the first one indicates that the exact location can only be consumed by family members, and the second one defines that the degraded one can be consumed by runners in nearby routes. Likewise, users can set up a lock for each of their endpoints,

5. Implementing PADEC

Based on the PADEC model from Section 4, we next present two PADEC implementations: PADEC, which directly implements the formal model with no additional details, and *PFE PADEC*, which makes use of Private Function Evaluation (PFE) techniques to evaluate the rules.[1]

5.1. PADEC implementation

PADEC is implemented in Java. Java was chosen for two main reasons: first, Java is a highly portable language supported by billions of devices,

[1]This implementation is available at `https://bitbucket.org/spilab/padec_theone`

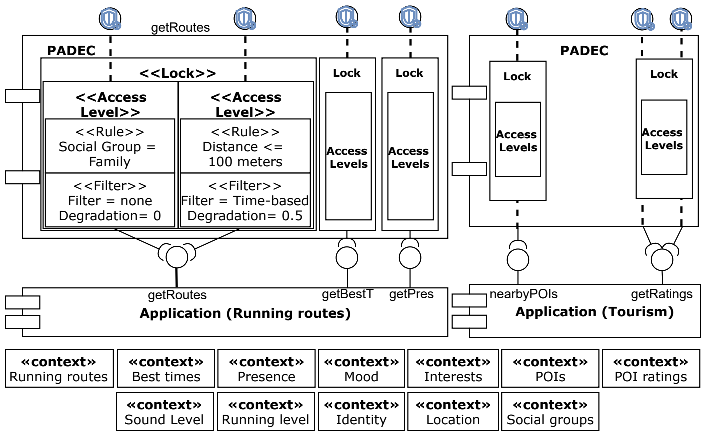


Figure 3: Diagram detailing how PADEC is used in the case study.

such as Android phones or Wear OS devices. The second reason for its use is Java's reflection capabilities, and its concept of interfaces, as well as its ease of integration with other languages. These characteristics fit PADEC's separation of concerns requirements.

Starting from the building blocks, PADEC defines endpoints through the *Endpoint* interface: a generic interface with a single method that takes generic parameters and returns a specific data type. Attributes are implemented as holders for a certain data type that inherit from the *Attribute* class. Therefore, each attribute is free to have its own validity checks, data type, and implementation. As stated above, attributes can be combined using operators (which are elements of $\mathcal{OP}$ and the logical connectors). All of the operators are implemented using interfaces. These operators generate booleans from two given objects (i.e., comparing attributes to values), or from other booleans (i.e., joining together the results of different clauses).

Rules with single clause are implemented by storing the attribute, values and operator. Rules with multiple clauses are created by storing two existing rules and the connecting operator that joins them. Both single and multi-clause rules inherit from the *Rule* class, this implementation allows us to joining any number of clauses, allowing more expressive and concrete rules. In addition, Rule provides a utility method that returns all the attributes required by the rule. This method is specially important for getting the *keyhole* for the rule. This *keyhole* with the names of the attributes is used by PADEC for creating the *key* that may open the keyhole.

Finally, an *AccessLevel* instance combines an endpoint, the rule that must hold in order to grant access, and a filter. Filters can define the data types they can operate with, as well as their QoI degradation as a real numbers. Filters can be parameterized to define their degradation.

PADEC has been implemented to be easily integrated with user-friendly tools. Nonetheless, in the performed evaluations in section 6, the average time for an interaction to be executed using this implementation is of approximately 25 ms, making the overhead of PADEC usage relatively low.

5.2. PFE PADEC

PFE allows entities to share information that are unable to understand, but still able to perform operations over it. PFE PADEC uses this technique for sharing and validating the keys, instead of encrypting them with a key. The PFE PADEC implementation is designed so three constraints hold: the consumer cannot know the logic behind the provider's access control rules, the provider cannot know the value of the consumer's attributes, and the validation results must be identical to those of PADEC.

To perform PFE, this implementation makes use of ABY [17], which, despite being intended as a secure two-party computation framework, serves the purpose of a PFE framework. ABY was chosen because of its relatively low overhead, and its availability as open source software.ABY is fully implemented in C++, and was integrated in PFE PADEC using the Java Native Interface (JNI). ABY's functions are built as *circuits*. By using ABY, consumer and provider are able to compute a common circuit, each with private inputs to it, though the circuit is public to both parties. To maintain the logic secret, PFE PADEC uses circuits that perform every available operation over the data, and have their output connected to a multiplexer. The multiplexer is the output of the circuit, and its output is chosen by a provider's private input. Thus, the consumer cannot know which operations will be outputted, and hence, cannot know the logic behind the rule.

The main difference between PFE and non-PFE PADEC are keys. In non-PFE PADEC, keys contain the values of the required consumer's attributes, exactly mimicking the formal model. While PFE PADEC still maintains the key concept and design from the formal model, the transmission and evaluation of keys is performed differently. In non-PFE PADEC, the key is transmitted by the consumer as a single message, containing the name of each attribute and the value for said attribute. A key is received by the provider, who knows all of its content and can reuse it on multiple access levels. In PFE PADEC, however, the key is transmitted as a series of messages. First, the consumer sends a synchronization message telling the provider which access levels they want to test, in which order, with which attributes, and in which order each of the attributes will be provided. After this message is sent, consumer and provider create the circuit on their side, provide their private inputs (in the case of the consumer, they are sent encrypted to the provider), and execute the circuit. The resulting output is also constrained so only the provider can obtain its value. However, the use of PFE raises the overhead of using PADEC, as the computation time for each interaction with the PFE implementation is, on average, 1.3 seconds, 4 times higher than the non-PFE implementation.

6. Evaluation

In this section, we present an evaluation of PADEC over the case study presented in Section 4.3, including both a simulation of the case study and the results of a conducted survey. The evaluation's objective is twofold: first, it is aimed at comparing PADEC with other access control mechanisms: RBAC, ABAC and DySP-RBAC, along with comparing PADEC and PFE PADEC. Second, it is also aimed at evaluating the usefulness of PADEC's expressiveness for usersIt is important to note that, due to their similarities in this opportunistic scenario, the standard mechanisms RBAC and ABAC are functionally equivalent to other state-of-the-art works. Thus, by comparing PADEC with them, the framework is also compared with Penumbra [20] (functionally equivalent to RBAC), Shafeeq *et al.* [21] (equivalent to ABAC) and SAC [27] (similar to ABAC).

The results of the evaluation are divided into two kinds of results: experimental evaluation (presented in Section 6.2), which comprises results obtained from the simulations using a baseline to compare multiple access control mechanisms, and survey evaluation (presented in Section 6.3), which includes results obtained from the conducted survey and the simulations that use it as a baseline. Before presenting the results, Section 6.1 contains the setup of simulator and the case study.

6.1. Scenario setup

PADEC allows device-to-device interactions within opportunistic networks. Nonetheless, please, note that PADEC is a security and privacy framework at the application level of the communication. Thus, the objective of the evaluation is not to evaluate or test the performance of the opportunistic networkThe simulated opportunistic scenario has been implemented using the tools from TheONE simulator [33], and all the simulations have been executed in a computer with an Intel i7-8565U CPU, with 4 CPU cores, and 16 GB of RAM. Within the simulator, the map of New York City is implemented, as obtained from OpenStreetMap [34], for users (i.e., nodes) to roam. Users are separated into two kinds: tourists and residents. Both kinds of users roam around the NYC map using shortest path map-based movement: they select a point in the map at random, find the shortest path to said point and move there. Once the point is reached, another point is selected randomly. Tourists roam around points from the touristic zones of the city, whereas residents roam around both touristic and residential zones. For each of the scenarios, 100 tourists and 50 residents are simulated. Furthermore, to ensure comparability among the results, the same RNG seeds are used in all the evaluation scenarios. Each of the considered nodes communicates with others using TheONE's SimpleBroadcastInterface, able to transmit data in a 50 meters range, and routes information using the default EpidemicRouter included in TheONE. The simulation lasts for 86000 seconds, which is approximately 24 hours, and each consumer attempts to make a new PADEC service discovery request every 50 seconds, unless the consumer is waiting for the response of a previous request. This opportunistic network environment is used to perform the comparative analyses included in the evaluation, ensuring a fair scenario for all four access control mechanisms. The contact times of the nodes last for an average of 73 seconds, with a standard deviation of 116.32 seconds. Moreover, 90% of the contacts last less than 100 seconds. Focusing on this 90% of the contacts, they follow a normal distribution, with a mean contact time of 44.4 seconds and a standard deviation of 12.8 seconds. Intercontact times do not follow a known distribution, and last on average 12029.56 seconds with a standard deviation of 12519.39 seconds.

The case study described in Section 4.3 has been implemented in the TheONE simulator. The evaluation is performed by measuring statistics of the *nearByPOIs* endpoint, which yield POIs obtained from the New York City POI dataset from [35]. Each of the simulated users is linked with an anonymized user ID in the dataset, and only releases POIs belonging to said user ID. Despite the fact that the case study presents a higher number of endpoints, we decide to measure *nearByPOIs* only. This decision is motivated by a key tenet of PADEC: separation of concerns. Endpoints belong to the application logic side, and hence, PADEC is not dependant on the endpoints or vice-versa.

In these simulations, each type of context has a different category of sensitivity from the consumer's perspective. Higher categories are considered to be more sensitive. The context types we use are: (1) *identity*, which each consumer perceives as the most sensitive attribute (category 3); (2) *location* (category 2); and (3) *sound level* (category 1). We define consumer privacy

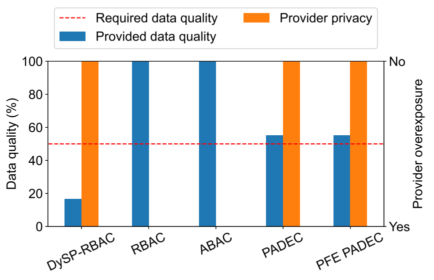


Figure 4: Data quality provided and provider privacy in PADEC

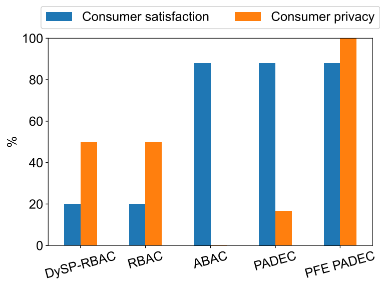


Figure 5: Consumer satisfaction and consumer privacy in PADEC.

as the average sum of the categories of the attributes shared, so that 0% means all attributes are shared and 100% means no attributes are shared.

6.2. Experimental evaluation

The first objective is to evaluate the trade-off between the privacy PADEC offers to providers and the data quality being released, so that consumers are able to get enough data quality but providers are not overly exposed. Moreover, this trade-off is also evaluated on the consumer side by comparing consumer privacy with the amount of consumers that could get their requests fulfilled within a time frameThe next objectives of the evaluation are to compare how well users are able to *hide in the crowd* using PADEC w.r.t. other mechanisms. We also intend to evaluate the number of false positives and negatives in PADEC generated due to the computation and communication overhead compared to alternative mechanisms.

Our first result is related to the number of successfully executed PADEC protocols. Due to the nature of the opportunistic network, any given two nodes may not be permanently connected to each other, as the range of their wireless interfaces is limited. Therefore, two nodes may start a PADEC interaction, but disconnect from each other before it has finished. As a result, in the simulations, a total of 216318 PADEC service discovery requests are initiated, and 216268 of those requests were successfully executed. Thus, PADEC presents a 99.976% of success rate within the simulation, and consequently, we consider the packet loss rate from the application's perspective low enough to provide meaningful results in application evaluation.

Figure 4 compares the data quality offered by providers with the privacy PADEC offers to them in terms of overexposure. To be able to better understand the implications of data quality, the red dashed line in the Figure depicts the data quality required by consumers, which is approximately 50%. In RBAC and ABAC, filters are not implemented, which implies that data is never degraded and, thus, the data quality is always 100%. On the other hand, this means that providers always share 100% of their data, hence being overexposed with no privacy. The implementation of filters in PADEC, as well as DySP-RBAC's restrictions on granularity, allow providers to customize the data quality they reveal, thus eliminating overexposure. This comes at the cost of reducing the average data quality of consumers. In the case of DySP-RBAC, this quality is under the required threshold.This problem is addressed by PADEC, thanks to the fact that consumers are automatically granted access at the highest access level they are allowed to. The raise in data quality does not mean that the data quality improves per-se in PADEC w.r.t. DySP-RBAC, it is a sign that more users are allowed access to higher access levels with higher data quality. Since the PFE integration does not affect data quality or provider privacy, its results are the same as PADEC.

Figure 5 shows the results comparing consumer privacy with consumer satisfaction. Consumer satisfaction is measured as the amount of consumers that got at least a satisfied request within a timeframe, in this case, 3 hours. It is important to note that the timeframe for satisfaction is completely application-dependent. Our baseline is not to have 100% consumer satisfaction, but rather to maximize consumer satisfaction along with consumer privacy, balancing the trade-off between them. With DySP-RBAC and RBAC, while consumers retain a high privacy due to only sharing their identity, a very low number of consumers, namely, those that are explicitly allowed access, can be satisfied. Satisfaction highly rises with ABAC due to a complete context check, with the cost of consumers having their context completely exposed, leading to no consumer privacy. With PADEC, consumers reveal more information than with DySP-RBAC or RBAC, decreasing their privacy, although releasing less information than ABAC. On the other hand, their satisfaction increases to ABAC levels, as their access attempts are always automatically adjusted to the according access levels. Finally, the introduction of PFE maintains satisfaction while guaranteeing 100% consumer privacy, as consumers do not have to reveal any of their contextual attributes any longer.

The consumer's privacy can also be seen as a metric of how easy they are to tell apart from other users. This definition of privacy, inspired by the concept behind differential privacy [36], is reflected as the *indistinguishability* of a key: given a certain key, its indistinguishability is the number of users who would be able to provide the same key. It is not possible to use techniques for differential privacy within PADEC, as differential privacy requires for all individuals to be contained in a dataset, while PADEC reveals a single

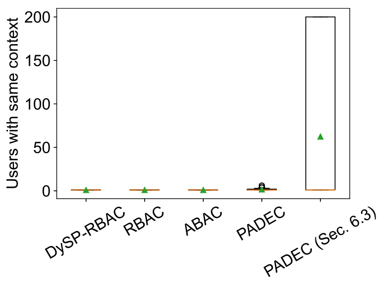


Figure 6: Consumer indistinguishability in PADEC.

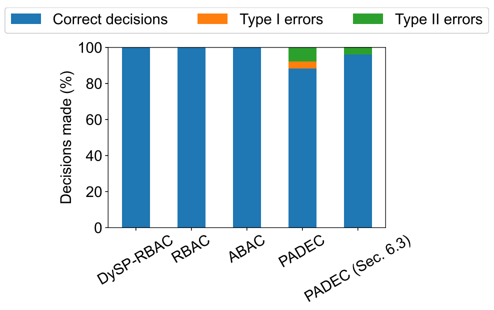


Figure 7: Type I and type II errors in PADEC.

individual's info (i.e., a single data point) at a time. However, an analysis of indistinguishability can be performed in a simulation in which it is possible to obtain the context of all the users. The results of the indistinguishability analysis are provided in Figure 6. In it, we can see DySP-RBAC, RBAC and ABAC have completely distinguishable keys, mainly due to the fact that they carry identity information. Nonetheless, PADEC provides more indistinguishable keys. In average, PADEC keys have an indistinguishability of 1.65 (1.10%), with an interquartile range of 1 (0.67%) to 2 (1.33%). This means that, normally, in our simulation, there is at least another user that is indistinguishable from the consumer. Moreover, we find some cases in which 3 (2%), 4 (2.67%) and even 6 (4%) users have been indistinguishable based on their key information. Thus, we conclude PADEC allows consumers to *hide in the crowd* better than DySP-RBAC, RBAC or ABAC.

An interesting analysis of the implications of the overhead is shown in Figure 7, which depicts the type I (false positive) and type II (false negative) errors for the each access control mechanism. It is important to note that these errors are not due to mistakes in rule evaluation, they are exclusively due to contextual changes over time. In some cases, by the time the rule is evaluated, the consumer's context has changed and, with it, the evaluation of the rule may change. For instance, it is possible that a consumer sends a key with a location that is close enough to the provider to get access, but, by the time the key is received and evaluated, the consumer's location has changed, and may not be close enough to get access. These are the kinds of errors reported in this analysis. We can see DySP-RBAC, RBAC and ABAC have no errors of this kind. Nonetheless, this phenomenon appears on PADEC in 12% of the interactions. In a 9%, the errors are of the false negative kindIn only 3% of the cases, an initially granted access could have been denied if the key was sent afterwards. These errors appear in PADEC due to its temporal and size overhead, as the PADEC handshake takes extra time to be sent and processed.

6.3. User survey

To evaluate PADEC from the user perspective, we have conducted a survey using the Qualtrics service, in which users were asked about their interest in the MCS case study applications, as well as to select and define their own, custom PADEC locks. The source of the survey is publicly available[2]. This survey is aimed at four main objectives: first, to determine if users are interested in context-aware applications. Second, to find out, for one of these applications, which kinds of information users would be willing to share with different groups of people (family, friends, or strangers). Third, out of different pre-made rules defined by users, how many of them could be expressed using different access control mechanisms. Finally, we allow users to define their own rules to protect their information, and evaluate which access control mechanisms would be required to implement them.

On the other hand, we have also performed additional analyses based on the survey: as the surveyees were able to select some pre-made locks or define their own custom locks, they have been implemented in PADEC and used in an additional simulation.Furthermore, both residents and tourists in the survey-based simulations use the state-of-the-art SWIM movement model [37]. In these simulations, contacts last for 2.59 seconds on average, while the standard deviation is of 48.96 seconds. In the case of SWIM, 99.5% of the contacts are below 10 seconds, with an average of 1.44 seconds and a standard deviation 0.84 seconds. Intercontact times do not follow a known distribution, and last an average of 14533.24 seconds with a standard deviation of 14122.40 seconds. In these survey-based simulations, different users apply different rules over each concrete endpoint. Thus, to ensure the fidelity of the results in this scenario, the survey results are measured among all the defined endpoints, rather than a single one.

Therefore, there are six evaluation objectives for the survey: using the results from the survey-based simulations, we analyze the indistinguishability of PADEC users, as well as the type I and II errors. On the other hand, using the survey results themselves, we analyze the interest of users in the proposed applications, the groups of people they would share the information with, and the access control mechanisms that need to be used to express both user-selected and user-made locks.

[2]`http://tiny.cc/padec_survey_source`

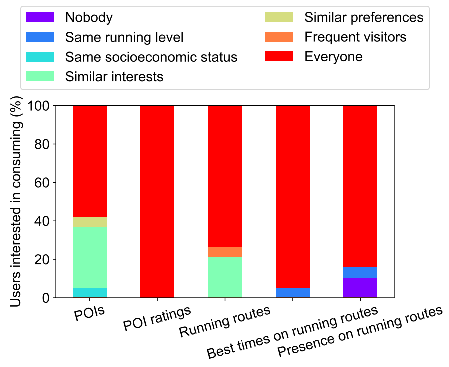


Figure 8: Interest of users on consuming the data of context-aware applications.

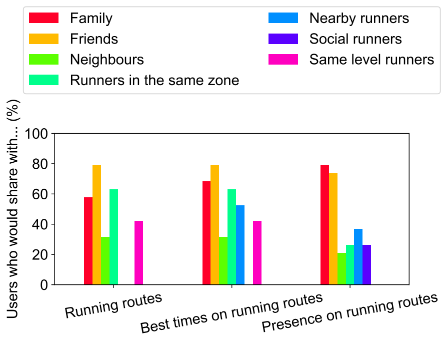


Figure 9: Groups that users would share different types of personal information with.

First, we analyze the indistinguishability of PADEC using the locks from the survey, as depicted in the rightmost box plot in Figure 6. In this case, the rules are very diverse, as the interquartile range from 1 to 200 shows. On average, the keys provided have an indistinguishability of 62.42, which exhibits how PADEC consumers can *hide in the crowd* in such a situation. Continuing with the errors, as shown in Figure 7, in 12% of the interactions, the decision taken by PADEC would have changed. If the user-provided locks are used, however, PADEC exhibits no false positives and a 4% of false negativesIn general, the survey-based simulations show that PADEC provides good results when users are given freedom to create their own locks, rather than sticking to a given baseline.

The third analysis, as depicted in Figure 8, is aimed at assessing whether the users are interested, as consumers, in knowing this information. In general, the vast majority of users are interested in obtaining this information: in the case of POI rating, all users answered this information is useful. Between 95 and 73% of the users also wish to obtain running route information unconditionally, while in the case of POIs other conditions, such as the socioeconomic status, interest or preferences of the producer, are also considered. The main highlight of this analysis, nonetheless, is that all users are interested in the first four types of information, and 89.5% of them are also interested on knowing the presence information.

Continuing with the analysis, Figure 9 shows with whom the users want to share their route-related information with. In the case of running routes, users would often share them with their friends (79%), runners in the same zone (63%), their family (58%), or runners with the same skill level (42%), while some would also share them with their neighbours (32%). While closed

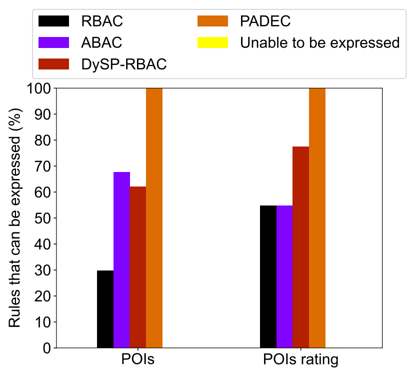


Figure 10: Percentage of pre-made rules selected by users that could be expressed with each access control mechanism.

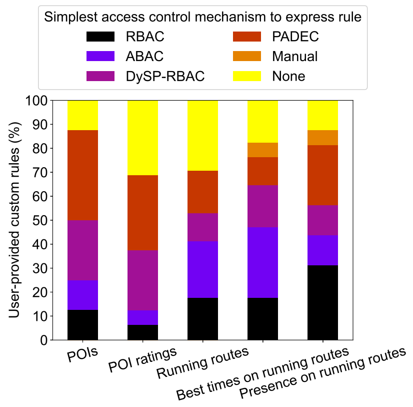


Figure 11: Simplest access control mechanism that can express the custom rules made by users.

social groups, such as family, friends or neighbours, can be expressed easily through classic access control mechanisms such as RBAC, some popular choices include strangers which meet given criteria. Thus, users have a need for more expressive access control mechanisms, such as PADEC. A similar trend can be seen in the rest of information types: most users find acceptable to share their personal information with their closest circles, such as family (68%, 79%) or friends (79%, 73%), but sharing it with strangers that are in the same zone (63%, 21%) or physically close in the moment they consume the information (52%, 37%) are also popular choices.

The next analysis assesses the ability of the analyzed access control mechanisms to express different pre-made rules selected by users. Concretely, for POIs and their rating, users were presented with a variety of pre-made PADEC rules to protect their information, and they had to rate how likely they would be to use each of the rules between 0 and 5. The results shown in Figure 10 depict the access control models needed to express the rules that were rated as 3 or more by users. We find that PADEC is able to express every proposed rule. Furthermore, users were often interested on filtering the information offered to different types of consumers, and hence, DySP-RBAC is able to express 62% of the POI-protecting rules and 77% of the rating-protecting rules. In the case of POIs, users highly rated rules that allowed strangers to access their information under certain circumstances, and thus, ABAC can express 68% of the proposed rules, in contrast with RBAC, which can only express a 30% of them. In the case of ratings, these rules were often tied to filters, and therefore unable to be expressed with

ABAC, which is only able to express the same rules as RBAC (55%).

Finally, Figure 11 shows the simplest access control mechanism to express the custom rules. Out of the four analyzed mechanisms, we consider RBAC to be the simplest, followed by ABAC, DySP-RBAC, and finally PADEC. Moreover, to ensure the fairness of this comparison, we also add two access control models that are even simpler than RBAC: manual, in which the user manually allow or deny each request, and none, in which the information is always granted. Starting with the rules that require no access control, in the cases where users find the information more sensitive, such as POIs or presence, only 12.5% of them chose to share the information freely. On the other hand, users find reasonable to share less sensitive information with everyone, such as POI ratings or running routes, with approximately 30% of the rules allowing for free information sharing. The case of manual choice, however, only exists in best times and route presence, where 5.8% and 6.25% of the rules, respectively, require for such access control mechanism. RBAC and ABAC have similar shares, between 6.25 and 30%. Both mechanisms are the least popular for protecting ratings, while RBAC is the most popular for protecting presence and ABAC for protecting running routes and best times. Rules that require DySP-RBAC are slightly more popular in general, oscillating between 12 and 25% and being a more common choice than RBAC and ABAC for protecting POIs and ratings. Finally, PADEC is the most popular for protecting POIs (38%) and ratings (31%). On average, PADEC is the most popular access control mechanism required to express user-made rules (25%), followed by none (21%), DySP-RBAC (18%), RBAC and ABAC (17% each) and finally manual choice (2%). Furthermore, PADEC is able to express 98% of the user-made rules, only being unable to express those that require manual choices.

7. Conclusions

In the current opportunistic space, users lack mechanisms to constrain the access to their exposed endpoints and to control the privacy of the released information. Furthermore, the nature of opportunistic interactions calls for allowing users to express arbitrary constraints on context for allowing or denying access to protected information. In this paper, we present how PADEC fills this gap by allowing users to express their context-sensitive access control rules, as well as to link them with QoI degradation methods. In the future,we will further analyze the complexity of creating PADEC rules for users, as well as to assist providers by giving them tools to assess the privacy of a certain rule. Finally, we will also enable PADEC to protect the endpoints exposed by applications and schemas that perform in-device computations on demand, such as federated learning, making sure the user is in full control of which circumstances are appropriate to perform certain tasks or share their results.

Acknowledgment

This work was supported by the project PID2021-124054OB-C31 and the grant CAS21/00057 (MCI/AEI/FEDER,UE) by the European Regional Development Fund, by the Department of Economy, Science and Digital Agenda of the Government of Extremadura (GR21133), by the Valhondo Calaff institution, and by the National Science Foundation (CNS-1703497). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the sponsors. Map data copyrighted OpenStreetMap contributors and available from `https://www.openstreetmap.org`.

References

[1] C. Wigginton, M. Curran, C. Brodeur, Global mobile consumer trends, 2nd edition (2020).
URL `https://www2.deloitte.com/content/dam/Deloitte/us/Documents/technology-media-telecommunications/us-global-mobile-consumer-survey-second-edition.pdf`

[2] EMarketer, US adult wearable penetration rate 2016-2022, Tech. rep., EMarketer (2019).
URL `https://www.statista.com/statistics/793800/us-adult-wearable-penetration/`

[3] J. Guillén, et al., People as a service: A mobile-centric model for providing collective sociological profiles, IEEE Software 31 (2) (2014) 48–53.

[4] J. Huang, et al., Blockchain-Based Mobile Crowd Sensing in Industrial Systems, IEEE Transactions on Industrial Informatics 16 (10) (2020) 6553–6563. doi:10.1109/TII.2019.2963728.

[5] M. Abouaroek, K. Ahmad, Foundations of opportunistic networks, Opportunistic Networks: Mobility Models, Protocols, Security, and Privacy (2018).

[6] L. Pelusi, A. Passarella, M. Conti, Opportunistic networking: Data forwarding in disconnected mobile ad hoc networks, IEEE Communications Magazine 44 (11) (2006) 134–141. doi:10.1109/MCOM.2006.248176.

[7] B. Guo, et al., Opportunistic iot: Exploring the harmonious interaction between human and the internet of things, Journal of Network and Computer Applications 36 (6) (2013) 1531 – 1539. doi:https://doi.org/10.1016/j.jnca.2012.12.028.

[8] D. Wu, et al., Security-oriented opportunistic data forwarding in mobile social networks, Future Generation Computer Sys. 87 (2018) 803–815.

[9] Y. Huang, A. Shema, H. Xia, A proposed genome of mobile and situated crowdsourcing and its design implications for encouraging contributions, International Journal of Human-Computer Studies 102 (2017) 69 – 80. doi:https://doi.org/10.1016/j.ijhcs.2016.08.004.

[10] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, P. Steggles, Towards a better understanding of context and context-awareness, in: International symposium on handheld and ubiquitous computing, Springer, 1999, pp. 304–307.

[11] K. Olejnik, et al., Smarper: Context-aware and automatic runtime-permissions for mobile devices, in: 2017 IEEE Symposium on Security and Privacy, 2017, pp. 1058–1076.

[12] H.-C. Chen, Collaboration iot-based rbac with trust evaluation algorithm model for massive iot integrated application, Mobile Networks and Applications 24 (3) (2019) 839–852.

[13] A. K. Malik, et al., A comparison of collaborative access control models, Environment 8 (3) (2017).

[14] D. Brossard, G. Gebel, M. Berg, A systematic approach to implementing abac, in: Proceedings of the 2nd ACM Workshop on Attribute-Based Access Control, 2017, pp. 53–59.

[15] H. Ouechtati, N. B. Azzouna, L. B. Said, Towards a self-adaptive access control middleware for the internet of things, in: 2018 International Conference on Information Networking, 2018, pp. 545–550.

[16] P. Mohassel, S. Sadeghian, How to hide circuits in mpc an efficient framework for private function evaluation, in: Annual International Conference on the Theory and Applications of Cryptographic Techniques, 2013, pp. 557–574.

[17] D. Demmler, T. Schneider, M. Zohner, Aby-a framework for efficient mixed-protocol secure two-party computation., in: NDSS, 2015.

[18] E. Hayashi, et al., CASA: Context-aware scalable authentication, in: Proceedings of the 9th Symposium on Usable Privacy and Security, 2013. doi:10.1145/2501604.2501607.

[19] D. Schurmann, A. Brusch, S. Sigg, L. Wolf, BANDANA - Body area network device-to-device authentication using natural gAit, in: PerCom, 2017, pp. 190–196. arXiv:1612.03472, doi:10.1109/PERCOM.2017.7917865.

[20] M. L. Mazurek, et al., Toward strong, usable access control for shared distributed data, in: 12th USENIX Conference on File and Storage Technologies, 2014, pp. 89–103.

[21] S. Shafeeq, M. Alam, A. Khan, Privacy aware decentralized access control system, Future Generation Computer Systems 101 (2019) 420 – 433. doi:https://doi.org/10.1016/j.future.2019.06.025.

[22] I. C. R. task group, Incits 39-2012: Information technology - role based access control (2012).

[23] Amazon, IAM Roles - AWS Identity and Access Management (2019). URL `https://docs.aws.amazon.com/en{\_}us/IAM/latest/UserGuide/introduction{\_}attribute-based-access-control.html`

[24] Google Cloud, Access control — google kubernetes engine (2022). URL `https://cloud.google.com/kubernetes-engine/docs/concepts/access-control`

[25] Microsoft, What is Azure role-based access control (Azure RBAC)?, Microsoft Docs (2021). URL `https://docs.microsoft.com/en-us/azure/role-based-access-control/overview`

[26] V. C. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, K. Scarfone, NIST Special Publication 800-162 Guide to Attribute Based Access Control (ABAC) Definition and Considerationsdoi:10.6028/NIST.SP.800-162.

[27] M. Sadeghi, L. Sartor, M. Rossi, A semantic-based access control approach for systems of systems, ACM SIGAPP Applied Computing Review 21 (4) (2022) 5–19.

[28] N. A. Mhetre, A. V. Deshpande, P. N. Mahalle, Experience-based access control in ubicomp: A new paradigm, Journal of Computer and Communications 10 (1) (2022) 133–157.

[29] L. Jain, V. Jayesh, Security Analysis of Remote Attestation, CS259 Project Report (Ml) (2011).

[30] E. Bresson, O. Chevassut, D. Pointcheval, Provably secure authenticated group diffie-hellman key exchange, ACM Transactions on Information and System Security 10 (3) (2007) 10–es.

[31] A. McMahon, Discovery of delay-tolerant networking endpoint elements, Ph.D. thesis, Trinity College Dublin (2013).

[32] N. Chomsky, Three models for the description of language, IRE Transactions on Information Theory 2 (3) (1956) 113–124. doi:10.1109/TIT.1956.1056813.

[33] A. Keränen, J. Ott, T. Kärkkäinen, The ONE Simulator for DTN Protocol Evaluation, in: Proceedings of the 2nd International Conference on Simulation Tools and Techniques, 2009.

[34] OpenStreetMap contributors, Planet dump retrieved from https://planet.osm.org , `https://www.openstreetmap.org` (2017).

[35] D. Yang, D. Zhang, V. W. Zheng, Z. Yu, Modeling user activity preference by leveraging user spatial temporal characteristics in LBSNs, IEEE Transactions on Systems, Man, and Cybernetics: Systems 45 (1) (2015) 129–142. doi:10.1109/TSMC.2014.2327053.

[36] C. Dwork, Differential privacy, in: International Colloquium on Automata, Languages, and Programming, Springer, 2006, pp. 1–12.

[37] S. Kosta, A. Mei, J. Stefa, Small world in motion (swim): Modeling communities in ad-hoc mobile networking, in: 2010 IEEE Communications Society SECON, 2010, pp. 1–9.

Appendix B

Continuous QoS-Aware Adaptation of Cloud-IoT Application Placements

ID	: **Computing 2022**
Authors	: Juan Luis Herrera, Javier Berrocal, Stefano Forti, Antonio Brogi, Juan Manuel Murillo
Publication type	: Journal article
Journal	: Computing
Status	: Under review (major revision).
JCR	: Q2 (Computer Science, Information Systems). IF: 2.420.

Continuous QoS-Aware Adaptation of Cloud-IoT Application Placements

Juan Luis Herrera[1*], Javier Berrocal[1], Stefano Forti[2], Antonio Brogi[2] and Juan M. Murillo[1]

[1]University of Extremadura, Spain.
[2]Department of Computer Science, University of Pisa, Italy.

*Corresponding author(s). E-mail(s): jlherrerag@unex.es; Contributing authors: jberolm@unex.es; stefano.forti@unipi.it; antonio.brogi@unipi.it; juanmamu@unex.es;

Abstract

Cloud-Internet of Things computing paradigms call for novel and efficient methodologies to decide where to place application services in continuity with Continuous Integration/Continuous Deployment pipelines and infrastructure monitoring. In this article, we present Continuous Adaptation (CA), a new DevOps practice for (1) detecting runtime changes in the application requirements or the infrastructure that, due to their change in resource consumption or their effects on the Quality of Service (QoS), can affect the validity and dependability of the current application placement, and for (2) locally intervening on them by suggesting new placements that ensure all (functional and non-functional) application requirements are met. We assess a prototype of CA, ConDADO, and analyze its performance over a motivating use case. ConDADO adapts the application placement to environmental changes through the use of continuous reasoning, reducing the size of the problem to be solved to optimize its performance. The evaluation shows that ConDADO is able to obtain nearly optimal QoS up to **4.2×** faster than alternative techniques, also minimizing the cost of service migration.

Keywords: Adaptation, cloud computing, fog computing, edge computing, microservices architecture, Internet of Things, DevOps

1 Introduction

The Internet of Things (IoT) can computerize real-world processes into cyber-physical ones, transforming real inputs and outputs into their digital equivalent. The next generation of IoT applications includes critical and dependable applications, such as in healthcare [1], industry [2], or smart cities [3]. In these applications, the criticality and dependability are reflected by strict Quality of Service (QoS) requirements, which are not trivial to meet. Therefore, we define a dependable application as one where the QoS is sufficient for it to work properly under a given scenario.

In this article, concretely, we focus on the minimization of response times, as performance is a key dependability issue. The cloud used to be the most popular paradigm to deploy IoT applications [4]. Nonetheless, the large distance between final IoT devices and cloud data centers complicates the achievement of these QoS requirements [4]. Thus, recent research is focused on the use of Cloud-IoT continuum paradigms, e.g., edge, or mist computing [4]. Furthermore, these paradigms are often combined with each other, as well as with cloud computing, and as such, the Cloud-IoT continuum epitomizes a multi-paradigm infrastructure. In this multi-paradigm environment, computing nodes that are closer to users can be leveraged to perform some computing tasks, thus reducing the response times from and to the data sources and reinforcing the application's dependability.

In this context, the existence of a wider variety of possible application placements, as well as the changing conditions of the continuum, imply that the management of the deployment and application placement in the Cloud-IoT continuum is also more complicated. For instance, a crashed node, an overloaded server, or network congestion can make an initially optimal application placement not as suitable for the current situation. Indeed, the problem of placing a multi-service application onto Cloud-IoT infrastructures is challenging and interesting for the scientific community, as testified by recent surveys [5, 6]. Solutions based on Mixed-Integer Linear Programming (MILP) have been proposed in the research literature to address this problem. To this end, in our previous work [7], we have proposed the DADO framework, supporting multi-service application placement and replica optimization. Nonetheless, deploying an application following one of these placements is not simple [5].

In cloud environments, deployments are often automatically managed using DevOps practices [8]. Nowadays, Continuous Integration (CI) and Continuous Deployment (CD) have become some of the most well-known practices in DevOps [9]. Companies such as Meta use CI/CD pipelines, coupled with continuous reasoning (i.e., incremental static analyses), to shorten the delivery time of new features [10, 11]. Furthermore, companies are hiring highly specialized system administrators for the management of application deployments in cloud environments, due to the required knowledge and skills [12].

In the Cloud-IoT continuum, the complexity of managing application deployments grows even larger, calling for even more specialized personnel. Nonetheless, the automation of application management in the cloud brought by DevOps would mitigate the management complexity of Cloud-IoT continuum environments, and therefore, DevOps proposals for IoT are already focusing on the continuum [13]. However, the QoS achieved by the original application may change during the whole application lifecycle. Hence, changes in the infrastructure (e.g., changes in the available computing resources due to the deployment of new applications), as well as changes in the app usage (e.g., increases of the user base) or requirements (e.g., library updates that change the resources needed by application components), may also affect whether the QoS constraints are met or not, possibly affecting the dependability of the system. This can reduce the usefulness of CI/CD pipelines because they are neither triggered by QoS degradation due to infrastructural changes nor able to automatically determine a new suitable application placement [13].

Therefore, the deployment of applications through the Cloud-IoT continuum may require the application placement to be optimized and adapted over time to meet the required QoS [5]. In this context, *adapting* the application placement refers to changing the application placement for it to better suit a new or modified environment. Although this problem may arise in multiple other environments, it is in this continuum where the impact is higher, due to its characteristics (e.g., high distribution, large traffic volume, a high number of nodes). Thus, QoS-aware DevOps systems able to identify services unable to meet their requirements due to environmental changes, as well as determining a new, proper placement for them, are desirable. Such systems can automate manual tasks, such as the identification of the problems' causes or the adaptation of the application placement, assisting the specialized personnel.

In this context, existing QoS-aware application placement systems need to become reactive towards environmental changes instead of re-optimizing scenarios from scratch whenever changes occur. Moreover, the systems must be aware of the previous status of the application placement and the environment, giving them a sense of continuity and detecting changes in the system. This awareness of the previous state will allow systems to become reactive to changes, instead of periodically optimizing the application placement. Continuous reasoning techniques were recently proposed to perform this adaptation [14, 15]. Continuous reasoning allows application placement systems to identify the impact of these changes, as well as determine a new placement for the affected services, capable of restoring their correct functioning. Proposals such as FogBrainX [14] use continuous reasoning to determine new potential application placements without, however, considering any optimization metrics (e.g., the response time of the application). To the best of our knowledge, no existing proposals apply continuous reasoning to QoS-optimizing application placement, nor is migration cost considered part of the placement problem. Nonetheless, there is a need for new practices in DevOps to support QoS-aware application placement throughout the Cloud-IoT continuum, as well as for automatic tools that support these practices.

In this work, we propose a new DevOps practice to support application management in continuity with CI/CD pipelines: *Continuous Adaptation*

(CA). It is important not to confuse CA (which is the proposed DevOps practice) with continuous reasoning, which refers to a technique that can be employed by concrete application placement systems.. CA features the incremental, continuous, and QoS-aware *optimization* of next-gen IoT applications and the *adaptation* of their application placement w.r.t. their target objectives (e.g., response time). Furthermore, we present an open-source prototype of a framework to support CA, Continuous DADO (ConDADO), implemented on top of the DADO framework [7]. ConDADO differs from DADO in that it enables incremental, continuous optimization of application placement through service migration, as well as considering the cost of such migrations. On the other hand, DADO is only able to perform optimizations from scratch, unable to know if a service is migrated or not, nor the cost of the suggested migrations. Moreover, DADO is unfeasible for dynamic optimization due to its high optimization times, whereas ConDADO is faster.

The main contributions of this work are: i) the proposal of CA as a new practice in DevOps that can be integrated into CI/CD pipelines to continuously review and adapt an application's placement, ii) the proposal of ConDADO, an enabler to CA that, through the use of continuous reasoning, can adapt an application's placement in a QoS-aware manner, and iii) the evaluation of ConDADO over an IoT facial recognition use case, in which we conclude that ConDADO can adapt application placements to environmental changes in a more efficient and responsive manner than the original DADO.

The rest of this paper is organized as follows: Sect. 2 introduces the motivation for the QoS degradation and dependability problem. Next, Sect. 3 details CA, while Sect. 4 introduces ConDADO. Sect. 5 presents the evaluation. Finally, Sect. 6 presents related works, and Sect. 7 concludes the paper.

2 Motivation

This section describes a use case to better illustrate the challenges of next-gen IoT applications and the need for CA to achieve QoS-awareness. It is important to note that the model for CA we propose abstracts from the concrete sensors, application components, computing devices, and requests behind their technical characteristics (e.g., CPU, RAM, size of data flows). Therefore, while the use case is based on an IoT facial recognition application to provide a concrete point of reference, CA is applicable to a wider variety of contexts, which can include heterogeneous sensors and applications. Moreover, the use case is not meant to be one of the main contributions of the present work.

The use case presented in this section is based on the proposal of Wang *et al.* [3], which present an IoT-based real-time facial recognition application. This application is deployed to a smart university campus. The objective of such an application is to detect any possible intruders that may try to enter the facility without authorization, as well as to ease and automate the check-in and check-out process of authorized personnel. To leverage this application, multiple IoT devices equipped with cameras have been set up at the entrances and exits of the facility. These IoT devices stream the video footage from their cameras directly to the application, which detect their faces, extracts the features of the images, and match them against a registry of known faces [3]. This application is expected to be dependable: it must run at nearly real-time performance, i.e. 20 frames per second [3], a strict QoS requirement that must be satisfied at runtime and over time.

Next-generation IoT applications, as well as modern facial recognition applications, tend to follow a microservices architecture (MSA), a design pattern in which the application consists of multiple loosely-coupled modules or *microservices* that collaborate to carry out complex functionalities [3, 5, 16]. Thus, the application is implemented as a set of four microservices, which are depicted in Fig. 1: i) a face detection and segmentation service that detects if and where faces are in each image frame, ii) a feature extraction service that extracts the features needed for face recognition, iii) a feature matching service that compares a set of extracted features with the existing registry, and iv) a storage service to add new entries to the registry. Each of these microservices can be accessed by IoT devices through the HTTP protocol [16]. Moreover, two main functionalities are supported by the application, the management of authorized users and intruder detection.

In order to perform these functionalities, the IoT devices can request for a *workflow* of these microservices to be executed, in which multiple microservices are pipelined. For instance, to register a new user into the system, an administrator or authorized user can trigger a request from the IoT device for the detection and segmentation microservice to obtain an image that contains exclusively the user's face. Then, the feature extractor is requested and fed the facial image. The resulting features are then added to the registry through the storage microservice. This workflow is depicted by the blue arrows in Fig. 1. On the other hand, to perform intruder detection, the IoT device sends a constant flow of video frames to the application. The video follows a similar pipeline, shown in green in Fig. 1 until the features are obtained. Nonetheless, rather than storing the features, they are fed to the matching microservice instead to recognize the user, as the green line from Fig. 1 represents. The number of requests to these workflows, as well as the computing capabilities and hardware resources of the nodes, affect how many replicas of each of these microservices are deployed, as well as each replica's placement in the Cloud-IoT continuum.

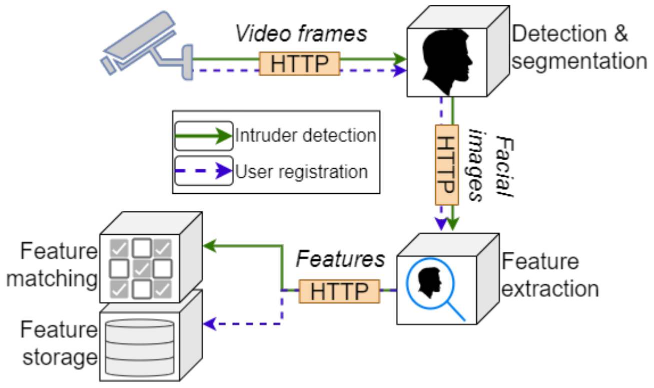


Fig. 1: Architecture of the IoT facial recognition application.

The IoT facial recognition application for this scenario makes use of a DevOps CI/CD pipeline, which follows the schema shown in Fig. 2, which in turn is based on the practices surveyed in [9]. After a planned feature is implemented, the pipeline is triggered by a developer pushing one or more *commits* to the *main* branch of the application repository. Whenever the pipeline is triggered, the CI system pulls the changes to the application and builds the new version [9]. If the build is performed successfully, the test suite for the application runs. The built and tested version of the application is then released to CD, which first executes *acceptance tests* over it, whose role is to ensure the application is production-ready. If the acceptance tests pass, the new build of the application is finally deployed to production, and monitored for additional insights (e.g., using multi-paradigm compatible tools such as FogMon [17]).

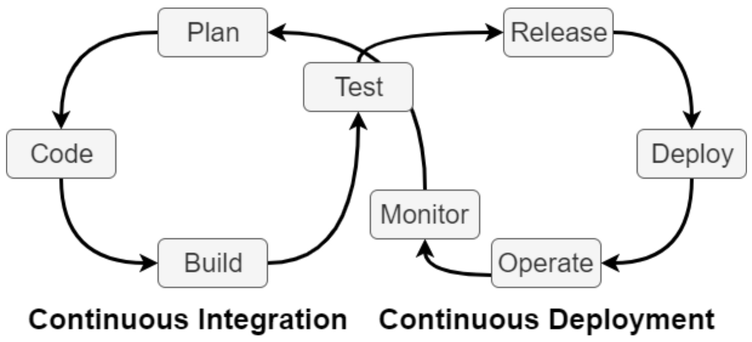


Fig. 2: DevOps CI/CD pipeline.

During the development of the application, in order to ensure the optimal application placement, as well as compliance with the defined QoS requirements, some techniques for design-time application placement optimization, have been proposed, such as [5, 7, 18]. It is important to note, however, that these techniques are meant to be used for the deployment of new releases of the application. Metaphorically, these techniques can only *see a still picture of a live show*: they do not receive information on how the application placement or the environment was prior to that point, and thus, cannot know whether they are suggesting for changes to the application placement or not. Furthermore, the CI/CD pipeline employed by the application is triggered by changes in the code, and not by changes in the infrastructure or the request volume. Hence, the initial deployment, which makes use of this initial application placement, is optimal and dependable, as the QoS requirements of the application are met.

However, most IoT environments, such as this university's infrastructure, are not static environments, and their dynamicity may change the dependability and even the validity of the initial application placement. For instance, in case one of the nodes goes down, or if a commit that updates a library increases the hardware resource requirements of the application, the initial optimization may not hold valid [15]. Furthermore, changes such as a reduction of the available resources due to the unforeseen execution of additional applications may affect the application's QoS, further complicating compliance with the QoS requirements. This motivates the need for a new step within the pipeline, able to be triggered by changes other than those made to the application's codebase and that continuously checks the application placement for the microservices.

This new step of the pipeline requires not only new practices within DevOps but also tools that support them. While one may think that the aforementioned problems can be solved by making use of the same design-time optimization framework, there are two important limitations. On the one hand, these systems often take a long time to yield a solution [7]. While this can be acceptable at design time, during the analyzed events, the application does not properly work until a new application placement is applied, which is unbearable for most applications. Moreover, design-time optimization frameworks do not receive information about the previous status of the environment, they can only *see the still picture*. Nonetheless, an approach to make existing optimization systems *able to see the live show*, and thus suitable for dynamic environments, is to apply *continuous reasoning* [14, 15]. This approach was originally used in large software repositories such as Facebook: each time there is a change in the repository, an analysis is triggered to statically analyze the parts of the codebase that are affected by the change [10]. Within application placement, continuous reasoning is aimed at finding which microservices of the application placement need to be migrated (i.e., moved between nodes) after a change [15]. Continuous reasoning can be triggered not only by changes to the application's code but also by changes in the available infrastructure resources. Continuous reasoning reduces the size of the application placement problem, as only those elements that were meaningfully affected by environmental changes need to be migrated, which may speed up decision-making [7], and allows for the consideration of migration costs. Thus, through the use of continuous reasoning, existing optimization frameworks can be adapted to CA.

To tame the exponential time complexity of optimization frameworks, it is possible to work on two aspects. On the one hand, we should employ more efficient algorithms to determine the new placement for microservices. On the other hand, we should reduce the size of the problem instance. ConDADO follows both lines, by exploiting state-of-the-art MILP solving techniques in combination with continuous reasoning to promptly respond to environmental changes that affect the dependability and validity of the placement of applications through the Cloud-IoT continuum.

3 Continuous Adaptation

To continuously adapt the application placement of next-gen IoT applications to the changes in their dynamic environment, we propose CA, a new practice in DevOps. Unlike other DevOps practices such as CI or CD, CA can be triggered by events that do not directly affect the application's code and check the validity and dependability of the application's placement after the environment changes, appropriately adjusting it if necessary. CA considers the following two kinds of effects that may appear as a result of environmental changes:

Broken deployments are situations in which the existing application placement cannot be used successfully after a change in the environment. For instance, in our running example, the feature extraction deep learning model may be substituted by a more precise one that also consumes more hardware resources, but the application placement may have the microservice running in a device without enough resources to properly run it, or a fog node where some microservices replicas are deployed may fail. In both of these cases, if the application placement is not adapted to this situation, some microservice replicas would be impossible to run, and thus, the deployment is labeled as *broken.* Due to the impact of broken deployments, detecting them is a priority within CA.
QoS violations are subtler than broken deployments and represent dependability issues. In CA, a QoS violation exists whenever, after a change in the environment and if the existing application placement is used, the QoS requirements of the application are not met. While it is possible that the QoS may degrade as a result of these changes in the environment, CA does not deem such situations as QoS violations if the QoS requirements of the application can still be met (i.e., if the application is still dependable enough). Thus, in a QoS violation, the application, or part of it, cannot meet the defined QoS. For instance, the university may need to close one of the entrances of campus to perform maintenance on the area. Therefore, the users that used that entrance to enter or exit the facility will go through other entrances. If that entrance was usually where users were registered, it is likely that no storage microservice exists at other entrances, as replicating the microservice would not improve the response times. However, the existing microservices may not be prepared for the additional burden. Thus, each of the registration requests requires communication with the fog node at the closed entrance. This longer communication increases the workflows' latencies, possibly lowering the performance to less than the required 20 frames per second. The priority of QoS violations is also very high, and thus, they are detected in the same step as broken deployments.

Once either broken deployments or QoS violations are detected, the application placement needs to be adapted to the new environment. For instance, to free some computing resources, some microservice replicas may be migrated from overloaded nodes to others, the microservices from a node that is currently down can be migrated to working nodes, or additional replicas may be deployed to reduce latency. To perform these actions, a new and adapted application placement must be determined, which is the ultimate task of CA. The new application placement must not contain broken deployments or QoS violations, and therefore the effects of QoS generated by the solutions to broken deployments, as well as the validity of the solutions for QoS violations, have to be considered during its determination process. It is also important to consider the migration cost at this point, and thus, when multiple actions can be performed to successfully adapt the application placement, it is preferable to execute those with lower costs.

As a DevOps practice, CA is meant to be integrated with existing CI/CD pipelines. Within this kind of pipeline, CA fits best at the beginning of CD, in a similar manner to continuous reasoning practices such as Facebook Infer fit into CI [10]. In our running example, a new version of the facial recognition application is coded and pushed to the repository. This new version is analyzed for changes using continuous reasoning *à la Facebook Infer*, and the parts affected by the changes in the code are rebuilt and validated using CI. Then, the information about changes to the application and the Cloud-IoT infrastructure is fed to CA, which detects whether a new application placement is required or not using continuous reasoning. Finally, if needed, a new partial application placement for the new release is determined using CA and is executed by CD, as depicted in Fig. 3.

However, for CA to act as intended, environmental changes that do not trigger the standard CI/CD pipeline (e.g., infrastructural changes) need to be able to trigger it as well. Therefore, the infrastructure's monitoring should be able to trigger CA. Furthermore, the source code may not be directly involved in these changes, and hence, the pipeline can be started directly from CA, as CI would not be necessary. The CA process consists of two main steps. First, the current application placement is scanned to find any broken deployments or QoS violations. Then, both broken deployments and QoS violations are reported and solved by determining a new application placement. The QoS-aware application placement is finally fed to CD and then applied to the infrastructure, closing the CI/CA/CD cycle. Regarding other recently proposed DevOps practices, as well as new practices that may be proposed in the future, we consider two different types of integration with CA. On the one hand, practices that maintain QoS outside of their scope, such as continuous testing [19], do not interfere with CA. As such, these practices can be integrated with CA in a seamless manner. On the other hand, practices that have an overlapping scope with CA may also be integrated with it, although not in such a seamless manner. For example, continuous evaluation [20], which evaluates the performance of the application, can be used to give CA a more precise view of the application's QoS. Furthermore, systems such as FOCloud allow for this performance to also be predicted in an explainable manner [21]. In this regard, the practices must be integrated by either modifying or refining CA's inputs or using CA's output. To better understand the integration of CA in DevOps, as well as its inputs and outputs for its integration with other practices, the integration of CA in a basic DevOps pipeline with CI, CA, and CD as practices is described in Tab. 1.

Nonetheless, a key to the CI/CD pipeline is the existence of automated tools that support both practices. Furthermore, tools for the automation of the continuous reasoning process, such as Facebook Infer, were also crucial for its integration within CI. Thus, successfully integrating CA into DevOps and enabling CI/CA/CD pipelines requires a framework that supports CA.

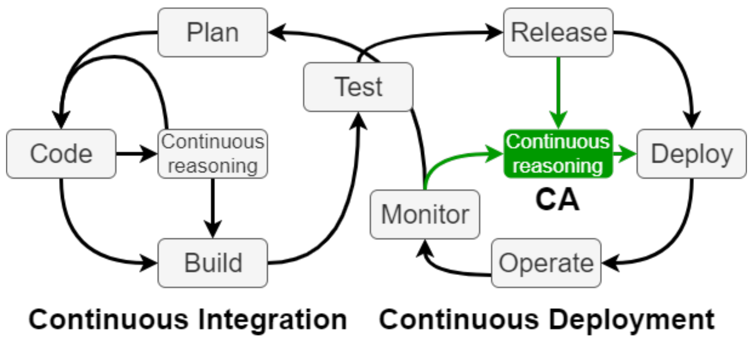


Fig. 3: Integration of CA in the DevOps pipeline.

Practice	Inputs	Description	Outputs
Continuous Integration	Key Performance Indicators (KPIs) of the production environment and requirement backlog	Automated inclusion of the implementation of requirements into the existing codebase, generating binaries for the new version and testing them	New validated application version
Continuous Adaptation	New validated application version or changes in the system's KPIs	Analysis of the validity and QoS of current application placement for the new version or environment and, if required, adaptation thereof	QoS-aware adapted application placement
Continuous Deployment	New validated application version and application placement	Release of the new version installing and maintaining it in the production environment following the application placement, and monitoring its KPIs	Working production environment and associated KPIs

Table 1: Summary of the practices of CI/CA/CD pipeline.

4 The ConDADO framework

In this section, we present ConDADO, a CA framework that applies the concepts of continuous reasoning to DADO. DADO [7] is a framework to optimize the application placement and replication of microservices in the Cloud-IoT continuum based on Mixed-Integer Linear Programming (MILP), a technique that provides optimal solutions at the cost of very high optimization times.

Fig. 4 illustrates the flow of information and processes used by ConDADO. The execution starts when the monitoring system, which is running in parallel to ConDADO, detects a change in the environment, including infrastructure changes or application commits. This monitoring system provides ConDADO with data containing the status of the current scenario, e.g., the QoS and resource requirements of the microservices, the available resources of the nodes, and the workflow requests. The current status data, along with the data of the previous status of the scenario, is fed to the *delta calculator* module. The role of the delta calculator is to quickly assess the differences between both statuses, such as new microservices and workflows, or changes in the existing workflows, microservices, or hardware infrastructure. After such difference is calculated, ConDADO overwrites the previous scenario data with the current one (dashed arrow in Fig.4) to prepare for the next continuous reasoning optimization step.

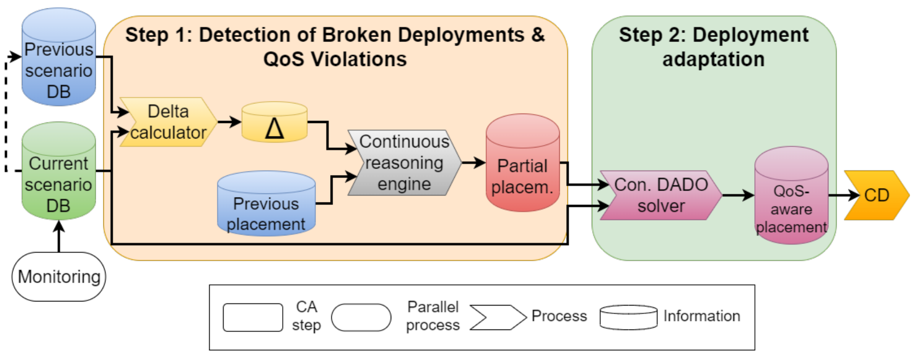


Fig. 4: Bird's eye view of ConDADO.

Once the difference has been calculated, it is fed, along with the previous application placement, to the *continuous reasoning engine (CRE)*, whose logic is described in Algorithm 1. The CRE performs the first step of CA by detecting broken deployments and QoS violations. As the CRE is called, it looks up the hardware elements whose resources have been negatively changed or that are running microservices that have increased in resource consumption (line 6). This includes the hardware whose resources have become zero, i.e., they are down. Then, the CRE calculates if the application placement is still valid in spite of the identified changes (lines 7-14). If it is not the case, the CRE detects the broken deployment (line 10), and selects and clears the placement of microservices (line 12) until the capacity is high enough (lines 10 and 13). These microservices that have their placement cleared are marked as placement decisions that need to be taken again because they are not valid, a process we call *invalidation*. For this process, the CRE invalidates first the placements of the microservices with the lowest migration cost (line 11).

The migration cost of a microservice can be estimated as the size of the package (e.g., Docker image) that needs to be migrated from one machine to another, as heavier microservices require more time to migrate. Nonetheless, the migration cost can be assessed with other criteria, such as the organizational cost of migrating a service (e.g., some services may be costly to migrate because of their criticity for the organization). If multiple microservices have the same migration cost, the CRE invalidates the microservice consuming the most resources first, as it frees more resources to do so, thus minimizing the number of microservices that are invalidated. Finally, the CRE detects QoS violations by listing all the workflows that have not been changed in the delta (line 16), as those that have been changed were already analyzed in the previous loop, and whether any of their placement decisions involves elements whose performance has been deteriorated in the delta (lines 18-21). For these workflows, the CRE calculates whether the performance has worsened enough to be deemed QoS violations (line 19). If so, the CRE invalidates all the application placement decisions of the workflow so they are re-optimized, accounting for the QoS violations that may arise (line 20).

Algorithm 1 Pseudocode for the CRE

```
Require: $P$: previous deployment, $\delta$: differences between the previous and current environment
 1: $invalid := \emptyset$
 2: $Nodes := \{n \| n \in \delta.HW \land (n.resourceDiff < 0 \lor (m \in \delta.SW \land m.resourceDiff > 0 \land on(m,n) \in P))\}$   ▷ Hardware nodes n, where the resources have decreased, or at least a microservice m deployed to it has increased its resource consumption
 3: for all $n \in Nodes$ do
 4:     $M := \{m \| on(m,n) \in P\}$   ▷ All microservices m placed in node n as of P
 5:     $required := \sum_{m \in M} m.resources$
 6:     while $required > P(n).resources$ do   ▷ While the sum of resource requirements of all microservices m placed in n as of P is greater than the resources of n
 7:         $m := extract(M, criteria = (migrationCost, resources))$   ▷ Extract a microservice m from M following the criteria
 8:         $invalid := invalid \cup \{m\}$
 9:         $required := required - m.resources$   ▷ Free up the resources taken by m
10:     end while
11: end for
12: $Workflows := \{wf \| wf \in P.workflows \land \notin \delta.workflows \land (\exists m \in \delta.SW \land m.resourceDiff > 0 \land partOf(m,w))\}$   ▷ Workflows wf that have not been directly modified and request at least a microservice m that has increased its resource consumption
13: for all $wf \in Workflows$ do
14:     $respTime := getRespTime(wf, P)$
15:     if $respTime > wf.minQoS$ then
16:         $invalid := invalid \cup wf.microservices$
17:     end if
18: end for
19: $D := P - invalid$
        return $D$: valid, partial deployment plan derived from $P$, i.e., $D \subseteq P$
```

After the CRE runs, the copy of the previous application placement becomes a list of the placement decisions that have not been invalidated (line 23). We label this list as the *partial application placement.* The final step of the CA pipeline is performed by the *ConDADO solver*[1], which takes as input both the current scenario data, as well as the partial application placement. Based on this information, the solver skips over the parts of the model involving decisions that are in the partial application placement. Skipping these operations speeds up the optimization time, as the problem instance is smaller, and thus, it is faster to generate and solve. After the problem is generated, the same MILP optimizer software leveraged by DADO is used to solve it. Once the QoS-aware application placement is obtained, the new migrations, replications, and updated placements can be easily identified by calculating the delta

[1] The MILP formulation used by DADO is not an original contribution of this paper, and therefore, it is not included. Nonetheless, interested readers can find the detailed formulation in [7].

between the original and the new application placement. Finally, the QoS-aware application placement obtained as a result can be fed to CD, continuing the CI/CA/CD pipeline.

5 Performance evaluation

In this section, the performance of ConDADO is evaluated over multiple conditions of a use case to validate the usefulness of CA in CI/CD pipelines.[2] The evaluation use case, the performed experiments, and the evaluation objectives are detailed in Sect.5.1. The results obtained by each of the experiments are analyzed in Sections 5.2, 5.3, and 5.4.

5.1 Evaluation setup

To evaluate ConDADO, an environment based on the facial recognition application from Sect. 2 has been defined. Following this application, IoT devices with cameras can request two kinds of workflows: facial matching and user registration. While registration workflows are triggered by an administrator, application-wise, they are requests of microservice execution that come from an IoT camera, and hence, they are modeled in this manner.

The placement of this IoT application is evaluated over two different scenario sizes: small, with 5 IoT cameras and a single fog node, and large, with 15 IoT cameras and 3 fog nodes. The IoT cameras are based on Texas Instruments CC2538 microcontrollers, which connect to the network using 6LoWPAN. The network itself is comprised of switches that connect using Gigabit Ethernet, the same technologies fog nodes use for connection. The fog nodes are based on the instances of the EDGE.NETWORK provider, while the cloud node is based on an AWS m5.xlarge instance. Moreover, the networking details such as latencies and bandwidths are also based on these protocols, as well as real metrics. Regarding the number of microservice replicas, they depend on the exact solution given by each solver. Nonetheless, if we account for the maximum number of replicas, up to 15 microservice replicas can exist in the small scenarios, and up to 45 can exist in the large scenarios. Approximately half of the IoT cameras in each scenario request each type of workflow, and in the cases with odd cameras, there is an additional facial matching workflow. In all cases, the scenarios are optimized with DADO first, and then ConDADO is used to adapt the application placement in a QoS-aware manner. All the experiments were executed with an Intel i7-8565U CPU, with 16 GB of RAM. The MILP solver used is Gurobi[3]. Both DADO and ConDADO are implemented in Python using the *mip* library. The numerical QoS-related results are obtained using DADO's QoS analysis tools [7]. On the other hand, the results related to speed-ups and optimization times are not obtained from these simulations, but from the real timings yielded by both DADO and ConDADO during the evaluation.

[2] ConDADO, along with the experiments' code and data is open-source and freely available at https://bitbucket.org/spilab/condado.

[3] https://www.gurobi.com/

Overall, the evaluation has three objectives:

- To validate the usefulness of CA in next-gen IoT applications, consequently validating ConDADO as an enabler for CA (Sect. 5.2)
- To evaluate the differences in the QoS achieved by ConDADO w.r.t. the original DADO (Sect. 5.3), and
- To assess the migration cost and the speed-up of CA solutions compared to the traditional optimization approach (Sect. 5.4).

To achieve these objectives, we have analyzed the results of different scenarios, in order to analyze of the effect of parameters such as the size of the scenario or the impact of the changes. Concretely 20 experiments of two kinds have been tested:

- In the broken deployment experiments, up to 4 microservices and all their replicas experiment an increase of up to 100% in the resources they consume, which may lead to broken deployment situations. These experiments simulate different kinds of changes in the application that could lead to such increases (e.g., the use of more accurate and complex matching techniques, an improvement in video quality, and more detailed video pre-processing).
- In the QoS violation experiments, for cost-effectiveness reasons, we assumed the university increases the resources of their nodes and changes to a less costly cloud provider, which increases the cloud latency by 5 ms.

Both experiments analyze the usefulness of CA by comparing the QoS obtained over time, generating a change in the environment at a given point and comparing the gap between the moment a dependable QoS is obtained with and without CA. In order to use a baseline for the usage of non-CA methods, we assume that the original DADO is used to re-optimize the application placement and that it is triggered immediately (i.e., there is no manual interaction). Thus, it is important to note that the gap is incomparable to the gap caused by manual re-optimization and management. Moreover, in order to better show the effects of the use of CA in terms of migrations, all microservices are considered to have a migration cost of 1.

5.2 CA applicability

The first analysis compares the continuity in the application's QoS in the large topology, in a scenario in which the resources consumed by a microservice are increased by 80%. In this analysis, the environmental change occurs at a given instant t. Due to the increase in consumption, the prior deployment becomes broken, and thus, there is a need to determine an adapted application placement. We consider a deployment becomes broken if its response time is greater than 50 ms, as the QoS requirement for the facial recognition application is to have 20 frames per second. This requirement is represented by the dashed red line labeled *Maximum acceptable response time* in the upcoming figures.

The analysis is depicted in Fig. 5, with the time relative to t shown in the X-axis, and the application's QoS can be seen in the Y-axis. The empty gap represents the application's downtime due to the broken deployment, which is of 27.5 seconds if ConDADO is used to optimize the application placement. On the other hand, the use of DADO requires 76 seconds to complete, a difference that is visually represented as the gap between ConDADO's marker ($*$) and DADO's ($\times$). Furthermore, both ConDADO and DADO achieve the same QoS with their new application placement. The gap between DADO and ConDADO is of 48.466 seconds, i.e., ConDADO is almost three times faster than DADO at suggesting new application placements in this scenario (viz., 2.76$\times$). Moreover, the application placement determined by ConDADO also obtains a similar QoS to that obtained with DADO's application placement, further motivating its use for CA, and providing smoother and faster decision-making.

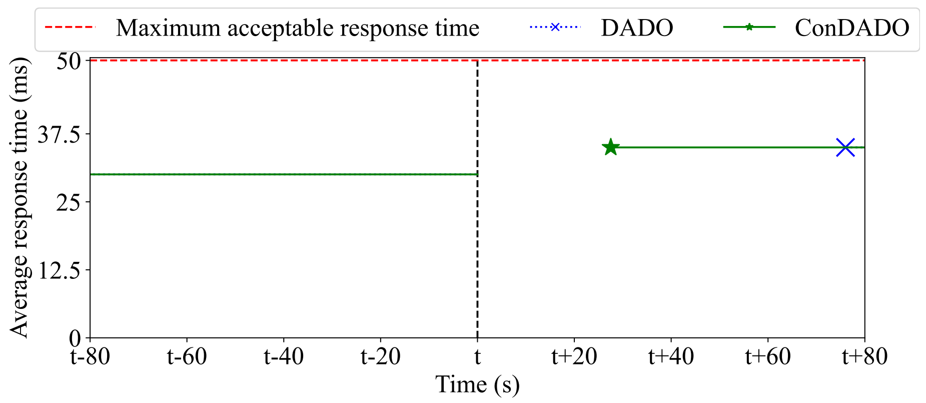


Fig. 5: Continuity in the application's QoS in the large topology.

The analysis depicted in Fig. 6 is aimed at evaluating ConDADO's ability and efficiency in resolving QoS violations. The first change in the topology is an increase in the resources of the nodes, which did not create any QoS violation or broken deployment, and thus, did not trigger CA. Nonetheless, as the latency in the cloud was increased at moment t, the response time quickly increased, triggering CA. It is important to note that CA was triggered at the peak of the increasing curve, as the shape has been interpolated. In the case of ConDADO, this implies that, at the peak, the delta calculator triggered the rest of the pipeline. Once at this peak, ConDADO determined the placement and migrated the services from the cloud to the fog nodes in 27.5 seconds, as depicted by the star marker. Conversely, DADO required over 5 minutes (302 seconds) to perform the same task, as it was not designed to solve QoS violations. Overall, the results show that ConDADO is better suited to adapt to dependability issues such as QoS violations, experiencing a speed-up of approximately 11$\times$.

5.3 QoS with and without CA

The next analysis is aimed at comparing the QoS experienced by the application workflows. The results of this analysis are shown in the form of a boxplot from the large topology, with 4 microservices affected, in Fig. 7. It is important to note that all 18 scenarios involving broken deployments yield very similar

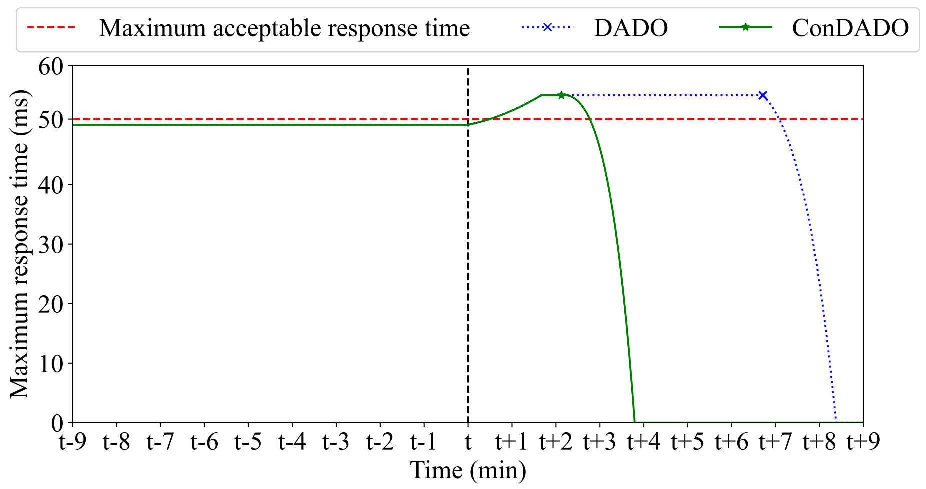


Fig. 6: Evolution of the application's response time in the QoS violation experiment, large topology.

results, and thus, only the three highest load scenarios are depicted. Nonetheless, the interested reader can find the data for the response times of workflows in the additional material of the paper. We can clearly split the workflows into two kinds: those that have all of their microservices in nearby continuum devices (i.e., those with nearly 0 response time), and those that have at least one microservice placed on the cloud (i.e., those with 50 ms response time). In the first case, with the 20% hardware increase, we find that DADO is able to bring a few more microservices closer to the IoT devices, as the median is very close to 0. On the other hand, while ConDADO needs to use the cloud more, as shown by the median, the impact on the average response time (depicted as a triangle) is not very large, with under 10 ms of difference. Continuing to the case with an 80% hardware increase, we see that both DADO and ConDADO yield equal results, i.e., ConDADO is able to find the same optimal solution as DADO. In the final scenario, ConDADO brings most microservices to the cloud, as the lower point is depicted as an outlier. Nonetheless, we see the difference in average response time w.r.t. DADO is also minimal, similar to the first scenario. Thus, ConDADO is able to provide a similar QoS to DADO for the applications that make use of its suggested placement. Furthermore, the small increase in response time is only experienced by a few users (i.e., workflows), rather than by the application as a whole: the median does not change, those points affected are represented as outliers, and the effects over average response time are minimal. Finally, as every workflow request has a response time of 50 ms or less, the performance requirement of 20 frames per second is fulfilled with the optimized application placements.

5.4 Migration cost and speed-ups with and without CA

Another key metric of the evaluation is the migration cost achieved by ConDADO's solutions in contrast to those provided by DADO. This metric has been analyzed in both the small and large topology, as depicted by Fig. 8 and Fig. 9, respectively. Starting with the smaller topology, in the cases with a

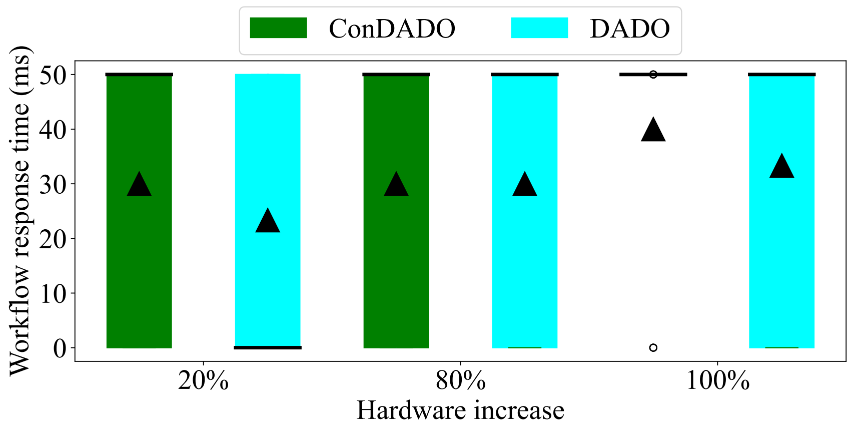
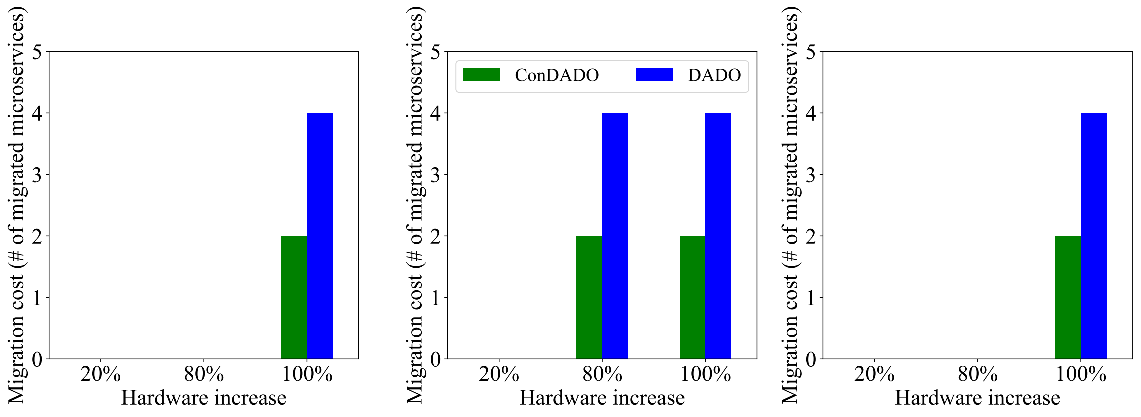


Fig. 7: Boxplot of the workflows' response times, large topology, 4 microservices affected.

single microservice affected (Fig. 8a), we can find that, in some cases (e.g., 20 and 80% hardware requirement increase), the migration cost is 0, i.e., no microservices are migrated. In these cases, the changes due to the hardware requirement increase are mainly a decrease in the number of replicas of certain microservices, and thus, no microservices are strictly *migrated*. Similar results are obtained when 2 (Fig. 8b or 4 (Fig. 8c) microservices are affected.

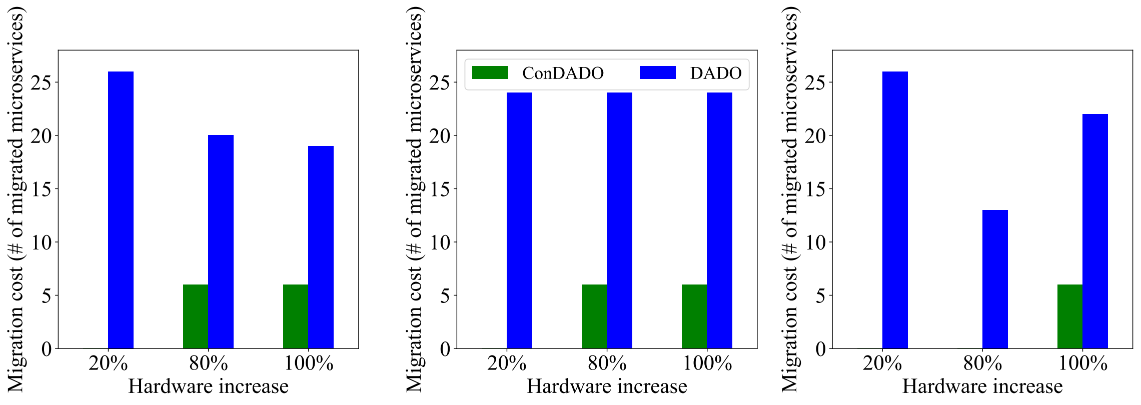
On the other hand, in cases where microservices are migrated (e.g., 100% hardware requirement increase in Fig. 8c), ConDADO's solutions have half the migration cost of DADO's. These results are the consequence of ConDADO's CRE, which tries to migrate a small number of microservices, while DADO's lack of CRE allows it to migrate any number of them. Moving to the large topology (Fig. 9), the reduction of migration cost is even higher. As the larger size of the topology implies a higher number of microservices deployed, DADO is more likely to try and migrate them, while ConDADO follows a similar approach to mitigate the migration cost. Focusing on Fig. 9a, in the case with 20% hardware requirement increase, ConDADO simply removes some replicas, while DADO migrates 26 of them, out of the existing 38. Even when ConDADO needs to migrate microservices, such as the one with the 80% increase, DADO needs to migrate up to threefold the number of microservices. If 2 microservices are affected (Fig. 9b), the results are similar, although in general, the difference is even higher (on average, ConDADO migrates 20 microservices less than DADO). Furthermore, the scenarios from Fig. 9c are also the ones depicted in Fig. 7, and thus, this reduction in migration cost has minimal effects on the QoS experienced by the users of the application.

Next, an analysis that compares the times ConDADO and DADO need to optimize each of the scenarios is also performed. The optimization times for the small topology are reported in Fig. 10, while Fig. 11 depicts those of the large topology. Starting with the smaller topology, we see that the times reported by ConDADO are shorter than those from DADO, with ConDADO taking 0.78 seconds on average, whereas DADO takes 1.38 seconds. Furthermore, the optimization times from ConDADO are not higher than those from DADO in any of the scenarios. If Figs. 10a,10b, and 10c are compared, it is



(a) 1 microservice affected (b) 2 microservices affected (c) 4 microservices affected

Fig. 8: Migration costs of DADO and ConDADO, small topology

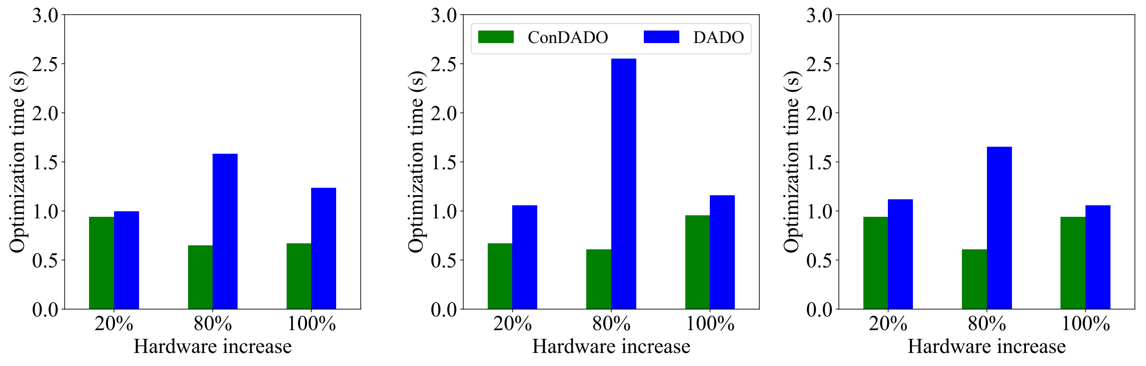


(a) 1 microservice affected (b) 2 microservices affected (c) 4 microservices affected

Fig. 9: Migration costs of DADO and ConDADO, large topology

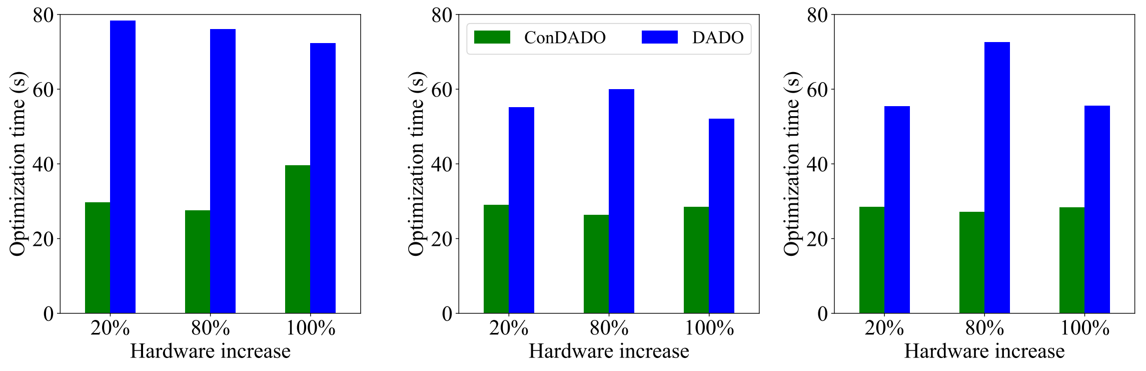
also possible to see that the number of microservices affected does not have an important impact on the optimization times, and the hardware increase does not have a clear impact either. It is also important to note the case with the 80% hardware increase from Fig. 10b, in which DADO takes 2.55 seconds to optimize the scenario, while ConDADO only takes 0.60 seconds.

In the large topology (Fig. 11), the times required by both ConDADO and DADO are higher. Thus, the size of the scenario has an important impact on the optimization time. On average, ConDADO takes 29.41 seconds to optimize, while DADO requires 64.15 to do so. In this topology, the number of microservices affected does have relevance on DADO's optimization times, as the times from Fig. 11a (on average, 75.52 seconds) are higher than those from Figs. 11b and 11c (55.72 and 61.19 seconds, respectively). While this trend is also followed by ConDADO, its impact is much smaller (32 seconds on average in Fig. 11a, 28 seconds on average in Figs. 11b and 11c. Similarly to the smaller topology, the increase in hardware requirements does not have a clear impact on optimization times. Finally, it is important to note that the results



(a) 1 microservice affected (b) 2 microservices affected (c) 4 microservices affected

Fig. 10: Optimization times of DADO and ConDADO, small topology



(a) 1 microservice affected (b) 2 microservices affected (c) 4 microservices affected

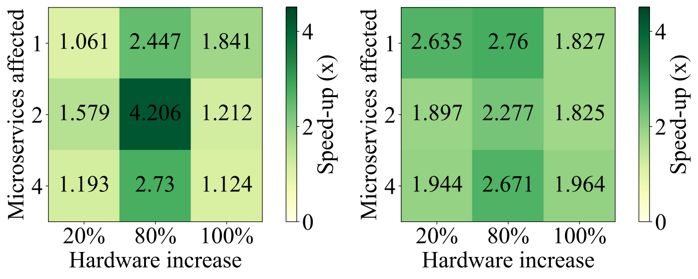
Fig. 11: Optimization times of DADO and ConDADO, large topology

from Fig. 11c correspond with those from Fig. 7, and thus, this reduction in optimization time has very minor effects on the solution's QoS.

Finally, Fig. 12 shows the optimization speed-ups obtained by ConDADO, compared with the original DADO. The reported speed-ups are high, with an average speed-up of 1.93× for the small topology (Fig. 12a) and 2.2× for the large topology (Fig. 12b), and thus, an overall average speed-up of 2.07×. The speed-up tends to grow with infrastructure size due to the fact that ConDADO needs to perform the continuous reasoning step before optimizing, while the original DADO does not. Nevertheless, since the delta calculator and CRE have a smaller complexity than the solver, their burden becomes relatively less significant in larger infrastructures, hence saving more time.

6 Related work

The classic approach to CI/CD pipelines in DevOps literature is to focus on the delivery of software [9]. Therefore, while CD automatically follows a specified application placement, it does so as a means of delivery to production, and not as a means of meeting the application's QoS requirements [9]. While this approach is adequate for cloud-based deployments, this is not the case in the



(a) Small topology (b) Large topology

Fig. 12: Optimization speed-ups of Continuous DADO w.r.t. original DADO

Cloud-IoT continuum, in which dependability is crucial and each node can provide substantially different QoS. In this section, we briefly summarize the state of the art in the Cloud-IoT continuum and IoT-oriented DevOps and QoS-aware application placement.

DevOps has the objective of shortening the delivery time of new features and allowing for quick reactions to client demands. These features are also relevant in next-gen IoT [13]. DevOps practices need to be adapted to this new paradigm. In [13], Lopez *et al.* propose an adaptation of the feedback and monitoring from DevOps to the IoT paradigm and multi-paradigm infrastructures. The proposal specifies how to perform fast and continuous monitoring in IoT applications, and can thus be an enabler for CA. Truong and Klein propose the development of DevOps contracts for IoT microservices in [22]. These contracts define the requirements each of the microservices have and are stored in a blockchain ledger. The application developer can write scripts that will be triggered by microservice deployments, requests, or violations of the contract. The main difference between these contracts and CA is their approach to QoS enforcement: DevOps contracts are imperative, as the IoT application developer needs to implement a method to obtain the desired state; while CA is declarative, and the developer only needs to define the desired state. EU projects, such as SODALITE [23], are also working on approaches towards DevOps-integrated systems for the orchestration of application deployment making use of declarative systems, which could be integrated with CA to enable a more fine-grained detail of the deployed microservices and their scaling. Furthermore, SODALITE@RT [24] allows proposals such as SODALITE to be used in the Cloud-IoT continuum.

On the other hand, the obtention of QoS-aware application placements in the Cloud-IoT continuum is currently an open research topic. In [6], Salaht *et al.* characterize the problem of placing an application's microservices in a dependable way as the Service Placement Problem (SPP). Furthermore, multiple solutions to the SPP are surveyed in this work, including approaches that use different techniques such as integer programming, constrained optimization, or Petri nets. Nonetheless, most of these works are not designed to dynamically adapt the application placement. One of the works that do consider dynamicity is MigCEP [18]. However, MigCEP's model is oriented to migrating complex event processing microservices to adapt the application placement to user mobility. Changes in the infrastructure or the application are not handled by MigCEP, and therefore, it is not suitable for CA.

Another possibility to consider of dynamicity is proposed by Detour [25]. In the Detour model, each request is analyzed before being sent, and an ad-hoc decision is taken on where the request should be executed. This model requires at least one replica of each microservice to be deployed to every node in order to make this ad-hoc decision. Furthermore, the ad-hoc decision process takes a certain amount of time, which delays every request, and may even become inefficient if the delay introduced by the decision process is higher than the obtained speed-up. Other authors, such as Maamar *et al.*, propose a coordination model for the Cloud-IoT continuum [26]. In this model, the data produced by IoT devices are categorized based on multiple attributes. Depending on these attributes, the model recommends the layer or layers they should be processed at. This approach is fundamentally different from ConDADO, as its model is data-centric rather than service-centric, and thus, it does not specify where to place application services nor how to replicate them.

Another system able to determine QoS-aware application placements is the original DADO framework. DADO was proposed by Herrera *et al.* in [7] as an enabler for the obtention of QoS-optimal application placements. This framework was meant to be leveraged during the design phase of next-gen IoT applications. DADO supports a wide array of infrastructures and applications, as well as for the consideration of multiple decisions that affect QoS, such as network latency. However, DADO was not designed to adapt its application placements to dynamic environments, and hence, it is unable to know whether a service is migrated or not, or the cost of such migrations. Furthermore, its high execution times are also unsuitable for this task. Thus, ConDADO is an evolution of DADO, adapting its original concepts for this task.

Finally, the use of continuous reasoning to solve the FAPP was initially proposed by FogBrainX [14], a framework oriented to the obtention of QoS-aware application placements, as well as their adaptation to dynamic environments. FogBrainX makes use of continuous reasoning by determining the microservices that are affected by broken deployments or QoS violations and migrating them to suitable servers. While ConDADO takes inspiration from FogBrainX on the implementation of continuous reasoning, there are multiple differences between the frameworks. The main difference is conceptual, as FogBrainX's model is based on the interactions between microservice replicas, while ConDADO's model is based on requests for microservice workflows, allowing ConDADO to replicate and move microservices if required. Furthermore, FogBrainX does not consider optimization objectives, while ConDADO optimize the response time QoS. Thus, while FogBrainX can be suitable for CA, its objectives and model differ from ConDADO's. We believe that, due to these differences, it is very complicated to fairly compare FogBrainX and ConDADO, as the problems they solve, although similar in nature, are fundamentally different.

7 Concluding remarks

The next generation of IoT applications brings computerization to critical real-world processes, and their continuous improvement will be managed using DevOps and CI/CD pipelines. Nonetheless, the criticality of these processes is reflected in IoT applications as strict QoS requirements. In this context, the use of the multi-paradigm Cloud-IoT continuum requires more complex management. Furthermore, the dynamicity of the environment can threaten its dependability. Thus, these continuous changes on QoS motivate the integration of CA as a practice of DevOps, assuring that the QoS of IoT applications is maintained acceptable by adapting its placement.

In this paper, we presented and motivated the concept of CA as a relevant practice for next-gen IoT applications, including its integration within DevOps in CI/CA/CD pipelines. Finally, we presented ConDADO, a framework to perform CA for microservice-based IoT applications in multi-paradigm infrastructures, such as the Cloud-IoT continuum. In the evaluation, ConDADO was shown to be able to provide dependable and valid application placements with a speed-up of up to 4.2× with regard to alternative solutions. CA provides benefits to developers and operators, as it is a framework for automatic optimization of application placement that can be directly integrated in their DevOps workflow. Moreover, ConDADO's continuous reasoning approach allows for the adaptation of the application placement to be highly reactive due to its speed-up w.r.t. the complete optimization approach, as well as to a reduction of the migration costs of each adaptation.

As future work, we expect to extend CA with prototypes able to adapt other aspects of the application's deployment, such as adapting the devices powered on and off to optimize energy efficiency. Moreover, we also intend to create a multi-objective version of ConDADO, able to consider migration cost as an explicit objective, modeling the tradeoff with techniques similar to cost-benefit analysis [27] or cost-benefit at runtime [28]. Furthermore, the cost will be optimized along with additional QoS metrics such as energy consumption. We also expect to integrate ConDADO with existing deployment orchestrators, i.e., Kubernetes. Moreover, we expect to create even faster solutions for CA and *stack* them on ConDADO, allowing CA to react even more quickly to small changes and easy situations, as well as falling back to ConDADO if a suitable solution cannot be found. Last, but not least, we consider evaluating ConDADO over real or emulated network and Cloud-IoT continuum testbeds.

Conflicts of interest

Not applicable.

Acknowledgments. This work was partly funded by project PID2021-124054OB-C31 (MCI/AEI/FEDER,UE), by the European Regional Development Fund, by the Department of Economy, Science and Digital Agenda of the Government of Extremadura (GR21133), by the Valhondo Calaff institution, and by project *Energy-aware management of software applications in Cloud-IoT ecosystems* (RIC2021PON_A18), funded with FSE REACT-EU resources by the *Italian Ministry of University and Research* through the *PON Ricerca e Innovazione 2014–20.*

References

[1] Greco L, Percannella G, Ritrovato P, et al. Trends in IoT based solutions for health care: Moving AI to the edge. Pattern Recognition Letters. 2020;135:346–353. https://doi.org/10.1016/j.patrec.2020.05.016.

[2] Xu H, Yu W, Griffith D, et al. A Survey on Industrial Internet of Things: A Cyber-Physical Systems Perspective. IEEE Access. 2018;6:78238–78259. https://doi.org/10.1109/ACCESS.2018.2884906.

[3] Wang S, Zafer M, Leung KK. Online placement of multi-component applications in edge computing environments. IEEE Access. 2017;5:2514–2533. https://doi.org/10.1109/ACCESS.2017.2665971.

[4] Bellavista P, Berrocal J, Corradi A, et al. A survey on fog computing for the Internet of Things. Pervasive and Mobile Computing. 2019;52:71 – 99. https://doi.org/10.1016/j.pmcj.2018.12.007.

[5] Brogi A, Forti S, Guerrero C, et al. How to place your apps in the fog: State of the art and open challenges. Software: Practice and Experience. 2020;50(5):719–740. https://doi.org/10.1002/spe.2766.

[6] Salaht FA, Desprez F, Lebre A. An overview of service placement problem in fog and edge computing. ACM Computing Surveys (CSUR). 2020;53(3):1–35. https://doi.org/10.1145/3391196.

[7] Herrera JL, Galán-Jiménez J, Berrocal J, et al. Optimizing the Response Time in SDN-Fog Environments for Time-Strict IoT Applications. IEEE Internet of Things Journal. 2021;https://doi.org/10.1109/JIOT.2021.3077992.

[8] Capizzi A, Distefano S, Mazzara M. From devops to devdataops: Data management in devops processes. In: International Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment. Springer; 2019. p. 52–62. Available from: https://doi.org/10.1007/978-3-030-39306-9_4.

[9] Shahin M, Babar MA, Zhu L. Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. IEEE Access. 2017;5:3909–3943. https://doi.org/10.1109/ACCESS.2017.2685629.

[10] O'Hearn PW. Continuous reasoning: Scaling the impact of formal methods. In: Proceedings of ACM/IEEE Symposium on Logic in Computer Science; 2018. p. 13–25. Available from: https://doi.org/10.1145/3209108.

[11] Calcagno C, Distefano D, Dubreil J, et al. Moving fast with software verification. In: NASA Formal Methods Symposium. Springer; 2015. p. 3–11. Available from: https://doi.org/10.1007/978-3-319-17524-9_1.

[12] Hayajneh S, Hamada M, Aljawarneh S. Project management knowledge areas and skills for managing software and cloud projects: Overcoming challenges. Recent Advances in Computer Science and Communications. 2020;13(3):454–469. https://doi.org/10.2174/2213275912666190429154641.

[13] López-Peña MA, Díaz J, Pérez JE, et al. DevOps for IoT systems: Fast and continuous monitoring feedback of system availability. IEEE Internet of Things Journal. 2020;7(10):10695–10707. https://doi.org/10.1109/JIOT.2020.3012763.

[14] Forti S, Bisicchia G, Brogi A. Declarative continuous reasoning in the cloud-IoT continuum. Journal of Logic and Computation. 2022;32(2):206–232. https://doi.org/10.1093/logcom/exab083.

[15] Forti S, Brogi A. Continuous Reasoning for Managing Next-Gen Distributed Applications. In: ICLP 2020 Tech. Comm.s. vol. 325 of EPTCS; 2020. p. 164–177. Available from: https://doi.org/10.48550/arXiv.2009.10245.

[16] Rojo J, Herrera JL, Moguel E, et al. A microservice architecture for access control based on long-distance facial recognition. In: International Workshop on Gerontechnology. Springer; 2019. p. 219–229. Available from: https://doi.org/10.1007/978-3-030-41494-8_22.

[17] Forti S, Gaglianese M, Brogi A. Lightweight self-organising distributed monitoring of Fog infrastructures. Future Generation Computer Systems. 2021;114:605–618. https://doi.org/10.1016/j.future.2020.08.011.

[18] Ottenwälder B, Koldehofe B, Rothermel K, et al. Migcep: Operator migration for mobility driven distributed complex event processing. In: Proceedings of the 7th ACM DEBS; 2013. p. 183–194. Available from: https://doi.org/10.1145/2488222.2488265.

[19] Angara J, Gutta S, Prasad S. DevOps with continuous testing architecture and its metrics model. In: Recent Findings in Intelligent Computing Techniques. Springer; 2018. p. 271–281. Available from: https://doi.org/10.1007/978-981-10-8633-5_28.

[20] Kao CH. Continuous evaluation for application development on cloud computing environments. In: 2017 International Conference on Applied System Innovation (ICASI); 2017. p. 1457–1460. Available from: https://doi.org/10.1109/ICASI.2017.7988191.

[21] Kumara IP, Ariz M, Chhetri MB, et al. FOCloud: Feature Model Guided Performance Prediction and Explanation for Deployment Configurable Cloud Applications. IEEE Transactions on Services Computing. 2022;https://doi.org/10.1109/TSC.2022.3142853.

[22] Truong HL, Klein P. DevOps contract for assuring execution of IoT microservices in the edge. Internet of Things. 2020;9:100150. https://doi.org/10.1016/j.iot.2019.100150.

[23] Di Nitto E, Gorroñogoitia Cruz J, Kumara I, et al.: Deployment and Operation of Complex Software in Heterogeneous Execution Environments: The SODALITE Approach. Springer Nature. Available from: https://doi.org/10.1007/978-3-031-04961-3.

[24] Kumara I, Mundt P, Tokmakov K, et al. Sodalite@ rt: orchestrating applications on cloud-edge infrastructures. Journal of Grid Computing. 2021;19(3):1–23. https://doi.org/10.1007/s10723-021-09572-0.

[25] Misra S, Saha N. Detour: Dynamic task offloading in software-defined fog for IoT applications. IEEE Journal on Selected Areas in Communications. 2019;37(5):1159–1166. https://doi.org/10.1109/JSAC.2019.2906793.

[26] Maamar Z, Baker T, Faci N, et al. Towards a seamless coordination of cloud and fog: illustration through the internet-of-things. In: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing; 2019. p. 2008–2015. Available from: https://doi.org/10.1145/3297280.3297477.

[27] Gerostathopoulos I, Raibulet C, Alberts E. Assessing Self-Adaptation Strategies Using Cost-Benefit Analysis. In: 2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C). IEEE; 2022. p. 92–95. Available from: https://doi.org/10.1109/ICSA-C54293.2022.00023.

[28] Van Der Donckt MJ, Weyns D, Iftikhar MU, et al. Cost-Benefit Analysis at Runtime for Self-adaptive Systems Applied to an Internet of Things Application. In: ENASE; 2018. p. 478–490. Available from: https://doi.org/10.5220/0006815404780490.

Appendix C

Improving the Energy Efficiency of Software-Defined Networks through the Prediction of Network Configurations

ID	: **Electronics 2022**
Authors	: Manuel Jiménez-Lázaro, Juan Luis Herrera, Javier Berrocal, Jaime Galán-Jiménez
Publication type	: Journal article
Journal	: Electronics
DOI	: `https://doi.org/10.3390/electronics11172739`
JCR	: Q3 (Computer Science, Information Systems). IF: 2.690.

electronics

MDPI

Article

Improving the Energy Efficiency of Software-Defined Networks through the Prediction of Network Configurations

Manuel Jiménez-Lázaro †, Juan Luis Herrera †, Javier Berrocal † and Jaime Galán-Jiménez *,†

Department of Computer Systems and Telematics Engineering, University of Extremadura, 10001 Cáceres, Spain
* Correspondence: jaime@unex.es
† These authors contributed equally to this work.

Abstract: During the last years, huge efforts have been conducted to reduce the Information and Communication Technology (ICT) sector energy consumption due to its impact on the carbon footprint, in particular, the one coming from networking equipment. Although the irruption of programmable and softwarized networks has opened new perspectives to improve the energy-efficient solutions already defined for traditional IP networks, the centralized control of the Software-Defined Networking (SDN) paradigm entails an increase in the time required to compute a change in the network configuration and the corresponding actions to be carried out (e.g., installing/removing rules, putting links to sleep, etc.). In this paper, a Machine Learning solution based on Logistic Regression is proposed to predict energy-efficient network configurations in SDN. This solution does not require executing optimal or heuristic solutions at the SDN controller, which otherwise would result in higher computation times. Experimental results over a realistic network topology show that our solution is able to predict network configurations with a high feasibility (>95%), hence improving the energy savings achieved by a benchmark heuristic based on Genetic Algorithms. Moreover, the time required for computation is reduced by a factor of more than 500,000 times.

Keywords: SDN; machine learning; logistic regression; energy efficiency

check for updates

Citation: Jiménez-Lázaro, M.; Herrera, J.L.; Berrocal, J.; Galán-Jiménez, J. Improving the Energy Efficiency of Software-Defined Networks through the Prediction of Network Configurations. *Electronics* **2022**, *11*, 2739. https://doi.org/10.3390/electronics11172739

Academic Editor: Lionel Nkenyereye

Received: 6 August 2022
Accepted: 28 August 2022
Published: 31 August 2022

1. Introduction

The energy consumption problem in communication networks was one of the most studied problems in the networking area during the 2010–2020 decade due to its negative impact on the environment. The implications of the power consumption generated by the Information and Communication Technology (ICT) sector, and especially by the networking equipment, led researchers in the area to prioritize their efforts to reduce the carbon footprint [1,2].

In recent years, sustainability has gained importance worldwide. Several studies have been published showing the positive impact that ICT solutions may have on sustainability [3,4]. In addition, during the COVID-19 pandemic, it has been demonstrated that ICT solutions, such as remote working, video meetings, etc., have been instrumental in keeping businesses and societies going, thereby proving these solutions for real. Moreover, with the advent of 5G and 6G technologies, in which the goal is to pursue a fully connected intelligent world, efforts must be made to keep alleviating the impact of the emissions generated by the telco sector.

Focusing on the networking area, a first concern is related to the inefficiency of network devices: (i) they are always active regardless of their use and (ii) their power consumption is independent from their traffic load. Moreover, networks are usually designed to avoid congestion during peak traffic periods. In order to reduce the network power consumption, *green networking* [2] aims at making the network consumption load-dependent. Different techniques have been developed in recent years: (i) the use of re-engineering approaches with more energy-efficient technologies such as Energy Efficient

Ethernet—IEEE 802.3az [5]; (ii) the exploitation of dynamic adaptation approaches, in which the modulation of capacities is considered according to traffic load [6]; and (iii) the use of sleeping methods, in which the goal is to use the least number of active devices (links, routers) so that the highest power savings are experienced [7].

The increasing interest in the application of Artificial Intelligence (AI) and Machine Learning (ML) techniques to the networking area [8,9] opens a new niche that accounts for the improvement of the energy efficiency of communication networks. If the traditional energy efficient approaches [1,10,11] are combined with the use of AI/ML techniques, a step further in the improvement of energy efficiency in communication networks will be achieved. Moreover, the central vision, flexibility and programmability of the Software-Defined Networking (SDN) paradigm enable this combination of techniques to re-think and build highly energy-efficient programmable networks [12–14]. Through the implementation of ML algorithms to be run at the SDN controller, control actions in the network may be sped up to modify the network configuration according to the traffic load with the final goal of minimizing the network power consumption [15].

In this work, we investigate how to use ML to target the *H Consumption Problem* (PCP) in the context of computer networks. The main goal of the PCP is to find a feasible network configuration for a given Traffic Matrix (TM) where the number of active network devices is minimal to save the highest amount of energy. In small or medium-sized network topologies, PCP is formulated and solved using Integer Linear Programming (ILP), which outputs an optimal network configuration in terms of energy consumption [16]. However, the PCP is known to be NP-hard, and it is not tractable to find an optimal solution for large networks [16]. In this way, heuristic algorithms are required to obtain (near) optimal solutions in a short period of time. Specifically, we transform the PCP problem into an ML-based classification problem. To that end, the solution to the PCP is provided by a classifier in response to a given TM passed as input.

A Logistic Regression-based Energy Efficient (LR-EE) algorithm is proposed and trained with historical data from a big dataset of TMs. Then, the classifier is able to provide a (near) optimal network configuration for a new (and probably unseen) TM without the need for running the ILP or a heuristic, thus speeding up the process in an online manner. Traffic load thresholds exploited by the SDN controller are considered to run the LR-based ML algorithm in order to potentially change the network configuration for an increase in the energy that is saved.

The main contributions of this work are:

- The definition of a novel algorithm to predict energy-efficient network configurations based on Logistic Regression (LR).
- The evaluation of the proposed LR-based algorithm over a realistic network topology.
- The comparison of the obtained results with energy-efficient ad hoc solutions.

The rest of the paper is organized as follows. Section 2 introduces some related work. A review on the PCP and the heuristic considered to solve it are described in Section 3. Section 4 describes the system model. Section 5 includes the description of the proposed ML algorithm. Experimental results are reported and analyzed in Section 6. Finally, Section 7 draws some conclusions and future works.

2. Research Gap

Over the last few years, extensive work has been conducted on power consumption management to improve the energy efficiency of communication networks. For instance, some works, such as [17–20], focus on reducing the energy consumption of satellite and terrestrial networks while trying to maximize the quality of service. Recent works also focus on SDN networks. The flexibility that these networks provide, such as the separation of the data plane from the control plane and its consequent advantages, creates new opportunities to define more dynamic and energy-aware networks.

Recent works in this area have focused on ILP-based approaches [21,22]. These works allow researchers to identify the optimal network configuration, but they cannot be used for large networks due to the complexity of the problem to address. Since the power consumption problem in SDN can be modeled as a Multi-Commodity Network Flow problem, which is known to be NP-hard, other works focused on applying heuristics, such as Genetic Algorithms (GA), Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO), among others [11,22]. Nevertheless, compared to the dynamicity that SDN networks (time scale of the order of milliseconds) require, these techniques take longer to identify an appropriate network configuration.

In order to reduce the re-configuration time, ML is used to improve network performance. For example, in the field of Traffic Engineering (TE), several works exploit the use of Reinforcement Learning to adapt the network configuration to the current traffic load and minimize the Maximum Link Utilization (MLU) [23,24]. In order to save energy, Ref. [25] proposes an algorithm based on DRL to predict traffic in order to optimize the energy efficiency and perform real-time load balancing. To the best of our knowledge, this is the closest solution to the problem we aim to tackle. In this paper, we focus on using Logistic Regression with the aim of reducing the complexity of the problem addressed. This technique is used to identify the subset of links to be turned off depending the network traffic load.

In the following section, the power consumption problem is studied in order to achieve the goal of this article, bearing in mind the previous works that have just been presented.

3. A Review on the Power Consumption Problem

In this section, a review on the power consumption problem in computer networks is provided. Both the classical ON-OFF model [26] and the one based on the Adaptive Link Rate technique [6,11] are analyzed. Then, a heuristic-based solution is proposed to solve the PCP in tractable time. This heuristic will be used to generate data to feed the proposed ML algorithm for the classification of TMs and the assessment of their corresponding network configuration.

3.1. The Power Consumption Problem (PCP)

PCP is typically modeled as a Multi-Commodity Flow (MCF) problem [16] in which the network is represented by a network graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with $\mathcal{V}$ as the set of nodes and $\mathcal{E}$ as the set of unidirectional links connecting them. Each link $l_{i,j} \in \mathcal{E}$ has a specific capacity of $C_{i,j}$ units to accommodate traffic flows and a power consumption of $p_{i,j}$ Watts if they are active and the classical ON-OFF power consumption model is adopted [26]. In case the Adaptive Link Rate (ALR) approach is considered [6,11], a link $l_{i,j} \in \mathcal{E}$ operating at rate k has a power consumption of $p^k_{i,j}$ Watts; $p_{i,j} = 0$ in case the link is powered off (sleeping). The MCF-PCP requires a traffic matrix $\mathcal{T}$ as input, with the description of the volume of traffic per source-destination flow $f_{i,j}$, represented by $d_{i,j} \in \mathcal{T}$. Considering the network features described above, the PCP aims at finding a network configuration $\mathcal{G}' \subseteq \mathcal{G}$ with the minimum power consumption that respects: (i) link capacity constraints, i.e., the traffic load of each link must be $C_{i,j} \leq 100\%$; and (ii) flow conservation constraints, i.e., the amount of traffic reaching a node must be equal to the volume of traffic leaving such node, excluding the traffic inserted/terminated at that node.

More formally, the Integer Linear Programming (ILP) formulation intended to solve the PCP requires a set of variables that are described as follows (The problem described refers to the classical link switch off problem. In case ALR is adopted, variable $x^k_{i,j}$ must be considered, along with an additional constraint to limit the maximum number of rates, k, per link.):

- $x_{i,j}$ is a binary variable whose value is equal to 1 if the link $l_{i,j}$ is active; 0 if the link is powered off.
- $f_{i,j}^{s,d}$ is a binary variable whose value is equal to 1 if the traffic demand of volume $d_{s,d}$ derived by flow $f_{s,d}$ is routed on the link $l_{i,j}$; 0 otherwise.

  After defining the required variables, the PCP formulation is described by Equations (1)–(3):

$$\min \sum_{l_{i,j} \in \mathcal{E}} x_{i,j} p_{i,j} \tag{1}$$

subject to:

$$\sum_{j \in \mathcal{V}_i^-} f_{i,j}^{s,d} - \sum_{j \in \mathcal{V}_i^+} f_{j,i}^{s,d} = \begin{cases} 1 & \text{if } i = s \\ -1 & \text{if } i = d \\ 0 & \text{if } i \neq s, d \end{cases} \quad \forall i \in \mathcal{V}, d_{i,j} \in \mathcal{T} \tag{2}$$

$$\sum_{d_{s,d} \in \mathcal{T}} f_{i,j}^{s,d} \cdot d_{s,d} \leq x_{i,j} \cdot C_{i,j} \qquad \forall l_{i,j} \in \mathcal{E} \tag{3}$$

Equation (1) aims at minimizing the network power consumption by finding a suitable network configuration that minimizes the number of active links subject to the constraints defined in Equations (2) and (3). Equation (2) describes the classical flow conservation constraints. It imposes that the volume of traffic that reaches a node must be equal to the amount of traffic that passes through it toward the next hop, unless the node is a source or a destination node. Equation (3) represents the link capacity constraint, where the amount of traffic on the link must be, at most, the capacity of the proper link, $C_{i,j}$. Since the ILP formulation falls into the category of MCF problems, which are known as NP-hard problems [16], heuristic solutions are required to validate their benefits over topologies of large size, and in tractable times. In this paper, the Genetic Algorithm (GA)-based solution proposed in [22] is used to find (near) optimal network configurations in terms of power consumption. Then, the output of the GA will be the inputs for our proposed LR-based ML algorithm.

3.2. GA-Based Heuristic for Power Consumption Minimization

In order to find a network configuration that minimizes the network power consumption and satisfies a given TM respecting flow conservation and link capacity constraints, we rely on the use of our GA-based solution [22]. This solution outputs a network configuration close to the optimal in terms of energy savings. In the following, we review the main aspects of the considered approach detailing the definition of the individuals that compose the population, the fitness function, and the considered biological operators (selection, crossover and mutation).

3.2.1. Chromosome Definition

GAs require a population of individuals, namely chromosomes, that represent potential solutions to the PCP. In our case, a chromosome $c \in \mathcal{P}$ represents a potential network configuration as a succession of L genes, where the k-th gene, $g_k \in c$, describes the operational mode $x_{i,j}$ of link $k = l_{i,j} \in \mathcal{E}$:

$$c = \{g_{1,2}; g_{1,3}; \ldots; g_{i,j}; \ldots; g_{N,N-1}\} \tag{4}$$

Thus, for the classical ON-OFF power consumption model, binary variables are considered, i.e., $g_{i,j} = 1$ in case link $l_{i,j}$ is active; $g_{i,j} = 0$ otherwise. If the ALR model is adopted, then K values are considered for each link configuration, i.e., $g_{i,j} = s, s \in [0, K-1]$.

3.2.2. Fitness Function

A fitness function is also required by GAs to evaluate the goodness of each chromosome (solution) of the population. Related to the objective function defined in Equation (1), Equation (5) is applied to each individual in the population. Function $P(c)$ assesses the aggregated power consumption of the network configuration represented by the potential solution c, according to the current operational mode p_k of each link k in the network. The resulting value of the sum is multiplied by θ, which is set to 1 if the network configuration mapped by c is feasible, i.e., the TM can be correctly routed according to the shortest path rule without violating any of the constraints reported in Equations (2) and (3). Otherwise, θ takes a value high enough to penalize the corresponding fitness value to such an unfeasible chromosome.

$$P(c) = \left(\sum_{g_k \in c} g_k \cdot p_k \right) \theta, \quad \forall k \in \mathcal{E} \tag{5}$$

3.2.3. Biological Operators

In order to perform the evolution procedure, GAs apply biologically inspired operators (selection, crossover, mutation) to the individuals in the population. The set of individuals that survive and form part of the next generation in the GA is selected by applying the classical roulette wheel criterion. Moreover, the combination of individuals to generate offspring is performed by means of the single-point cross-over function. Regarding the mutation process, a two-step uniform mutation function is applied. First, a fraction of each individual is selected for mutation. Every gene in this fraction has a probability rate of being mutated. The second step is to replace each selected gene by another valid value. The application of selection, crossover and mutation operators is repeated in each generation of the GA-based algorithm.

4. System Model

As introduced in Section 1, the main goal of this work is to define a model that is able to add energy efficiency features to SDN networks by putting unneeded links to sleep depending on the current traffic load. To achieve this goal, a Logistic Regression-based ML algorithm has been used.

The proposed algorithm is implemented in SDN networks, which are composed of SDN switches that represent the data plane of the network. Moreover, there is a centralized element that has a global view of the network and is in charge of determining the routing logic. This centralized element is the SDN controller and has a global view of the network, composing the control plane. The SDN controller is able to monitor the network elements by exchanging messages with them. These messages are defined by the OpenFlow protocol [27]. They allow the SDN controller to assess the traffic load as well as network statistics to obtain network metrics such as MLU, average delay or packet loss.

Figure 1 represents the whole process that is carried out for the proposed ML algorithm to work in an SDN environment. As can be seen, a 6-node network with 10 links is represented. First ①, the SDN controller estimates a TM based on average data from historical measurements, as in [28]. After having the TM calculated, the ML algorithm (which is installed as an application inside the controller) is executed by passing the TM as input. The ML algorithm, depending on the learning process of its model, will assign a specific configuration to the network, which implies turning off specific links and keeping the rest active. In the case of Figure 1, it can be seen in ② that the output of the ML algorithm application determines whether to turn off 4 out of the 10 links, thus saving 40% energy and still being able to satisfy the estimated traffic demand.

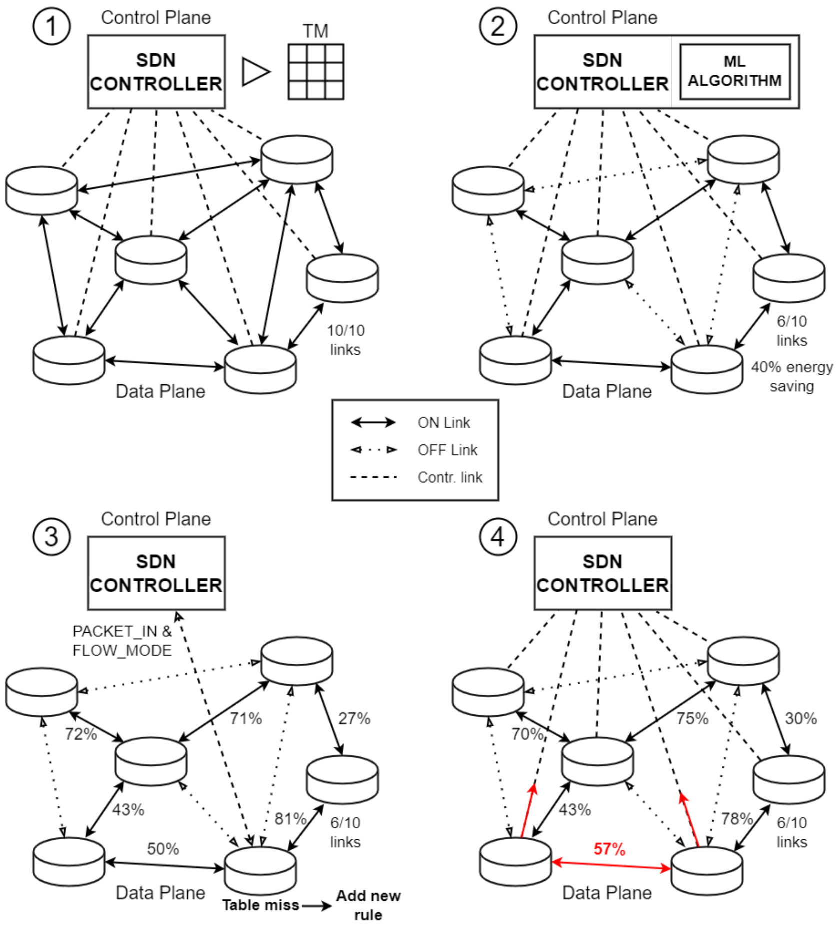


Figure 1. System model overview.

The flow table of each SDN switch stores rules with routing information to carry packets from their source to their destinations. The dynamicity of the network traffic implies that such rules are constantly changing. The routing information will be modified depending on the packets being routed at any given time and the network configuration with the set of active and powered off links. If a flow table does not store the rules that allow a packet intended to be sent to reach its destination, the SDN switch will inform the controller of this situation by sending a PACKET_IN message, as can be seen in ③. With this message, the node is requesting the controller to add, modify or delete a rule in order to route the incoming packet. Thus, the controller replies with a FLOW_MODE message. After each execution of the ML algorithm, the network configuration may change, so new rules must be added, removed or modified, to re-route the packets.

Therefore, since the network configuration is applied after the execution of the ML algorithm, the traffic load of each link may change. Thus, the controller asks the switches about the load of their links from time to time by means of the OFPIT_STAT_TRIGGER message [27]. In our model, it is defined that if the traffic load of a link differs by 5% since the previous configuration was applied, the switches that are connected by the link will notify back to the SDN controller. As can be seen in ④, the load of the red link passed from 50% to 57%, thus triggering the process to notify the controller. At this point, as the controller knows the network traffic load, it is able to assess a new TM, going back to ②, where the ML algorithm is called again with the new TM. Indeed, the ML algorithm will determine the new configuration that fits the new network situation and the process is iteratively repeated.

5. Logistic Regression-Based Energy Efficient Algorithm

In this section, the algorithm that is proposed to predict energy-efficient network configurations based on Logistic Regression (LR-EE) is described. Thus, a methodology to adapt the PCP into a *supervised classification problem* is provided. The aim is to apply a solution based on LR to predict the network configuration associated to a TM and save energy. However, before applying the supervised ML algorithm based on LR, a method for the reduction of the number of classes must be considered. In this sense, a non-supervised ML algorithm is first considered for such a reduction.

5.1. Clustering Process for Network Configurations Reduction

As described in Section 3, each TM is taken as input by the GA results in an associated network configuration. These network configurations indicate which links are active and which ones are put to sleep to save energy, along with a suitable routing for the given TM. In the worst-case scenario, the application of the GA over a set of different TMs may report a set of network configurations that are all different. In the case of thousands of different TMs, thousands of network configurations can be assessed, with an upper bound of $2^{\mathcal{E}}$ due to the binary values of the genes. On the other hand, it may happen that the same network configuration can be applied to a subset of TMs.

Since the main goal of the ML algorithm proposed in this work is to perform a classification of TMs and assign them a valid network configuration able to route the traffic and save energy, a big dataset is required for the training process. However, a first problem related to the number of classes must be tackled. If there are many different network configurations for the prediction, the ML algorithm may not learn correctly due to the ratio between traffic matrices and configurations to be classified. Therefore, its performance may be low.

In order to reduce the number of classes, a non-supervised ML algorithm based on the concept of *K-means* [29] is first applied to group the set of input TMs to ϕ different clusters. The process followed by the algorithm is as follows: each TM belonging to the data set is mapped to one of the ϕ network configurations representing the different clusters. For each cluster, there is only a single network configuration that is valid for all the TMs belonging to the cluster. In this way, a dimensionality reduction is performed:

- If none of the original configurations are valid for all the TMs belonging to that cluster, the configuration with the highest number of active links is selected, and links are iteratively switched on until a valid configuration for all TMs is found.
- If there is an original configuration that is valid for all the TMs in that cluster, it is selected.
- If there is more than one original configuration that is valid for all the TMs in that cluster, the one with the highest number of links off is selected (highest energy savings).

With this approach, we assume a potential gap in terms of energy savings. However, a reduced number of classes will lead to a better classification with the application of the proposed LR-based ML algorithm described next.

5.2. Turning the PCP into a Supervised Classification Problem

In the following, a description of the methodology followed to convert the PCP into a supervised classification problem is provided.

Let us denote $d_{i,j} \in \mathcal{T}$ as the volume of traffic to be sent from node $i \in \mathcal{V}$ to node $j \in \mathcal{V}$, with $i \neq j$. The GA-based solution reports as output a (near) optimal network configuration $C_{\mathcal{T}} = (\mathcal{S}_{\mathcal{T}}, \mathcal{R}_{\mathcal{T}})$ with the status of each link in the network (active or powered off), $\mathcal{S}_{\mathcal{T}} = \{p_{i,j}\}, \forall l_{i,j} \in \mathcal{E}$, and the routing configuration $\mathcal{R}_{\mathcal{T}}$ for each source-destination flow. As an example, let us consider the topology shown in Figure 2a, which is composed of six nodes connected by eight links. One potential solution of the GA is the one depicted by Figure 2b, in which the optimal configuration to route a specific TM $\mathcal{T}_1$ saving the most amount of energy is $C_{\mathcal{T}_1} = (\mathcal{S}_{\mathcal{T}_1}, \mathcal{R}_{\mathcal{T}_1})$. Link configuration is given by Equation (6):

$$\begin{aligned}\mathcal{S}_{\mathcal{T}_1} &= \{s_{1,2}; s_{1,3}; s_{2,4}; s_{2,5}; s_{3,4}; s_{3,5}; s_{4,6}; s_{5,6}\} \\ &= \{0; 1; 1; 0; 1; 1; 1; 0\}\end{aligned} \tag{6}$$

where five links are active ($s_{1,3}; s_{2,4}; s_{3,4}; s_{3,5}; s_{4,6}$) and three links are powered to save energy ($s_{1,2}; s_{2,5}; s_{5,6}$). This leads to 37.5% power savings. The associated routing configuration for all the flows originated by node 1 is shown in Figure 2c. As it can be seen, each flow avoids using the powered off links, which can result in an increase of the traffic volume on specific links such as, e.g., $l_{3,4}$. Thus, the routing configuration is given by Equation (7):

$$\begin{aligned} d_{1,2} &\longrightarrow (l_{1,3}, l_{3,4}, l_{4,2}) \\ d_{1,3} &\longrightarrow (l_{1,3}) \\ d_{1,4} &\longrightarrow (l_{1,3}, l_{3,4}) \\ d_{1,5} &\longrightarrow (l_{1,3}, l_{3,5}) \\ d_{1,6} &\longrightarrow (l_{1,3}, l_{3,4}, l_{4,6}) \\ &\vdots \\ d_{i,j} &\longrightarrow h_{i,j} \\ &\vdots \\ d_{6,5} &\longrightarrow (l_{4,6}, l_{3,4}, l_{3,5}) \end{aligned} \tag{7}$$

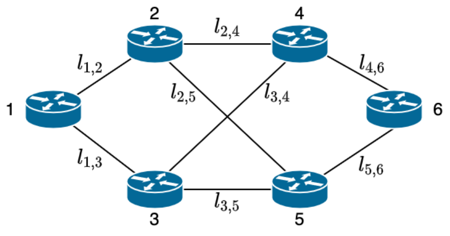
where the path for traffic demand $d_{i,j} \in \mathcal{T}_1$ according to routing configuration $\mathcal{R}_{\mathcal{T}_1}$ is $h_{i,j}$. For instance, the flow originated at node one destined to node six must traverse links $l_{1,3}$, $l_{3,4}$ and $l_{4,6}$. Thus, the supervised classification problem receives as input the serialized TM $\mathcal{T}_1$ with the set of demands to be routed and provides as output a near optimal network configuration $\mathcal{C}_{\mathcal{T}_1}$. Generally, the execution of the GA outputs a specific network configuration. In the worst-case scenario, this can result in a situation in which each TM is associated with a different network configuration. However, in practical scenarios, many of the configurations can be applied to a set of the considered TMs (and not only to one TM), thus reducing the space of configurations under consideration. Therefore, if the number of network configurations to classify the TMs with is reduced, the complexity of the classification problem is lower.

In order to turn the PCP into a supervised classification problem, we consider the input variables as the elements of a serialized TM, with a tuple t_k of type:

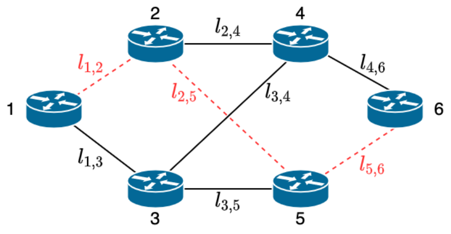
$$t_k = \{d_{1,2}, d_{1,3}, ..., d_{i,j}, ..., d_{N,N-1}\}, \forall i, j \in \mathcal{V} \tag{8}$$

and the output labels are the associated network configurations $\mathcal{C}_{\mathcal{T}}$ to such tuple t_k (TM) obtained by the GA. This dataset can then be used to train a classical supervised ML algorithm based on Logistic Regression. With a big enough number of tuples $(t_k, \mathcal{C}_{\mathcal{T}})$, the trained ML algorithm would be able to predict a network configuration for a given TM without the need of executing the GA-based solution. Moreover, for a new TM not considered in the training set, the ML model should be able to generalize and produce a valid network configuration able to save as much energy as possible close to the one that would be obtained by the GA.

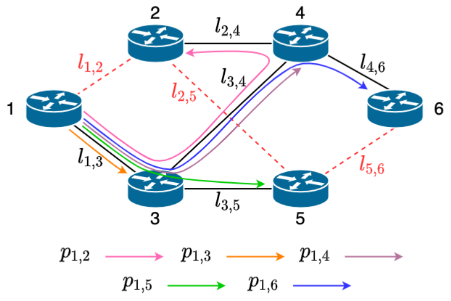
Then, in order to classify new and potentially unseen TMs, the RL-based ML algorithm is invoked and returns a specific network configuration that is expected to (i) be valid, and (ii) able to save energy. With the application of such an algorithm by the SDN controller, the GA is no longer needed and the computation time will be reduced.



(**a**) Network topology



(**b**) GA output. Links configuration



(**c**) GA output. Routing configuration

Figure 2. Example of GA output on a 6-node topology. (**a**) Network topology. (**b**) GA output. Links configuration. (**c**) GA output. Routing configuration.

6. Experimental Results

In this section, an experimental evaluation of the proposed solution is provided. At first, the simulation environment is described. Next, a performance evaluation of the proposed solution in terms of ML and network metrics is carried out to analyze its benefits and potential drawbacks, along with the energy savings achieved.

6.1. Simulation Set-Up

In order to evaluate the effectiveness of the proposed ML solution, Abilene topology (12 nodes and 30 links) is considered. A set of 3871 TMs retrieved from [30] are taken as input, with a time granularity of 5 minutes. Link capacities are set as follows. First, we select the peak TM in the dataset and route it over a set of shortest paths derived after the application of the Dijkstra algorithm on the network graph. After this step, each link $l_{i,j}$ is carrying an amount of traffic $t_{i,j}$. Then, we assume that the capacity of each link can be upgraded by installing a set of line cards. A line card has a capacity of Δ_C equal to $0.5 \max\limits_{l_{i,j}}(t_{i,j})$, i.e., the half of the traffic carried by the link with highest link utilization. Finally, we consider installing the minimum number of line cards needed by each link to make their utilization not greater than 100%, i.e., MLU = 1.

Regarding the power consumption model that is considered, we assume that links can be either powered off (sleeping) or powered on (active). All the links in the network have the same power consumption when they are active ($p_{i,j} = 1$). It is worth remarking that, although this assumption is unrealistic (there can be different types of links with different values of power consumption [11]), it is a classical approach followed in a big set of works tackling the power consumption problem in computer networks [7,26,31].

6.2. Performance Evaluation

As previously introduced, the GA-based solution outputs a (near) optimal network configuration for each TM passed as input. Then, in case Abilene topology is considered, 3871 network configurations are assessed, one per TM. In order to find a suitable number of clusters to group all the TMs, a prior analysis has been carried out. Figure 3 shows the score after the application of the K-means algorithm as a function of the number of clusters. The score is the average of the inverses of the distances of each configuration from its centroid configuration. Clearly, the score is reduced with the number of clusters, following a logarithmic pattern. In particular, a sharp increase is experienced in the range $\phi = [1, 50]$, and the line is stabilized with a large number of clusters. In order to evaluate the impact of the cluster size on the effectiveness of the proposed solution, four values of ϕ are considered in the following performance analyses: $\phi = \{10, 50, 100, 200\}$. Furthermore, in order to carry out the training and testing of LR-EE, the set of TMs has been divided into two subsets. One for training (66% of the data), and the other for evaluating the performance of the algorithm (33% of the data).

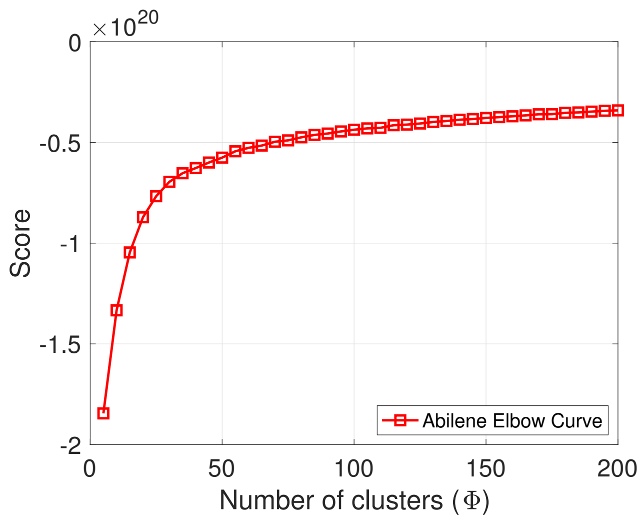


Figure 3. Score of K-means vs. number of clusters (ϕ).

Tables 1 and 2 report the results obtained by LR-EE for the different values of ϕ for comparison with the GA-based solution. Two different types of metrics are shown: (i) ML metrics, with precision, recall and F1-score for each model (Table 1); and (ii) network metrics such as the average link load, MLU, average number of hops per flow, maximum number of hops per flow, average energy saving gap (which is assessed as the energy savings of the original configuration outputted by the GA minus the energy savings of the network configuration of the cluster) and the feasibility, which is the percentage of predictions that, whether or not they are correctly predicted, are valid for the associated TM (Table 2).

Table 1. Machine Learning metrics for LR-EE and GA.

	ML Metrics			**Computation Time**	
	Precision	**Recall**	**F1-Score**	**Train. T**	**Exec. T**
LR-EE $\phi = 10$	0.84	0.84	0.84	7.67 s	1.5 µs
LR-EE $\phi = 50$	0.76	0.76	0.75	3.10 s	1.9 µs
LR-EE $\phi = 100$	0.75	0.74	0.73	3.20 s	2.2 µs
LR-EE $\phi = 200$	0.74	0.72	0.71	5.41 s	4.2 µs
GA	-	-	-	-	2.21 s

Table 2. Network metrics for LR-EE and GA.

	Network Metrics					
	max_LL	**avg_LL**	**avg_hops**	**max_hops**	**avg_gap**	**Feasibility**
LR-EE $\phi = 10$	0.26	0.99	3.60	10	11.25%	97.25%
LR-EE $\phi = 50$	0.42	1	4.50	11	−3.90%	97.80%
LR-EE $\phi = 100$	0.45	1	4.65	11	−5.96%	97.10%
LR-EE $\phi = 200$	0.47	1	4.76	11	−6.95%	95.53%
GA [22]	0.45	1	4.14	11	-	100%

Results show that as the number of clusters (ϕ) increases, the ML metrics perform worse. This is due to the fact that if there is a high number of classes to associate the TMs with, it will be more difficult for the LR algorithm to determine to which class a TM belongs. This is partly because there are classes that have very few associated TMs, while other classes have hundreds of associated matrices. However, this is not reflected in the network metrics. The higher the number of clusters, the better the average energy saving gap is. This is because the configuration associated with each cluster can be more specific to the associated TMs, resulting in a higher number of powered off links. As a result, links are more loaded and flows require a higher number of hops to reach their destination. In fact, for $\phi = 10$, ML prediction is 10% better compared to the case of $\phi = 200$. However, worse outcomes in the energy savings are obtained for a small number of clusters, e.g., $\phi = 10$, where an average gap of 11.25% compared to the GA is obtained. On the contrary, significantly better results are obtained for the case of $\phi = 200$, where 6.59% of energy is saved, on average, when it is compared with the GA. Moreover, the feasibility remains stable above 95% for all tests. It is therefore appropriate to select the option that saves the highest amount of energy, i.e., $\phi = 200$. Finally, it can be seen that the reduction in execution time of the ML-based solution compared to GA is notable. The LR-EE configuration prediction time has an acceleration factor between 526,190 and 1,473,333 times higher than the time needed by GA to generate a new configuration.

Figure 4 reports the power savings achieved by the GA (Figure 4a) and by LR-EE (Figure 4b–e) for different values of ϕ as a function of the TM Id. Note that TMs are sorted according to their traffic load in ascending order. Clearly, it can be seen that higher power savings are achieved when the number of clusters is high (compare $\phi = 10$ clusters of Figure 4b with $\phi = 200$ clusters of Figure 4e). As soon as we increase the number of clusters, the density of the bars in the figures is higher, meaning that the power savings are increased for most of the TMs.

Finally, Figure 5 reports the gap of the GA compared with LR-EE and 200 clusters. It can be seen that LR-EE outperforms the GA in terms of power savings (negative values) for the majority of the TMs. As a summary, the LR-EE proposed solution is able to obtain better power saving outcomes with respect to the GA for a number of clusters above 10 (see *avg_gap* column of Table 1), with the corresponding reduction in the computation time.

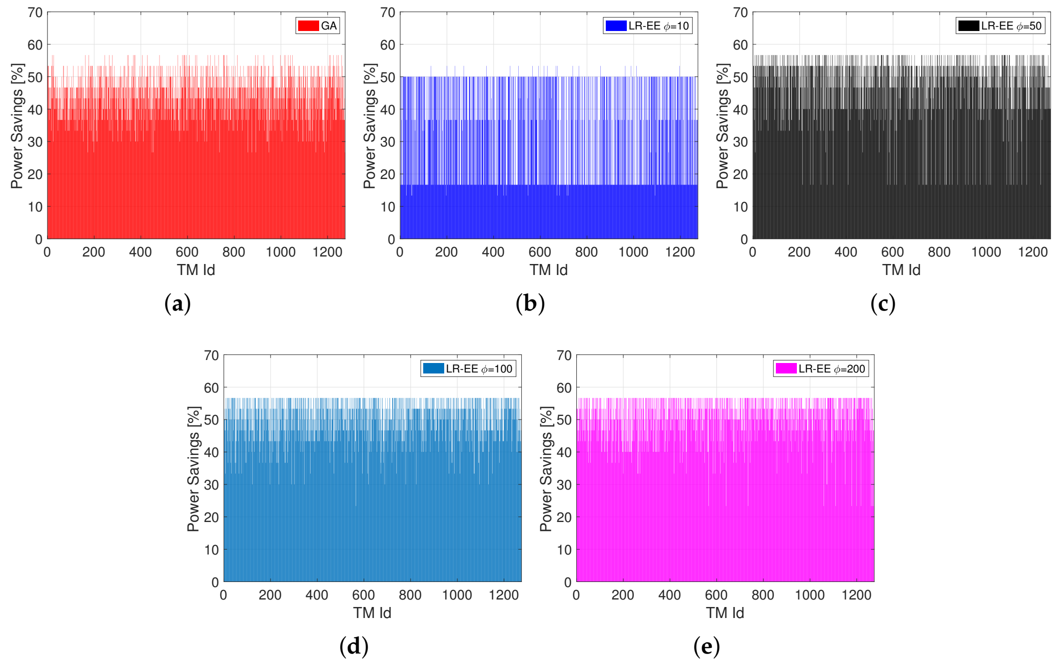


Figure 4. Power savings as a function of the TM Id for GA and LR-EE (TMs are sorted according to their traffic load in ascending order). (**a**) GA [22], (**b**) LR-EE $\phi = 10$, (**c**) LR-EE $\phi = 50$, (**d**) LR-EE $\phi = 100$, (**e**) LR-EE $\phi = 200$.

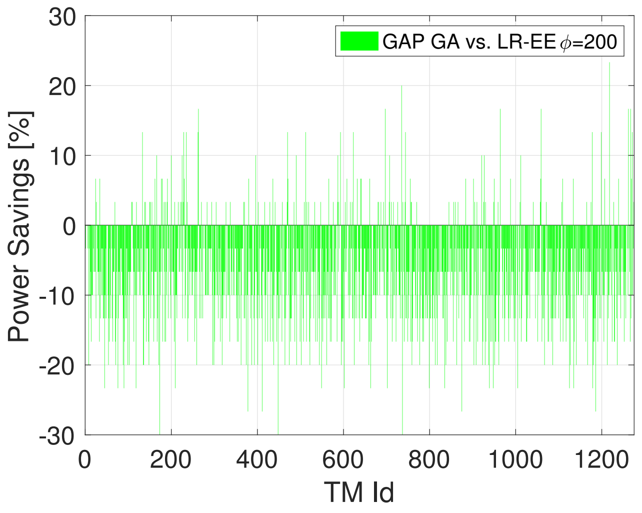


Figure 5. Power saving GAP of GA vs. LR-EE with $\phi = 200$.

7. Conclusions and Future Work

In this paper, an ML solution based on LR has been proposed to predict energy-efficient network configurations in SDN, avoiding the execution of optimal or heuristic algorithms at the SDN controller. Thus, a reduction in the computation time derived by the non-execution of these algorithms at the controller is set as objective, along with the improvement in energy savings. Experimental results over a realistic network topology show that, by applying the combination of unsupervised and supervised learning techniques, a notable reduction of power consumption can be achieved by our proposed LR-EE solution compared to the results obtained by energy-efficient ad hoc solutions, with the corresponding significant reduction in the computation time.

Regarding possible future research activities, we work on testing different ML methods to compare them and select the one that provides the best results in terms of power savings and computation time. We also work on defining a framework that can select the ML technique to use depending on the network topology and other characteristics. In this sense, we are working on using different cluster configuration depending on the network behavior. Finally, we are evaluating the proposed algorithm with larger networks to evaluate the scalability of the solution and how it behaves in emulated environments.

Author Contributions: This work was developed by the authors as follows: Conceptualization, J.L.H., J.B. and J.G.-J.; Data curation, J.L.H.; Formal analysis, J.L.H. and J.G.-J.; Funding acquisition, J.B. and J.G.-J.; Investigation, M.J.-L., J.L.H., J.B. and J.G.-J.; Methodology, M.J.-L., J.L.H., J.B. and J.G.-J.; Project administration, J.G.-J.; Software, M.J.-L.; Supervision, J.G.-J.; Validation, M.J.-L. and J.L.H.; Visualization, M.J.-L. and J.G.-J.; Writing—original draft, M.J.-L., J.B. and J.G.-J. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been partially funded by the Diputación de Cáceres (AV–7), the project PID2021-124054OB-C31 (MCI/AEI/FEDER, UE), 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 (GR21133), and by the European Regional Development Fund.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Idzikowski, F.; Chiaraviglio, L.; Cianfrani, A.; López Vizcaíno, J.; Polverini, M.; Ye, Y. A Survey on Energy-Aware Design and Operation of Core Networks. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 1453–1499. [CrossRef]
2. Bianzino, A.P.; Chaudet, C.; Rossi, D.; Rougier, J.L. A Survey of Green Networking Research. *IEEE Commun. Surv. Tutor.* **2012**, *14*, 3–20. Available online: https://www.researchgate.net/publication/50238130_A_Survey_of_Green_Networking_Research (accessed on 25 August 2022). [CrossRef]
3. Strielkowski, W.; Firsova, I.; Lukashenko, I.; Raudeliūnienė, J.; Tvaronavičienė, M. Effective Management of Energy Consumption during the COVID-19 Pandemic: The Role of ICT Solutions. *Energies* **2021**, *14*, 893. [CrossRef]
4. Rume, T.; Islam, S.D.U. Environmental effects of COVID-19 pandemic and potential strategies of sustainability. *Heliyon* **2020**, *6*, e04965. [CrossRef] [PubMed]
5. Cenedese, A.; Tramarin, F.; Vitturi, S. An Energy Efficient Ethernet Strategy Based on Traffic Prediction and Shaping. *IEEE Trans. Commun.* **2017**, *65*, 270–282. [CrossRef]
6. Nedevschi, S.; Popa, L.; Iannaccone, G.; Ratnasamy, S.; Wetherall, D. Reducing Network Energy Consumption via Sleeping and Rate-Adaptation. In Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation, NSDI'08, San Francisco, CA, USA, 16–18 April 2008; USENIX Association: Berkeley, CA, USA, 2008; pp. 323–336.
7. Galán-Jiménez, J.; Gazo-Cervero, A. ELEE: Energy Levels-Energy Efficiency Tradeoff in Wired Communication Networks. *IEEE Commun. Lett.* **2013**, *17*, 166–168. [CrossRef]
8. Musumeci, F.; Rottondi, C.; Nag, A.; Macaluso, I.; Zibar, D.; Ruffini, M.; Tornatore, M. An Overview on Application of Machine Learning Techniques in Optical Networks. *IEEE Commun. Surv. Tutor.* **2019**, *21*, 1383–1408. [CrossRef]
9. Boutaba, R.; Salahuddin, M.A.; Limam, N.; Ayoubi, S.; Shahriar, N.; Solano, F.E.; Rendón, O.M.C. A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities. *J. Internet Serv. Appl.* **2018**, *9*, 16. [CrossRef]
10. Assefa, B.G.; Özkasap, O. A survey of energy efficiency in SDN: Software-based methods and optimization models. *J. Netw. Comput. Appl.* **2019**, *137*, 127–143. [CrossRef]
11. Galán-Jiménez, J.; Gazo-Cervero, A. Using bio-inspired algorithms for energy levels assessment in energy efficient wired communication networks. *J. Netw. Comput. Appl.* **2014**, *37*, 171–185. [CrossRef]
12. Ahmad, S.; Jamil, F.; Ali, A.; Khan, E.; Ibrahim, M.; Whangbo, T. Effectively Handling Network Congestion and Load Balancing in Software-Defined Networking. *CMC-Tech. Sci. Press* **2021**, *70*, 1363–1379. [CrossRef]
13. Ali, J.; Roh, B.h.; Lee, S. QoS improvement with an optimum controller selection for software-defined networks. *PLoS ONE* **2019**, *14*, e0217631. [CrossRef]
14. Ali, J.; Roh, B. Quality of service improvement with optimal software-defined networking controller and control plane clustering. *Comput. Mater. Contin* **2021**, *67*, 849–875. [CrossRef]
15. Xie, J.; Yu, F.R.; Huang, T.; Xie, R.; Liu, J.; Wang, C.; Liu, Y. A Survey of Machine Learning Techniques Applied to Software Defined Networking (SDN): Research Issues and Challenges. *IEEE Commun. Surv. Tutor.* **2019**, *21*, 393–430. [CrossRef]
16. Chiaraviglio, L.; Mellia, M.; Neri, F. Reducing Power Consumption in Backbone Networks. In Proceedings of the 2009 IEEE International Conference on Communications, Dresden, Germany, 14–18 June 2009; pp. 1–6. [CrossRef]
17. Lin, Z.; Niu, H.; An, K.; Wang, Y.; Zheng, G.; Chatzinotas, S.; Hu, Y. Refracting RIS-Aided Hybrid Satellite-Terrestrial Relay Networks: Joint Beamforming Design and Optimization. *IEEE Trans. Aerosp. Electron. Syst.* **2022**, *58*, 3717–3724. [CrossRef]

18. Lin, Z.; An, K.; Niu, H.; Hu, Y.; Chatzinotas, S.; Zheng, G.; Wang, J. SLNR-based Secure Energy Efficient Beamforming in Multibeam Satellite Systems. *IEEE Trans. Aerosp. Electron. Syst.* **2022**, 1–4. [CrossRef]
19. Lin, Z.; Lin, M.; Wang, J.B.; de Cola, T.; Wang, J. Joint Beamforming and Power Allocation for Satellite-Terrestrial Integrated Networks With Non-Orthogonal Multiple Access. *IEEE J. Sel. Top. Signal Process.* **2019**, *13*, 657–670. [CrossRef]
20. Lin, Z.; Lin, M.; de Cola, T.; Wang, J.B.; Zhu, W.P.; Cheng, J. Supporting IoT With Rate-Splitting Multiple Access in Satellite and Aerial-Integrated Networks. *IEEE Internet Things J.* **2021**, *8*, 11123–11134. [CrossRef]
21. Naeem, F.; Tariq, M.; Poor, H.V. SDN-Enabled Energy-Efficient Routing Optimization Framework for Industrial Internet of Things. *IEEE Trans. Ind. Inform.* **2021**, *17*, 5660–5667. [CrossRef]
22. Galán-Jiménez, J.; Polverini, M.; Cianfrani, A. Reducing the reconfiguration cost of flow tables in energy-efficient Software-Defined Networks. *Comput. Commun.* **2018**, *128*, 95–105. [CrossRef]
23. Guo, Y.; Wang, W.; Zhang, H.; Guo, W.; Wang, Z.; Tian, Y.; Yin, X.; Wu, J. Traffic Engineering in Hybrid Software Defined Network via Reinforcement Learning. *J. Netw. Comput. Appl.* **2021**, *189*, 103116. [CrossRef]
24. Guo, Y.; Chen, J.; Huang, K.; Wu, J. A Deep Reinforcement Learning Approach for Deploying SDN Switches in ISP Networks from the Perspective of Traffic Engineering. In Proceedings of the 2022 IEEE 23rd International Conference on High Performance Switching and Routing (HPSR), Taicang, Jiangsu, China, 6–8 June 2022; pp. 195–200. [CrossRef]
25. Chen, X.; Wang, X.; Yi, B.; He, Q.; Huang, M. Deep Learning-Based Traffic Prediction for Energy Efficiency Optimization in Software-Defined Networking. *IEEE Syst. J.* **2021**, *15*, 5583–5594. [CrossRef]
26. Chabarek, J.; Sommers, J.; Barford, P.; Estan, C.; Tsiang, D.; Wright, S. Power Awareness in Network Design and Routing. In Proceedings of the IEEE INFOCOM 2008—The 27th Conference on Computer Communications, Phoenix, AZ, USA, 13–18 April 2008; pp. 457–465. [CrossRef]
27. OpenFlow Switch Specification. Version 1.5.1, Standard, Open Networking Foundation. 2015. Available online: https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf (accessed on 25 August 2022).
28. Polverini, M.; Baiocchi, A.; Cianfrani, A.; Iacovazzi, A.; Listanti, M. The Power of SDN to Improve the Estimation of the ISP Traffic Matrix Through the Flow Spread Concept. *IEEE J. Sel. Areas Commun.* **2016**, *34*, 1904–1913. [CrossRef]
29. Han, J.; Kamber, M.; Tung, A. Spatial Clustering Methods in Data Mining: A Survey. *Data Min. Knowl. Discov.—DATAMINE* **2001**. Available online: https://www.researchgate.net/publication/238687113_Spatial_clustering_methods_in_data_mining_a_survey (accessed on 25 August 2022).
30. Orlowski, S.; Pióro, M.; Tomaszewski, A.; Wessäly, R. SNDlib 1.0–Survivable Network Design Library. In Proceedings of the 3rd International Network Optimization Conference (INOC 2007), Spa, Belgium, 22–25 April 2007. Available online: http://sndlib.zib.de (accessed on 25 August 2022).
31. Phillips, C.; Gazo-Cervero, A.; Galán-Jiménez, J.; Chen, X. Pro-active energy management for Wide Area Networks. In Proceedings of the IET International Conference on Communication Technology and Application (ICCTA 2011), Beijing, China, 14–16 October 2011; pp. 317–322.

Appendix D

Joint Energy Efficiency and Load Balancing Optimization in Hybrid IP/SDN Networks

ID	: **Annals 2022**
Authors	: Jaime Galán-Jiménez, Marco Polverini, Francesco G Lavacca, Juan Luis Herrera, Javier Berrocal
Publication type	: Journal article
Journal	: Annals of Telecommunications
DOI	: `https://doi.org/10.1007/s12243-022-00921-y`
JCR	: Q3 (Telecommunications). IF: 1.901.

Joint Energy Efficiency and Load Balancing Optimization in hybrid IP/SDN networks

Jaime Galán-Jiménez[1*], Marco Polverini[2], Francesco G. Lavacca[2], Juan Luis Herrera[1] and Javier Berrocal[1]

[1]Dpt. of Computer Systems and Telematics Engineering, University of Extremadura, Avda. de la Universidad, S/N, Cáceres, 10003, Spain.

[2]DIET Department, University of Rome "Sapienza", Via Eudossiana, 18, Rome, 00184, Italy.

*Corresponding author(s). E-mail(s): jaime@unex.es; Contributing authors: marco.polverini@uniroma1.it; francescogiacinto.lavacca@uniroma1.it; jlherrerag@unex.es; jberolm@unex.es;

Abstract

Software-Defined Networking (SDN) is a paradigm that provides flexibility and programmability to computer networks. By introducing SDN nodes in a legacy IP network topology, network operators can benefit on higher control over the infrastructure. However, this migration is not a fast or straightforward process. Furthermore, to provide an adequate quality of service in hybrid IP/SDN networks, the coordination of buth IP and SDN paradigm is fundamental. In this paper, this coordination is used to solve two optimization problems that are typically solved separately: i) traffic load balancing and ii) power consumption minimization. Each of these problems has opposing objectives, and thus, their joint consideration implies striking a balance between them. Therefore, this paper proposes the Hybrid Spreading Load Algorithm (HSLA) heuristic that jointly face the problems of balancing traffic by minimizing link utilization and network's power consumption in a hybrid IP/SDN network. HSLA is evaluated over differently sized topologies using different methods to select which nodes are migrated from IP to SDN. These evaluations reveal that alternative approaches that only address one of the objectives are outperformed by HSLA.

Keywords: Hybrid IP/SDN, load balancing, energy efficiency, traffic engineering, heuristic

1 Introduction

The internet has drastically changed how our society creates, shares, and consumes information. As a consequence, the volume of traffic flowing through the internet increases exponentially over time, as the number of users continues rising from 3.9 billion in 2018 to the predicted 5.3 billion by 2023 [1], generating an estimate of 2.6 Exabytes of traffic daily [2].

It is because of this growth in traffic demand that, to address peak traffic loads, as well as to react to events that degrade network performance, such as link failures [3] or router malfunctions [4], internet service providers (ISPs) need to reinforce the existing network infrastructure. Such an objective can be obtained by overprovisioning existing resources or by providing additional network redundancy. However, the use of such techniques leads to high energy consumption within normal network operations because all the network equipment is powered on, regardless of the traffic flowing through it [5].

Nonetheless, the flexibility and the programmability provided by the software-defined networking (SDN) paradigm, through the global network knowledge obtained by the controller, enables the proposal of solutions that optimize the amount of network equipment (e.g., links and nodes) that need to be powered on to successfully fulfill the quality of service (QoS) requirements [6–8]. These solutions adapt the network configuration to the traffic demand and improve the network's energy efficiency [9]. Furthermore, the use of SDN allows for the application of additional solutions, such as traffic engineering (TE) techniques. SDN enables the network to design optimal policies for routing and, therefore, flow forwarding, thus performing traffic load balancing at the network level [10–12].

To make use of the SDN paradigm in a network infrastructure, the network equipment needs to be replaced with SDN-compatible nodes. However, such migration is not a straightforward or fast process, and it is not feasible to migrate large network topologies in the short term (e.g., labeling a *flag day* to migrate from IP to SDN), especially due to the economic costs of the replacement and the lack of provision of Service Level Agreement (SLA)-included services to the ISP's customers during the migration. A solution to this problem, commonly used by ISPs due to its cost-effectiveness, is to migrate incrementally, replacing only a subset of network elements and evaluating their practicality after a certain period of time [13]. Such partial migrations lead to hybrid IP/SDN networks, where both legacy IP nodes and SDN nodes coexist. In this type of network, both kinds of equipment need to be coordinated, as well as the protocols used for routing (i.e., OSPF and OpenFlow) [14, 15].

The problem of energy efficiency has been studied in related literature. In [16], an optimal, linear programming-based solution that optimizes the energy efficiency of an SDN network, along with a greedy heuristic for its application in large networks, are proposed. These solutions consider the possibly limited number of OpenFlow rules that can be installed in each SDN switch and provide up to 18.2% energy savings w.r.t. no energy efficiency solutions. However, this proposal is designed for pure SDN networks and is, thus, unsuitable for hybrid IP/SDN networks. Wang *et al.* extended the problem to hybrid IP/SDN networks in [17]. This work demonstrated the NP-hardness of the energy efficiency optimization problem in hybrid IP/SDN networks and proposed a heuristic algorithm to solve the problem based on spanning tree algorithms. The impact of traffic load, network size and SDN nodes' number on the achieved energy savings was also analyzed.

A different approach for energy efficiency optimization is described in [18], which optimizes the energy efficiency of a hybrid IP/SDN network by proposing the SENAtoR algorithm. SENAtoR employs a *smooth power-off* technique where SDN switches stop sending control traffic to legacy IP equipment before being powered off, giving the IP part of the infrastructure time to adapt. SENAtoR is shown to have near-optimal performance and takes only milliseconds to execute. Finally, [19] presented *Hybrid Energy-Aware Traffic Engineering* (HEATE). In contrast to the previous approaches, which focused on optimizing energy efficiency on SDN devices and allowing the solution to be interoperable with IP equipment, HEATE actively optimized energy efficiency in both kinds of equipment for hybrid IP/SDN networks. To do so, HEATE found the optimal setting of the OSPF link weight and the split ratio on SDN nodes. In particular, HEATE considered that IP routers performed shortest path routing by optimizing OSPF link weights, while SDN nodes satisfied multipath routing with traffic flow splitting through the action rules defined by the controller. HEATE is shown to provide better performance than flow allocation-based and OSPF-based energy efficiency algorithms in multiple network topologies. However, all these approaches are fundamentally different from the proposal of this work, as they exclusively focus on energy consumption, while other crucial aspects (e.g., QoS metrics, link performance) are not considered [20].

The consequence of single-objective energy efficiency optimization is that load balancing is not performed. Load balancing is a key TE technique in hybrid IP/SDN networks, normally performed by minimizing the maximum link utilization (MLU) through an equal spreading of the traffic load among links (ECMP protocol) [21]. Load balancing allows for a reduction of the OPEX in networks, as well as for a growth of the user base due to the low average link utilization derived from its usage [22, 23]. In hybrid IP/SDN networks, this kind of optimization was introduced in [24]. This proposal separated flows into two categories: *uncontrollable flows*, i.e., flows that were routed by legacy IP equipment, and thus, could not be controlled, and *controllable flows*, which could be controlled because they made use of SDN switches. This work formulated the problem of MLU minimization in hybrid IP/SDN networks by using

ECMP techniques in controllable flows. Furthermore, a fully polynomial time approximation scheme (FPTAS) was also proposed. In [25], Ren *et al.* achieved a near-optimal TE in hybrid IP/SDN networks in polynomial time by proposing a distributed algorithm named TEDR. TEDR relaxed and decomposed the original problem, which was formulated using linear programming, into multiple subproblems related to each of the SDN switches. A small TEDR controller associated with each SDN switch then solved the subproblem assigned to the switch and broadcasted this local solution to the whole network. Finally, local solutions are compared to find a globally near-optimal solution. TEDR achieved a substantial MLU reduction in hybrid IP/SDN networks with at least a 30% SDN deployment rate.

One of the latest related works on the subject is [26]. In this paper, Guo *et al.* presented HybridFlow, a load balancing solution centered on scalability. HybridFlow made use of the hybrid mode presented in some SDN switches that allowed them to route most flows using OSPF without involving the SDN controller to reduce the overhead. To do so, HybridFlow detected *crucial flows*: flows that carried a high traffic load and that would be directly routed through critical links using OSPF. These crucial links were detected using a *Variation Slope*, a metric also proposed by this work to identify the links carrying the highest set of flows with higher traffic loads. Crucial flows are then rerouted to perform load balancing using OpenFlow. While HybridFlow enabled near-optimal load balancing with minimal overhead, it also required a fully deployed SDN topology rather than a hybrid IP/SDN topology. Finally, in [27], the NP-hardness of the traffic engineering problem in hybrid IP/SDN networks was proven, and an algorithm named *SDN/OSPF Traffic Engineering* (SOTE) was proposed to solve it. The main purpose of SOTE was to minimize the MLU of a hybrid IP/SDN network by jointly optimizing the OSPF weight setting of the entire network in order to balance outgoing flows at IP nodes, as well as the traffic splitting ratio of flows that aggregated SDN nodes. SOTE was shown to achieve nearly optimal results in realistic topology, even with only a 10% SDN deployment rate. However, none of these works considered the effects of load balancing on energy consumption, as rerouting and the use of the ECMP protocol make use of more links, which in turn requires more links to be powered on, increasing the network's overall energy consumption.

Although both the energy consumption problem and the traffic load balancing problem have been studied separately in hybrid IP/SDN networks, studying them as a joint problem, considering the trade-off between the optimization of routing to maximize energy efficiency and its optimization for MLU minimization, has not been addressed in prior works. Thus, to solve this joint problem and provide a guaranteed QoS, this paper proposes a multi-objective optimization problem, including both its definition and formalization. In particular, two metrics are considered as the minimization objectives: the power consumption of the network and the MLU. This work extends our previous work already published in [28] by performing an evaluation of the impact that SDN nodes have on the flows number and on the amount of traffic that can be controlled, analyzing the impact of the initial weight setting of SDN links on the obtained results and proposing an additional genetic algorithm (GA)-based approach for the power saving phase. Moreover, for this solution to be able to scale, and thus, for it to become practically solvable in large topologies, we also present the Hybrid Spreading Load Algorithm (HSLA), a heuristic solution for this joint problem. Concretely, the main contributions of this work are as follows:

- The consideration of the trade-off between energy consumption and load balancing during the migration from legacy IP to SDN-enabled networks.
- The definition of the joint, multi-objective energy efficiency and the load balancing problem in hybrid IP/SDN networks that fairly pursues both goals.
- The evaluation of the impact of different SDN selection methods on flows number and on the amount of traffic that can be controlled.
- The evaluation of the impact of the initial weight setting of SDN links on the aforementioned metrics.
- The proposal of the HSLA heuristic algorithm, including a GA-based approach for the link switch off process, which enables the practical and timely solution of the problem in large topologies.
- The comparison of HSLA with existing solutions from both the energy efficiency and the load balancing state-of-the-art within realistic conditions.

The rest of this paper is organized as follows. Sec. 2 describes and defines the problem. The HSLA algorithm is described in Sec. 3. Sec. 4 reports experimental results and their analysis. Ultimately, Sec. 5 draws some conclusions and future works.

2 Problem Definition

We consider a hybrid IP/SDN network modeled as a directed network graph $\mathcal{G} = (\mathcal{N}, \mathcal{L})$, where $\mathcal{N}$ is the nodes set and $\mathcal{L}$ is the directed links set. The set of nodes $\mathcal{N}$ is further divided in two subsets: SDN nodes, $\mathcal{N}_{SDN} \in \mathcal{N}$, or IP nodes, $\mathcal{N}_{IP} \in \mathcal{N}$, where $\mathcal{N}_{SDN} \cup \mathcal{N}_{IP} = \mathcal{N}$. With respect to the set of links, each link $l_{i,j} \in \mathcal{L}$ connecting node $i \in \mathcal{N}$ to node $j \in \mathcal{N}$ has a capacity of $C_{i,j}$ units to accommodate traffic flows, and a power consumption referred to as $P_{i,j}$.

Traffic matrix $\mathcal{T}$ models the traffic and is composed of a set of demands $\delta_{s,d} \in \mathcal{T}$ from each source node $s \in \mathcal{N}$ to each destination node $d \in \mathcal{N}$ in the network. In other words, $\delta_{s,d} \in \mathcal{T}$ refers to the traffic demand entering the network from node s and leaving it through node d.

The goal of the problem is to find an optimal network configuration that jointly minimizes the maximum link utilization and the network power consumption.

Moreover, to define the Mixed Integer Linear Programming (MILP) formulation aimed at solving the Multi-Objective and the Multi-Commodity-Flow (MO-MCF) optimization problem, the following set of variables is required:

- $y_{i,j}$ is a binary variable that represents the operational mode of link $l_{i,j}$. Its value is $y_{i,j} = 1$ if the link is active. Otherwise, if $y_{i,j} = 0$, the link is turned off;
- $f_{i,j}^{s,d}$ is an integer variable that represents the amount of traffic demand $\delta_{s,d} \in \mathcal{T}$ that is routed on link $l_{i,j}$;
- $n_{i,j}^{d}$ is a binary variable indicating whether node i uses node j as the next hop in its forwarding table to reach destination node d;
- c_i^d is an integer variable representing the cost of the path to go from node i to node d;
- $h_{i,j}$ is an integer variable indicating the interior gateway protocol (IGP) weight of link i,j, which is considered by the IP routers to determine the shortest path routing;
- U is a real variable indicating the maximum link utilization (MLU) in the network.
- $\mathcal{N}_i^+$ and $\mathcal{N}_i^-$ are subsets of $\mathcal{N}$ that contain the nodes to which node i has *incoming* and *outgoing* links, respectively.

Considering the variables just described, the optimization problem has as goal the jointly minimization of the network power consumption , as well as its maximum link utilization (MLU). For this purpose, two objective functions are needed to solve the multi-objective optimization problem, that are defined in eqs. (1-2) as follows:

$$F_1 = \min \sum_{l_{i,j} \in \mathcal{L}} y_{i,j} \cdot P_{i,j} \tag{1}$$

$$F_2 = \min \quad U \tag{2}$$

The minimization of the global network power consumption is represented by Eq. (1), while the objective pursued by eq. (2) minimizes the MLU. Therefore, the MO-MCF problem is defined by the joint optimization of both functions as follows:

$$\min \left[F_1;\ F_2\right] \tag{3}$$

subject to:

$$\sum_{j \in \mathcal{N}_i^-} f_{i,j}^{s,d} - \sum_{j \in \mathcal{N}_i^+} f_{j,i}^{s,d} = \begin{cases} \delta_{s,d} & \text{if } i = s \\ -\delta_{s,d} & \text{if } i = d \\ 0 & \text{if } i \neq s, d \end{cases} \quad \forall i, s, d \in \mathcal{N}, \delta_{s,d} \in \mathcal{T} \tag{4}$$

$$\sum_{\delta_{s,d} \in \mathcal{T}} f_{i,j}^{s,d} \leq y_{i,j} \cdot C_{i,j} \qquad \forall i, j \in \mathcal{L} \tag{5}$$

$$\frac{1}{C_{i,j}} \sum_{\delta_{s,d} \in \mathcal{T}} f_{i,j}^{s,d} \leq U \qquad \forall i,j \in \mathcal{L} \tag{6}$$

$$f_{i,j}^{s,d} \leq n_{i,j}^{d} \cdot \delta_{s,d} \qquad \forall i,j \in \mathcal{L} : i \in \mathcal{N}_{\mathrm{IP}}, \delta_{s,d} \in \mathcal{T} \tag{7}$$

$$\sum_{j \in \mathcal{N}_i^-} n_{i,j}^{d} = 1 \qquad \forall i,d \in \mathcal{N} : i \neq d \tag{8}$$

$$0 \leq c_j^d + h_{i,j} - c_i^d \leq (1 - n_{i,j}^d) \cdot M \qquad \forall i,j \in \mathcal{L}, d \in \mathcal{N} \tag{9}$$

$$1 - n_{i,j}^d \leq c_j^d + h_{i,j} - c_i^d \qquad \forall i,j \in \mathcal{L}, d \in \mathcal{N} \tag{10}$$

$$1 \leq h_{i,j} \leq h_{\max} \qquad \forall i,j \in \mathcal{L} \tag{11}$$

Classical flow conservation constraints are described in eq. 4, whereas the capacity constraint on links is represented by eq. 5, where the available capacity upper binds the total traffic routed on a link. Specifically, if the link is in on status ($y_{i,j} = 1$), all of its capacity can be employed to transport the traffic; otherwise, its capacity is equal to 0 and the link cannot be used. Eq. 6 leads the variable θ to be equal to or greater than the MLU in the network. In the presented model, IP routers are constrained to use a single and shortest path routing and to forward traffic according to the destination. These aspects are involved in the problem formulation through the constraints reported in Eqs. 7-11. Especially, Eq. 7 models destination-based forwarding by allowing the IP router i to send traffic destined to node d over link i,j only if node j is the next hop for the destination ($n_{i,j}^d = 1$). Eq. 8 imposes the single path routing for IP routers, and specifically it forces node i to choose only one next hop to reach destination node d. It is worth mentioning that, in spite of the fact that SDN nodes can forward traffic over multiple paths, it is important to impose Eq. 8 to all the nodes (not limited only to IP routers). In this way, IP routers are forced to consider only single and shortest paths to forward the traffic, which are calculated according to the weight variables ($h_{i,j}$). Anyway, SDN nodes are able to override these rules since the next hop variables for these types of nodes are not logically bound to the traffic forwarding (Eq. 7 is imposed only on IP routers). Finally, Eqs. 9-11 enforce the shortest path policy. They impose that if $n_{i,j}^d = 1$, then the cost of the path between the nodes i and d must be equal to the weight of the arc connecting it with its neighbor, j, ($h_{i,j}$) including the cost of the path between j and d.

Since the presented problem formulation is NP-hard [29], next, a heuristic approach is presented to solve it in polynomial time.

3 Hybrid spreading load algorithm (HSLA)

The proposed algorithm, named Hybrid Spreading Load Algorithm (HSLA), which aims at solving the MILP problem defined in Sec. 2, is described in next subsections by pointing out algorithmic aspects and architectural details.

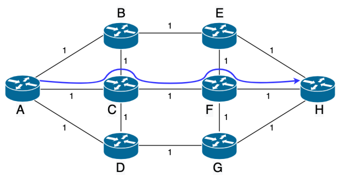


Fig. 1: A to H path in a legacy IP network.

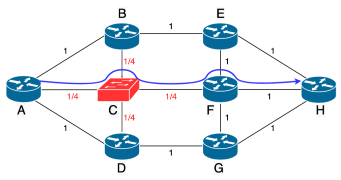


Fig. 2: A to H path in a hybrid IP/SDN network considering C as SDN node.

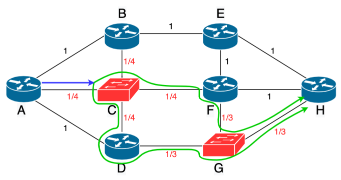


Fig. 3: A to H path in a hybrid IP/SDN network considering C and G as SDN nodes.

3.1 Exploiting SDN nodes for traffic splitting

The HSLA concept is to take advantage by using the generalized forwarding of the SDN paradigm to split the traffic entering the set of SDN nodes to balance the traffic load. Therefore, an SDN node can split a reaching flow over the rest of the outgoing ports to ensure that the traffic is balanced toward the next hop. The main benefits of doing this according to the generalized forwarding and the SDN control plane are: i) the traffic can be accurately divided over several outgoing paths and ii) the considered paths do not need to be the shortest ones. These two points can be summarized according to the concept of multipath routing (MPR). In contrast, traditional IP routers, which work according to destination-based forwarding and rely on shortest path routing protocols (e.g., OSPF [10]), can split traffic over equal cost multipaths (ECMPs, [21]), but this operation has several limitations. Firstly, traffic splitting is based on flow hashing methods, that allow to balance only in terms of flow number (not considering the actual traffic volume) and to lead to low accuracy. Furthermore, only the shortest paths can be considered, thus, reducing the load balancing opportunities. Finally, all these decisions are delegated to the local control plane agent with logic that cannot be centrally controlled to achieve a global objective. For this reason, in this paper, we decide not to rely on the ECMP mechanism of the IP routers, allowing traffic splitting only at SDN nodes, where this process is effective and flexible as a result of the availability of the routing mechanism of group tables or using specific flow rules. In particular, we assume that the capabilities of SDN switches are exploited to perform accurate traffic flow over multiple shortest paths.

The idea behind the proposed routing scheme is represented through Figs. 1-3, where an 8-node hybrid IP/SDN network is reported. We are interested to the path that is evaluated to route the flow traffic from node A to node H. In Fig. 1 the resulting path for the case where all the nodes are legacy IP nodes ($P_{A,H} = \{A - C - F - H\}$) is represented by the blue line. For the sake of simplicity, all the weights for links are here set to 1. In this case, destination-based forwarding is applied, such as in traditional routing protocols, e.g., OSPF [10].

Fig. 2 represents the scenario where a traditional IP node is replaced by an SDN node at a particular stage in the transition from IP to SDN and the selection method for such replacement indicates that the upgraded node must be C. In this case, the link associated with the SDN node are weighted by decreasing their values from $h_{i,j} = 1$ (initial setting) to $h_{i,j} = \frac{h^-_{i,j}}{k_i} = \frac{1}{4}$, where the weight of link $l_{i,j}$ before the update is represented by $h^-_{i,j}$ and k_i is a metric that purposes to attract the flows to node i, e.g., its degree. The decreasing of the link weights related to SDN nodes is done to force the flows to pass through them, leading to balance the traffic among different links exploiting the ECMP protocol [21]. Despite this, in the reported example, although link weights are modified, the single SDN node replacement does not affect the path followed by flow $A - H$ (blue line) with respect to the case of Fig. 1.

By upgrading a second node to SDN, e.g., G, the flow arriving at node C is equally divided among the set of shortest paths from it toward the destination (see Fig. 3). In the example, two shortest paths with the same cost exist from C to H: $P^1_{C,H} = \{C - D - G - H\}$ and $P^2_{C,H} = \{C - F - G - H\}$, which are represented by green lines. Thus, the traffic volume represented by the blue line is split by two and steered through the aforementioned (green) paths. With this traffic division, the proposed solution targets the traffic load to be balanced and the MLU to be reduced through the use of the ICMP protocol at the SDN nodes.

3.2 HSLA Description

Next, we present the proposed solution, which is composed of 2 algorithms. The former aims at spreading the traffic through the subset of SDN nodes as a result of using the ECMP protocol (pseudo-code reported in Alg. 1). Once the traffic is handled, the second algorithm tries to save energy by switching off links. In addition, for the link switch off (LSO) process, two solutions have been considered: the Less Loaded Links Algorithm (L3A), which iteratively removes the less loaded link in the network (see Alg. 2) and repeats and the Genetic Algorithm-based LSO (GA-LSO), which exploits the use of GAs to power off links and steer the traffic among the remaining subset of links (Alg. 3).

Starting with the HSLA, whose goal is the minimization of the MLU, 2 parameters are required as input. The first parameter is the network graph $\mathcal{G} = (\mathcal{N}, \mathcal{L})$, with $\mathcal{N}_{\mathcal{IP}} \in \mathcal{N}$ as the set of legacy IP nodes, $\mathcal{N}_{\mathcal{SDN}} \in \mathcal{N}$ composed of the set of SDN nodes, and $\mathcal{N}_{SDN} \cup \mathcal{N}_{IP} = \mathcal{N}$. The second parameter is the traffic matrix $\mathcal{T}$ to be satisfied. First, an initialization phase for setting the link weights is performed in lines $(1 - 9)$. As introduced in Sec. 3.1, $h_{i,j} = 1$ is set for the weight for links connecting two IP nodes, whereas they are decreased to $h_{i,j} = \frac{1}{k_i}$ when they connect an SDN node i, with k_i as the related metric representing the node importance.

Once the initial weight setting is performed, the objective of the HSLA is to provide for all the traffic demands contained in the traffic matrix $\mathcal{T}$ passed as input, i.e., to find a feasible routing matrix $\mathcal{R}$ for $\mathcal{T}$ that minimizes the MLU. For this aim, each traffic demand (s, d) is iteratively handled. First, the

k shortest paths are evaluated from source node s to destination d according to destination-based forwarding (line 13).

Algorithm 1 Pseudo code of the proposed Hybrid Spreading Load Algorithm (HSLA).

```
Require: A network graph: $\mathcal{G} = (\mathcal{N}, \mathcal{L})$, a traffic matrix: $\mathcal{T}$
 1: create $\mathcal{W} \leftarrow \emptyset$                                    ▷ Set for link weights
 2: create $\mathcal{R} \leftarrow \emptyset$                                    ▷ Routing matrix
 3: for all $l_{i,j} \in \mathcal{L}$ do                                         ▷ Update link weights
 4:     if $i \in \mathcal{N}_{\text{SDN}}$ then
 5:         $w_{i,j} \leftarrow \frac{1}{k_i}$
 6:     else
 7:         $w_{i,j} \leftarrow 1$
 8:     end if
 9: end for
10: for all $\delta_{s,d} \in \mathcal{T}$ do                                    ▷ Find path $\forall \delta_{s,d} \in \mathcal{T}$
11:     create $p_{s,d} \leftarrow \{\}$                                         ▷ Path for demand $\delta_{s,d}$
12:     create $\lambda \leftarrow$ false                                        ▷ Boolean for path found
13:     $P \leftarrow$ k_shortest_paths($\delta_{s,d}, \mathcal{G}$)
14:     while $\lambda ==$ false do                                              ▷ Check each path $p \in P$
15:         if $p \in P$ is an IP path then                                      ▷ No SDN nodes
16:             $b \leftarrow$ check_path_constraints($\delta_{s,d}, p, \mathcal{G}$)   ▷ Check link constraints and no loops
17:             if $b ==$ true then
18:                 $\mathcal{R} \leftarrow$ assign_flow_to_path($\delta_{s,d}, p, \mathcal{G}$)
19:                 $p_{s,d}$.concatPath($p$)
20:                 $\lambda \leftarrow$ true
21:             end if
22:         else                                                                 ▷ There are SDN nodes in the path
23:             for all $l_{i,j} \in p$ do
24:                 if $i \in \mathcal{N}_{\text{SDN}}$ then                     ▷ It is an SDN node
25:                     $P^* \leftarrow$ k_shortest_paths($\delta_{i,d}, \mathcal{G}$)
26:                     $u \leftarrow$ num_paths_same_cost($P^*$)
27:                     for all $p' \in P^*$ with same cost do
28:                         $b \leftarrow$ check_path_constraints($\frac{\delta_{s,d}}{u}, p', \mathcal{G}$)
29:                         if $b ==$ true then
30:                             $\mathcal{R} \leftarrow$assign_flow_to_path($\frac{\delta_{s,d}}{u}, p', \mathcal{G}$)
31:                             $p_{s,d}$.concatPath($p'$)
32:                         else
33:                             clear incomplete flow assignments
34:                             check next path $p \in P$ (line 14)
35:                         end if
36:                     end for
37:                     $\lambda \leftarrow$ true
38:                 else                                                         ▷ It is an IP node
39:                     $b \leftarrow$ check_link_constraints($\delta_{s,d}, l_{i,j}, \mathcal{G}$)
40:                     if $b ==$ true then
41:                         $\mathcal{R} \leftarrow$ assign_flow_to_link($\delta_{s,d}, l_{i,j}, \mathcal{G}$)
42:                         $p_{s,d}$.addLink($l_{i,j}$)
43:                     else
44:                         clear incomplete flow assignments
45:                         check next path $p \in P$ (line 14)
46:                     end if
47:                 end if
48:             end for
49:         end if
50:     end while
51: end for
52: return feasible routing matrix $\mathcal{R}$ if it exists
```

Then, each path in the reported set is verified to check if it is a *noncontrollable path* (all the nodes are IPs) or a *controllable* path (at least one node is an SDN). In the former case (lines $15-23$), a classic destination-based forwarding is employed to steer the traffic, checking that link constraints are satisfied and no loops are formed (line 16). If the SDN nodes are found in the evaluated path (it is a *controllable path*), the heuristic verifies each node in the path from s to d to discern whether they are SDN or IP nodes (lines $24-49$). If the checked node i is an SDN, the k shortest path procedure is executed again (line 27) to get the subset of u paths with the same cost from the SDN node i to the destination d. Then, the traffic demand (s,d) arriving at i is equally divided among the subset of u paths with an equal cost to d and a volume of $\frac{(s,d)}{u}$ (lines $29-37$). For each evaluated path, link constraints and the presence of loops are also checked (line 28). If the evaluated node i is an IP, the traffic demand is fully routed over the outgoing link without performing traffic splitting (lines $39-47$). This process is repeated until all the nodes in the path from s to d are evaluated and the traffic is successfully steered, satisfying link constraints throughout of the path and the avoidance of loops. Finally, the output reports a feasible routing matrix $\mathcal{R}$.

3.3 Less Loaded Links Algorithm

Once Alg. 1 evaluates a routing solution where the traffic load is balanced and thus the MLU is minimized, a feasible network configuration that minimizes the required power consumption is found by a second algorithm. Alg. 2 reports the pseudo code of the Less Loaded Links Algorithm (L3A). It takes as input the routing matrix outputted by the HSLA, $\mathcal{R}$, as well as the network graph and the traffic matrix to be served. Initially, the set of links are sorted according to their load in ascending order (line 4). Then, each link is removed (turned off) from the network topology, and the HSLA is executed again to find feasible routing (if possible). If there is no feasible routing, the link is returned to the topology (switched on), and the next link is considered. This process is repeated until all the network links are evaluated. Ultimately, a feasible routing matrix, $\mathcal{R}'$, is reported, which i) balances the link load and ii) reduces the network power consumption.

3.4 Genetic Algorithm-based Link Switch Off

The second algorithm that is proposed for the power saving procedure is based on GAs, a type of heuristic based on the mechanisms of natural evolution where individuals reproduce and the more skilled ones survive [30].

To solve the problem, a chromosome representing a solution must be defined. In our case, a chromosome c represents a network configuration in terms of link activity (Eq. 12). In particular, each gene, g_k, of chromosome c is a binary value representing the operational mode of the k-th link. If link k is powered on, then $g_k = 1$. Otherwise, if the link is powered off, then $g_k = 0$.

Algorithm 2 Pseudo code of the proposed Less Loaded Links Algorithm (L3A).

```
Require: $\mathcal{G} = (\mathcal{N}, \mathcal{L})$, $\mathcal{T}$, the routing matrix obtained by Alg. 1: $\mathcal{R}$
 1: create $\mathcal{G}' \leftarrow \emptyset$                              ▷ Aux network graph
 2: create $\mathcal{L}^* \leftarrow \emptyset$, $\mathcal{L}' \leftarrow \mathcal{L}$                          ▷ Aux links set
 3: create $\mathcal{R}' \leftarrow \emptyset$                         ▷ Output routing matrix
 4: $\mathcal{L}^* \leftarrow$ sort_links_by_load($\mathcal{G}, \mathcal{R}$, 'ascend')
 5: for all $l_{i,j} \in \mathcal{L}^*$ do
 6:     $\mathcal{L}' \leftarrow \mathcal{L}' - l_{i,j}$                   ▷ Remove link from the set
 7:     $\mathcal{G}' = (\mathcal{N}, \mathcal{L}')$
 8:     $\mathcal{R}' \leftarrow$ HSLA($\mathcal{G}', \mathcal{T}$)
 9:     if $\mathcal{R}'$ is not feasible then
10:         $\mathcal{L}' \leftarrow \mathcal{L}' \cup l_{i,j}$
11:     end if
12: end for
13: return feasible routing matrix $\mathcal{R}'$ if it exists
```

This representation fits with the philosophy of canonic GAs, which use binary strings to represent individuals.

$$c = \{g_1, g_2, ..., g_L\}, g_k \in \{0;\ 1\} \tag{12}$$

Each potential solution represented by a chromosome must be evaluated to obtain its suitability to the problem, i.e., its goodness, according to a fitness function. In this problem, the fitness function must check if the network configuration mapped by the chromosome ensures that the routing constraints (link capacities) and whether all the traffic demands are satisfied. If the feasibility check is correct, then the resulting fitness value is the number of powered links. The goal of the GA-LSO is, therefore, to minimize this number to ensure that the power savings are increased.

In the following, the description of the proposed GA-based LSO algorithm is provided, including the operators and functions used to carry out the evolutionary process. It takes six parameters as input: i) the network graph, $\mathcal{G} = (\mathcal{N}, \mathcal{L})$, ii) the routing matrix obtained by HLSA, $\mathcal{R}$, iii) a traffic matrix, $\mathcal{T}$, iv) the population size, κ, v) the mutation rate, m_r, and vi) the maximum generations number that compose the evolutionary process, θ.

At the beginning, four variables are initialized: *gen* as an indicator of the current generation, P_{gen} as the population of individuals to be evaluated (initially empty), and the two variables that will be returned as output: sol_{best} as the best solution found and $fval_{\text{best}}$ as the numerical goodness of such a solution. The latter variable is initialized to a large value to be minimized during the execution of the GA-based LSO.

In the first step, the GA creates the initial population, P_{gen}, of size κ with the set of individuals to be evaluated by the fitness function. Next, each individual $c \in P_{\text{gen}}$ is evaluated by applying the fitness function previously described, and the best solution found is iteratively updated (lines $3-9$). The evolutionary process starts at line 10 with the selection of the individuals that will be part of the next generation. In particular, *crossover* and *mutation* processes are applied to the current population, P_{gen}, to create offspring (*children*) that will be evaluated in the following generations. Specifically, single-point crossover and uniform mutation are applied. The mutated *children** obtained after applying the genetic operations are evaluated by the fitness function, and the best solution found is again iteratively updated (lines $15-21$). The process ends after the execution of θ iterations, i.e., the maximum generations number passed as input. The algorithm returns the best solution found, sol_{best}, and its associated fitness value, $fval_{\text{best}}$. The best solution represents the best network configuration in terms of power savings (least number of active links), which respects the routing and capacity constraints.

Algorithm 3 Pseudo code of the proposed GA-based LSO algorithm.

```
Require: A network graph: $\mathcal{G} = (\mathcal{N}, \mathcal{L})$, the routing matrix obtained by Alg. 1: $\mathcal{R}$, a
    traffic matrix: $\mathcal{T}$, population size: $\kappa$, mutation rate: $m_r$, maximum generations: $\theta$
 1: create $gen \leftarrow 0$, $P_{\text{gen}} \leftarrow \emptyset$, $sol_{\text{best}} \leftarrow \emptyset$, $fval_{\text{best}} >> L$
 2: $P_{\text{gen}} \leftarrow createPopulation(\mathcal{G}, \kappa, \texttt{uniform})$
 3: for all chromosome $c$ in $P_{\text{gen}}$ do
 4:     $(fval_c, sol_c) \leftarrow \text{fitness}(c, \mathcal{G}, \mathcal{R}, \mathcal{T})$
 5:     if $fval_c < fval_{\text{best}}$ then
 6:         $sol_{\text{best}} \leftarrow sol_c$
 7:         $fval_{\text{best}} \leftarrow fval_c$
 8:     end if
 9: end for
10: while $gen \leq \theta$ do
11:     $parents \leftarrow selectParents(P_{\text{gen}}, \texttt{roulette})$
12:     $P^* \leftarrow P_{\text{gen}} \setminus parents$
13:     $children \leftarrow performCrossover(parents, \texttt{single-point})$
14:     $children^* \leftarrow performMutation(children, m_r, \texttt{uniform})$
15:     for all chromosome $c$ in $children^*$ do
16:         $(fval_c, sol_c) \leftarrow \text{fitness}(c, \mathcal{G}, \mathcal{R}, \mathcal{T})$
17:         if $fval_c < fval_{\text{best}}$ then
18:             $sol_{\text{best}} \leftarrow sol_c$
19:             $fval_{\text{best}} \leftarrow fval_c$
20:         end if
21:     end for
22:     $P^* \leftarrow P^* \cup children^*$
23:     $gen \leftarrow gen + 1$
24:     $P_{\text{gen}} \leftarrow P^*$
25: end while
26: return $sol_{\text{best}}, fval_{\text{best}}$
```

4 Experimental Results

In this section, we provide an experimental evaluation of the proposed solution. Firstly, the simulation environment is reported. Next, an analysis is provided to evaluate different methods for the selection of the nodes to be migrated from IP to SDN, both in terms of controllable flows and the amount of controllable traffic. We adopt the definition given in [24] to refer to *controllable flows*. A *controllable flow* is a flow that is routed by SDN equipment. In contrast, An *uncontrollable flow* is a flow that is completely routed by legacy IP equipment, and thus, cannot be controlled. Therefore, controllable traffic is the amount of traffic associated with the set of controllable flows. Moreover, the impact of the weight setting of links connecting SDN nodes is also evaluated by considering different approaches. To analyze the benefits of applying the HSLA over topologies of different sizes and traffic loads, a thorough performance analysis is carried out. Metrics, such as MLU and power savings, are evaluated. Finally, a comparison with two representative state-of-the-art solutions that separately address the load balancing and the energy consumption problem in hybrid IP/SDN networks is also performed.

4.1 Simulation Setup

In the simulations, we consider three network topologies of different sizes: Nobel (17 nodes, 52 links), Geant (22 nodes, 72 links), and Germany (50 nodes, 176 links). To evaluate the variability of the daily traffic patterns in each network, a set of traffic matrices are retrieved from [31]. In particular, 5 TMs are considered for the analyses by scaling down the peak TM in the data set by a factor of 0.4, 0.5, 0.7, and 0.9, respectively.

Link capacities are set as follows. First, we select the peak TM, and we route it over a set of shortest paths derived after applying the Dijkstra algorithm on the network graph. After this step, each link $l_{i,j}$ carries an amount of traffic $t_{i,j}$. Then, we assume that the capacity of each link can be upgraded by installing a set of line cards. A line card has a capacity of Δ_C equal to $0.5 \max\limits_{l_{i,j}}(t_{i,j})$, i.e., half of the traffic carried by the link with the highest link utilization. Finally, we consider installing the minimum line cards number that is needed by each link to make their utilization less than 100%.

In our analysis, we assume that all the network links have the same power consumption when they are active ($P_{i,j} = 1$). The transition from the IP to SDN networks is performed step-by-step, i.e., by upgrading one IP legacy node to SDN at a time. At each stage, one node is selected as the most suitable to be upgraded to the SDN. For the selection process, four methods are considered: i) Highest degree first (HDF), which sorts the set of potential nodes (remaining IP nodes) according to their degree, i.e., the number of incoming and outgoing links; ii) Highest closeness centrality (HCC), to select the node having the shortest distance to the rest of the nodes; iii) Highest betweenness centrality (HBC), with the aim of obtaining the node that is involved in most of the shortest paths among the nodes in the network; and iv) Random.

The motivation for the choice of the aforementioned selection methods is explained next. If the upgrade of IP nodes to the SDN is performed according to the HDF method, it means that, after each stage, the controller will have the largest possible amount of information under control compared with the case where other potential nodes would be selected. Thus, the node with the highest degree is chosen so that the controller will be able to control the highest number of ports through, e.g., the use of the *OFPMP_PORT_STATS* messages. If HCC is the method for the SDN nodes selection, the focus is placed on the distance of a node to the rest of the nodes. This situation is interesting for detecting nodes that are able to spread information very efficiently throughout the network. If a node is selected as an SDN using HCC, it means that the destination node (as well as the source node) is close; thus, the path is shorter with regard to other potential nodes, and improvements in the energy efficiency can be experienced. Ultimately, we evaluate the importance of a node with HBC. In this case, the node with HBC is involved in most of the shortest paths for each source-destination pair. Since the HSLA is based on the computation of the shortest paths both from the source node and from each SDN node (if any) to the destination, the HBC seems to be an interesting option to be exploited in the SDN node selection process. Finally, randomness is also considered to show the benefits of using methods that are based on the features of the network graph.

4.2 Evaluation of the SDN nodes selection method

The first analysis that is proposed aims at evaluating the controllable flows number and the amount of controllable traffic that results after the migration of legacy IP nodes to SDN ones. In particular, Fig. 4 reports the controllable flows number as a function of the SDN nodes number for the Nobel, Geant, and Germany topologies. Four different methods for the selection of the SDN nodes are considered: HDF, HCC, HBC and Random. Simulations were run 25 times, and confidence intervals are shown in the figure. Since the initial link weight setting is important for evaluating the effectiveness of the proposal, the link weight setting of the SDN nodes in this evaluation is set according to their degree, $w_{i,j} = \frac{1}{k_i}$, with i as the SDN node and k_i as its degree. In the case of links connecting only IP nodes, their weight is set to $w_{i,j} = 1$. Looking at Fig. 4, it can be seen that the three methods that are based on network features (HDF, HCC and HBC) present similar results for small networks (Nobel and Geant), overcoming the one exploiting randomness. In the case of the Germany network (see Fig. 4c, larger differences can be found within the range $[0, 20]$ of SDN nodes: i) lines for HCC and HBC are close for a small SDN nodes number $[1 - 5]$ with a slightly better performance with regard to HDF; and ii) the performance of HCC decreases in the range $[5 - 20]$ compared to HDF and HBC, which maintains the best results for all the problem instances. Thus, in general, the HBC selection method is the one that presents the best

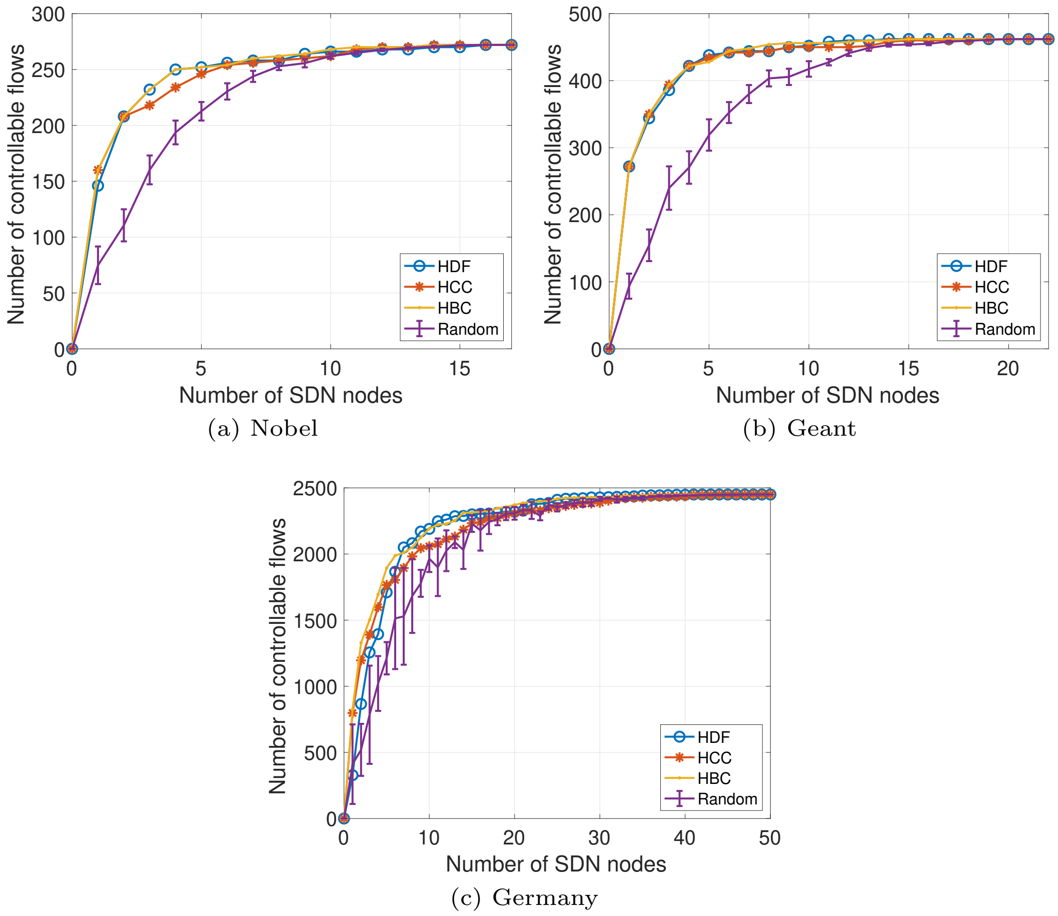


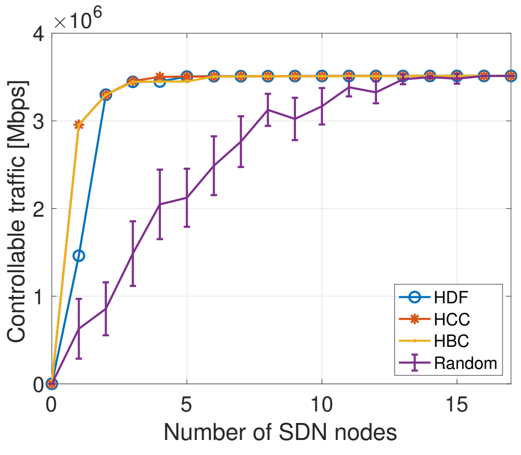
Fig. 4: Number of controllable flows as a function of the SDN nodes number for Nobel, Geant, and Germany. The link weight setting for the SDN nodes is set according to their degree, $w_{i,j} = \dfrac{1}{k_i}$.

performance in terms of controllable flows during the transition from IP to SDN.

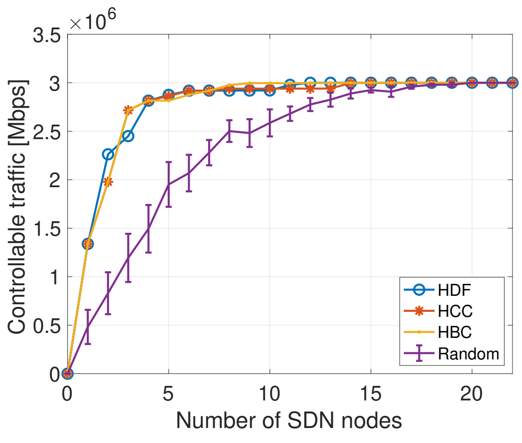
To verify the previous outcomes, Fig. 5 reports the amount of controllable traffic as a function of the SDN nodes number for the four selection methods and the three topologies. In this case, it can be remarked that HDF struggles with respect to HCC and HBC for a small SDN node number, especially in the case of large networks (see Fig. 5c), where its performance is similar to that of the worst selection method, i.e., Random. Thus, from these analyses, we can state that HBC is the selection method that is able to control the highest number of flows and the largest amount of traffic when migrating an IP network to an SDN network.

4.3 Evaluation of the link weight setting method

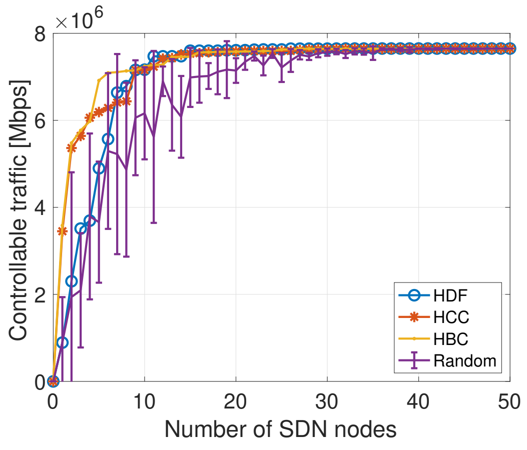
In this section, a discussion of the importance of selecting an appropriate link weight setting is provided. In particular, we focus on tuning the weights of the links connecting the SDN nodes. For the remaining links uniquely connecting



(a) Nobel



(b) Geant



(c) Germany

Fig. 5: Amount of controllable traffic as a function of the SDN nodes number for Nobel, Geant, and Germany. The link weight setting for SDN nodes is set according to their degree, $w_{i,j} = \dfrac{1}{k_i}$.

legacy IP nodes, their weight is set to 1, as in the OSPF protocol. This choice derives from the control a network admin is able to carry out on the traffic traversing SDN nodes. As a result, if the definition of the SDN link weights is set according to metrics related to the network structure and the flows number (and traffic) to be handled by the SDN nodes is increased, our proposed algorithm handles a thinner granularity on the traffic.

Three methods for setting the link weights are evaluated: i) LWS: the degree where the weights of the SDN links are inversely set according to their degree (as in the evaluations of Sec. 4.2 and the description of Fig. 3); ii) LWS: the closeness where the node closeness centrality is considered for the link weights; and iii) LWS: the betweenness where the metric used to assess the link weights is the betweenness centrality of the nodes. In all the cases, links attached to an SDN node are assigned a weight that is inversely proportional to the considered metric (degree, closeness or betweenness).

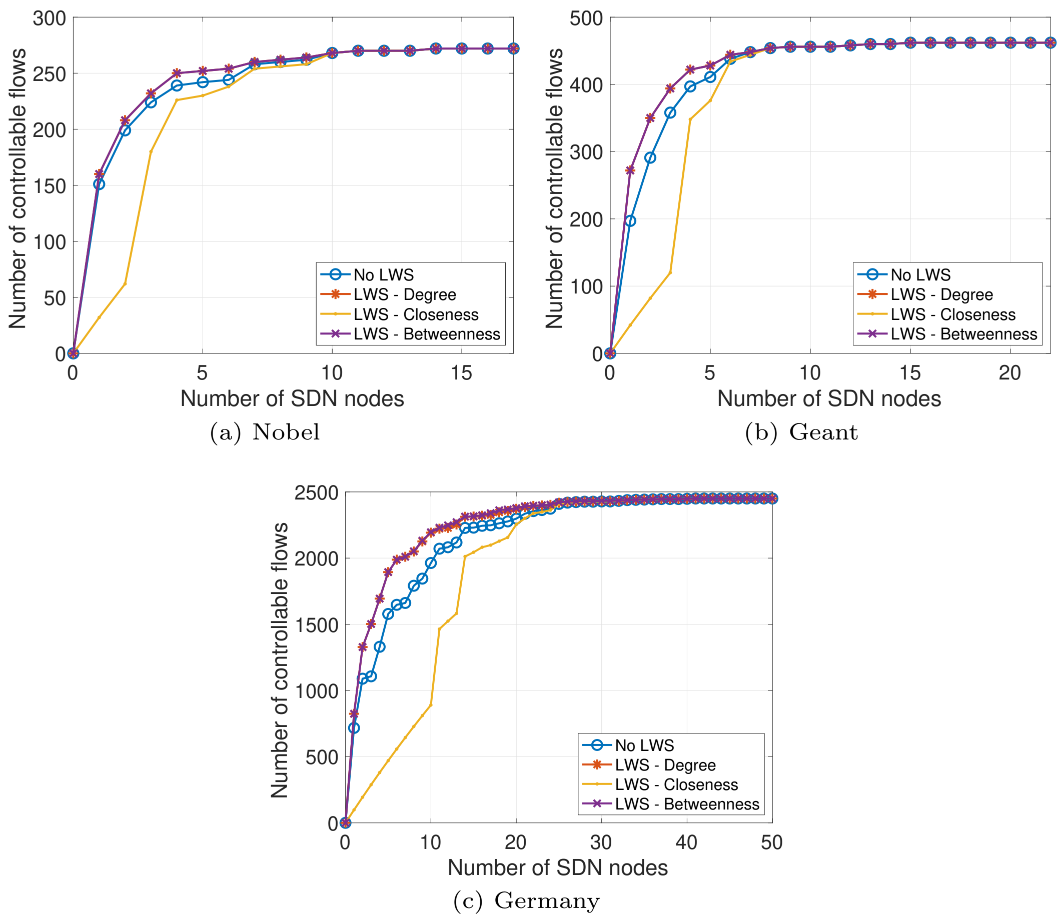
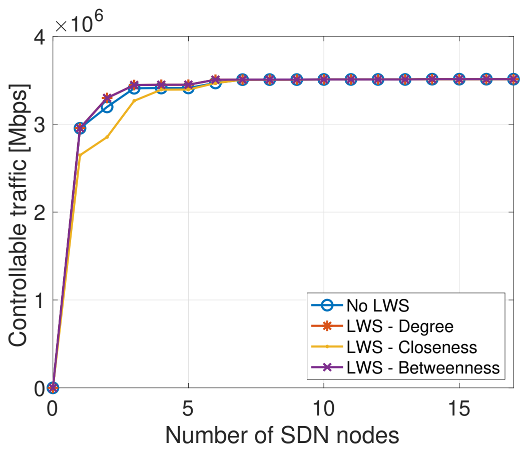


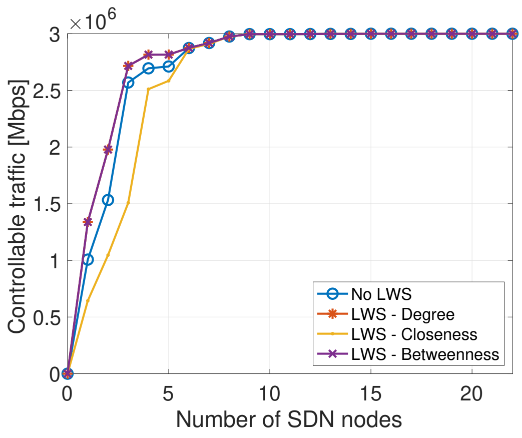
Fig. 6: Number of controllable flows as a function of the SDN nodes number for Nobel, Geant, and Germany. The SDN node selection method is HBC.

Fig. 6 shows the number of controllable flows as a function of the SDN nodes number for Nobel, Geant and Germany considering the three methods described above. Moreover, the case where all the links in the network have the same weight, i.e., $w_{i,j} = 1$ regardless of whether they are IP or SDN links, is also reported with the legend 'No LWS'. In the figures, it can be seen that the link weight setting based on either node degree or node betweenness centrality represents the best choice to increase the number of controllable flows for all the network topologies. The worst-case scenario is reported for the closeness centrality method, which is even worse than the case where no differences are adopted for SDN links and regular IP links, i.e., when no considerations for the link weights are assumed.

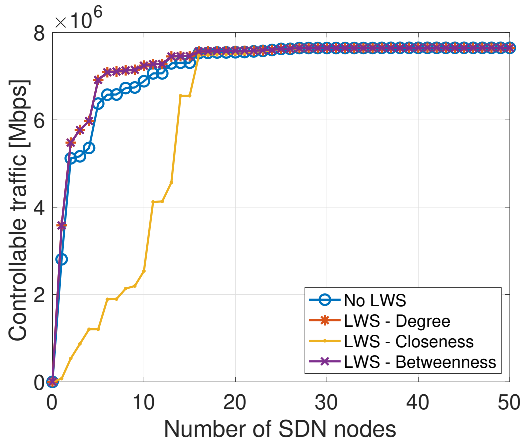
These results can be confirmed when looking at Fig. 7, where the amount of controllable traffic is shown as a function of the SDN nodes number. In this case, although the benefits of using a method to set the weights of the SDN links are slightly reduced compared to the number of controllable flows (especially for small networks, see Figs. 7a and 7b), the metrics based on the node degree and the betweenness centrality clearly outperform the rest. In conclusion, we can state that the impact of selecting an appropriate method



(a) Nobel



(b) Geant



(c) Germany

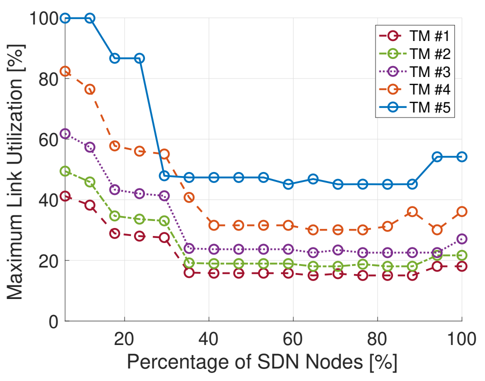
Fig. 7: Amount of controllable traffic as a function of the SDN node number for Nobel, Geant, and Germany. The SDN node selection method is HBC.

for the link weight setting is very important for the controllability of the flows, and hence, the amount of traffic that can be managed in the hybrid IP/SDN network. Thus, for the evaluation of the proposed solution, we consider the link weight setting based on the SDN node degree.

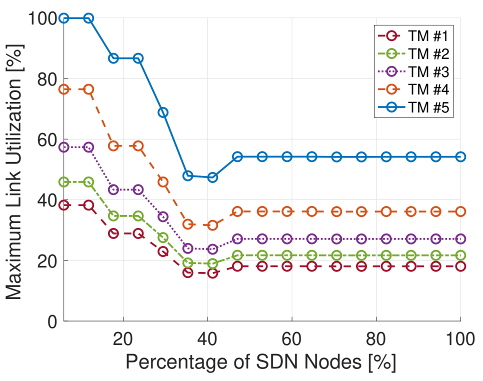
4.4 HSLA Performance Evaluation

The goal of the first analysis is to evaluate the effectiveness of the proposed solution considering the joint optimization of energy efficiency and load balancing during the transition from IP to SDN nodes. Fig. 8 shows the MLU obtained by the HSLA as a function of the SDN deployment rate for the Nobel topology. Specifically, the results for each of the 5 TMs are accounted for the three SDN node selection methods (HDF, HCC and HBC). Note that TM 5 is the peak TM in the data set and TM 1 is the non-peak TM.

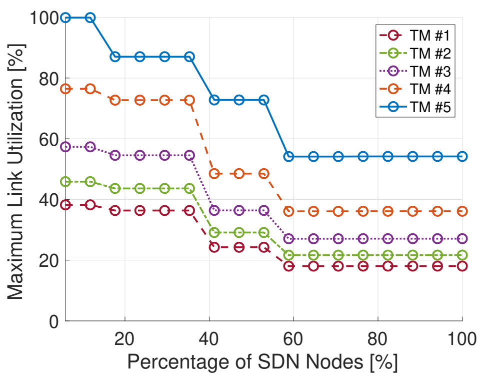
Fig. 8 shows a general trend. The MLU decreases with the SDN deployment rate for a small percentage of migrated SDN nodes. This MLU reduction is different depending on the considered selection method, with an early impact



(a) HDF



(b) HCC



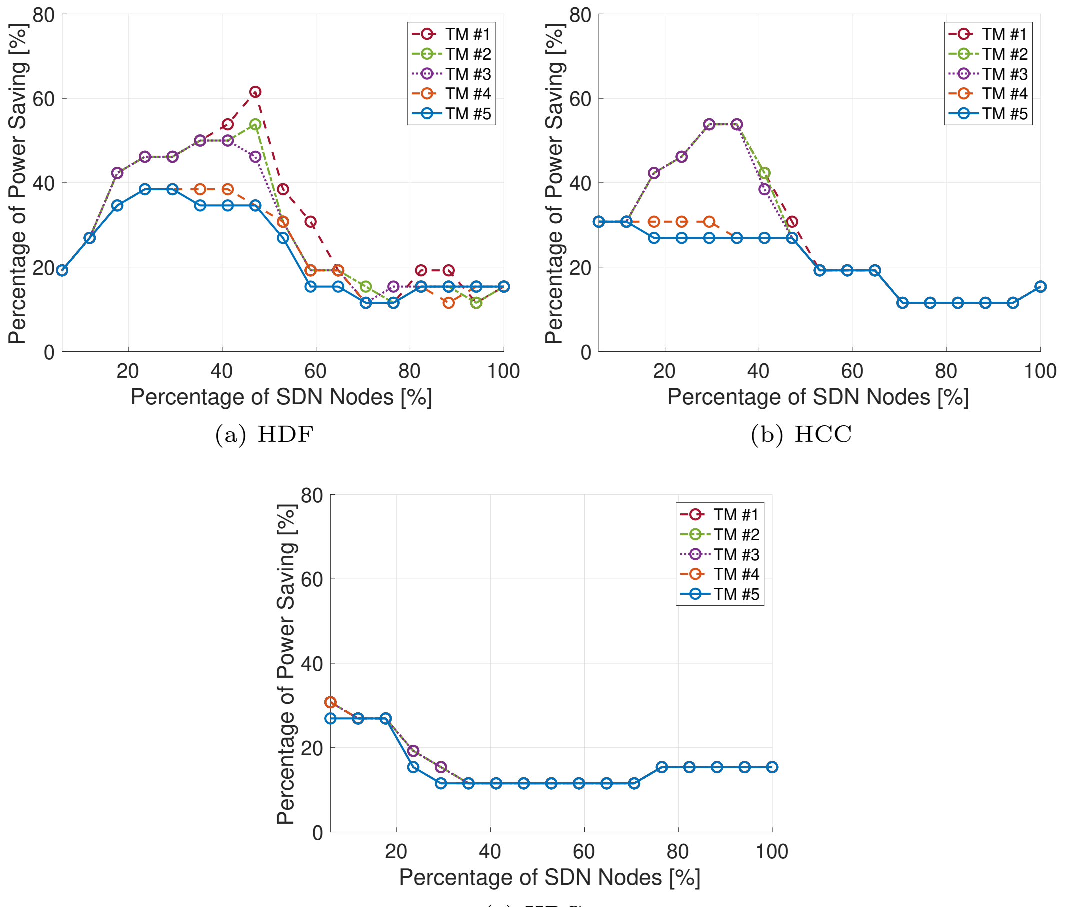
(c) HBC

Fig. 8: MLU value with respect to the percentage of SDN nodes for Nobel topology.

for HDF and HCC and a more progressive impact for HBC. After that, the MLU is stabilized, and adding SDN nodes does not impact the resulting MLU. Clearly, there is an existing direct proportional relationship between the overall traffic load that is injected into the network and the obtained MLU. TM 5 represents the worst-case scenario and TM 1 represents the case for non-peak traffic, where more energy saving opportunities can be found.

A considerable decrease in the obtained MLU is tested every time an IP node is replaced by an SDN node when the SDN deployment rate is below 40% both for HDF (Fig. 8a) and HCC (Fig. 8b). After such a percentage, the achieved MLU is approximately constant up to reaching the full SDN scenario (deployment rate of 100%). In other words, the highest impact during the transition from IP to SDN in terms of MLU is found in the range $(0, 40)\%$ of the deployment rate. It is in Fig. 8c where the MLU behavior is stepwise. In this case, the benefits of adding a new SDN node are not as high as in the case of HDF or HCC, regardless of the transition stage.

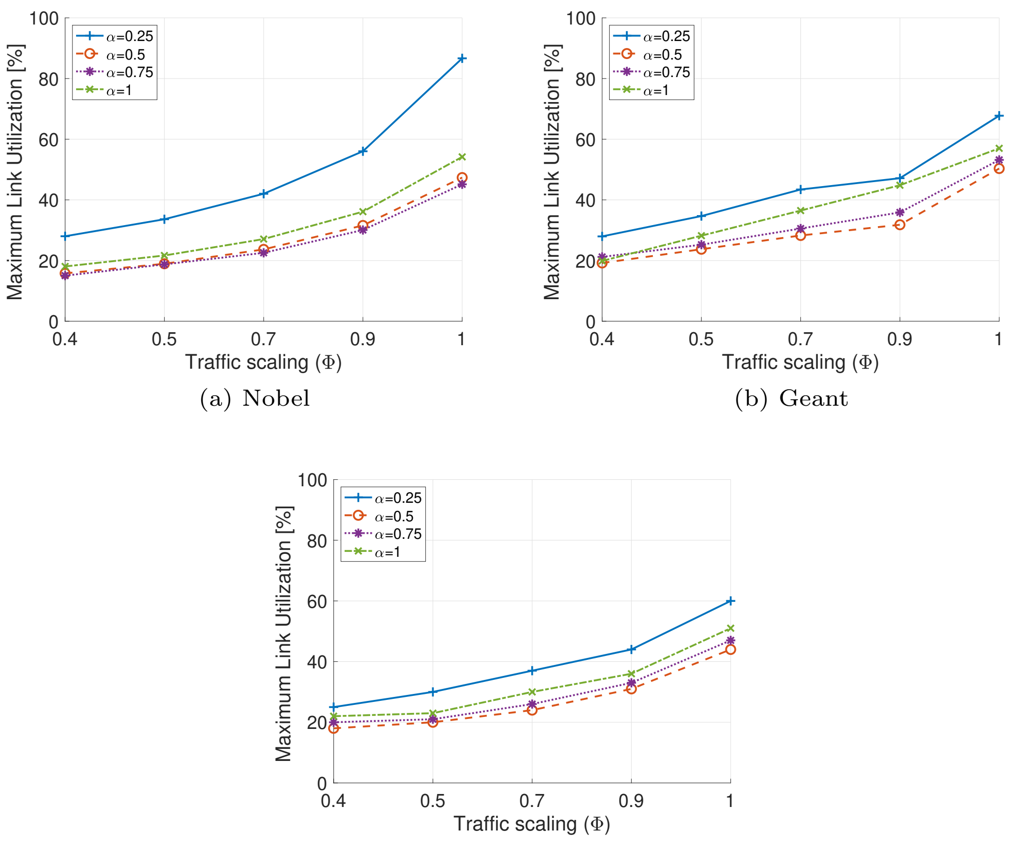
Once the evaluation of the obtained MLU has been carried out, we move our attention to the power savings achieved by the HSLA throughout of the IP/SDN transition process. In Fig. 9, the power savings obtained by L3A are reported for the 5 TMs and the three selection methods, considering different



(a) HDF (b) HCC (c) HBC

Fig. 9: Percentage of power savings obtained by L3A with respect to the percentage of SDN nodes for Nobel topology.

percentages of the SDN deployment rate. Considering the three subfigures, HDF is clearly the method that outperforms the other ones, reaching a power saving peak of approximately 50% when half of the network nodes are upgraded to SDN. In this case, the obtained trend is different from the one experienced when evaluating the MLU in Fig. 8: the power savings are increased with the SDN deployment rate up to reaching a threshold that slightly exceeds the half of the nodes considered as SDN. After such a value, the power savings sharply decrease, and no substantial differences can be found among traffic matrices and with the increase in the deployment rate. Although the HCC method slightly improves the power savings outcomes with regard to HDF for a deployment rate of 30% and low traffic scenarios (see Fig. 9b), it struggles when the traffic load is increased. Finally, HBC has been found to be the worst selection method for the power savings objective. In summary, the SDN selection method that presents the best results, both for the minimization of the MLU and the minimization of the power consumption, is HDF. An explanation for this outcome is derived by the fact that such a method aims at selecting the node with the highest degree, hence involving the maximum number of links and controllable paths.



(a) Nobel (b) Geant

(c) Germany50

Fig. 10: MLU value with respect to the traffic load for Nobel, Geant and Germany50 topologies considering HDF method.

4.5 Performance analysis over larger networks

The aim of the next analysis is to evaluate the results of the proposed solution over topologies with different sizes. Specifically, in Fig. 10 the MLU is reported as a function of the traffic scaling factor for the three topologies under test: Nobel, Geant and Germany50. For each topology, 4 deployment rate percentages (α) have been considered, approximating 25%, 50%, 75% and 100% of the nodes in the network. HDF is adopted as the node selection method. In the figures, it can be seen that the MLU increases with traffic, as previously discussed when analyzing Fig. 8 for the three topologies. It is worth to note that the best outcomes for the medium and large topologies are not obtained in the full SDN scenario ($\alpha = 100\%$) except for when half of the nodes are replaced (see Figs. 10b and 10c). In the Nobel case, the minimal MLU is experienced when $\alpha = 75\%$. Regardless, the difference with the half hybrid IP/SDN network is negligible.

Fig. 11 reports power savings outcomes as a function of the traffic load for the three considered topologies and the different values of α. By analyzing Fig. 11, there are differences between the results obtained for small and large topologies. Indeed, in the latter case (Geant and Germany50), a decreasing

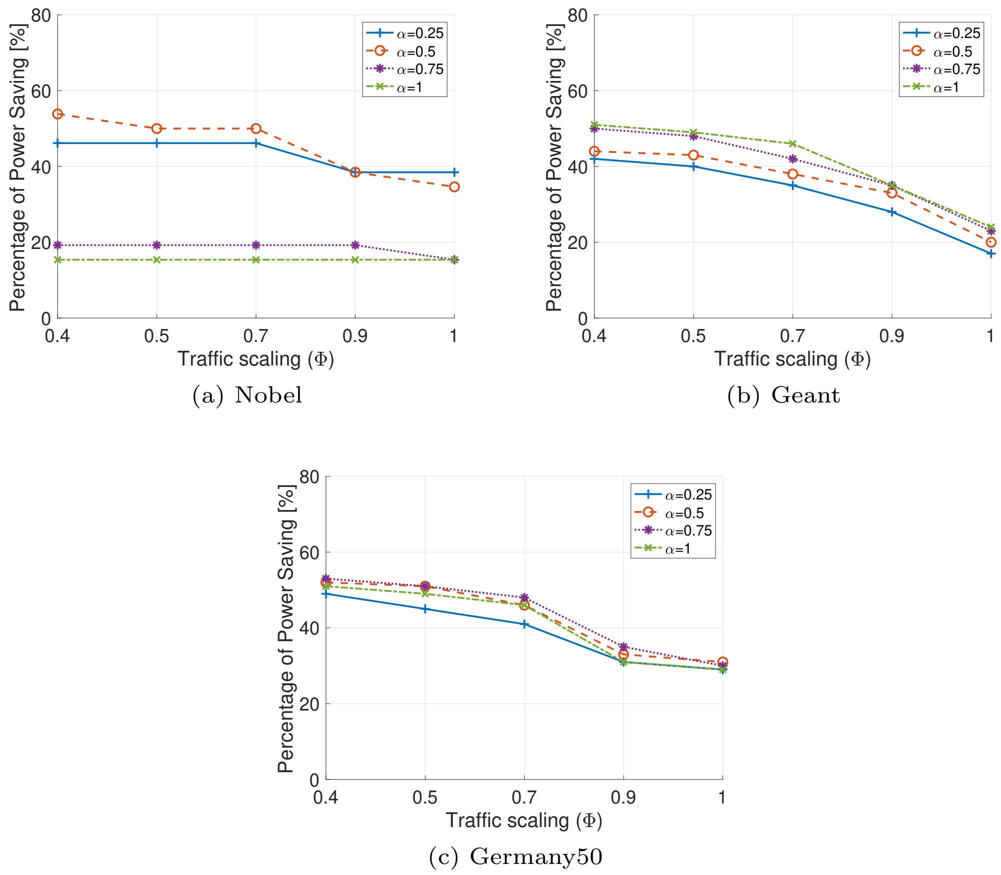
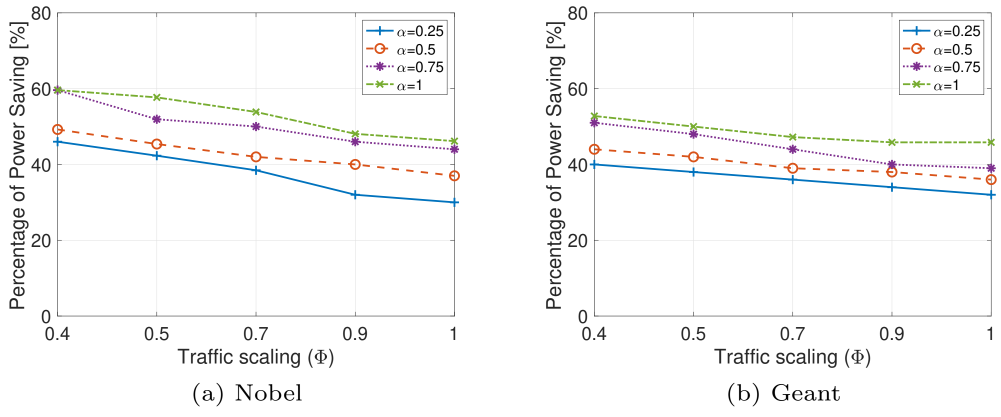


Fig. 11: Percentage of power savings obtained by L3A with respect to the traffic load for Nobel, Geant and Germany50 topologies considering HDF method.

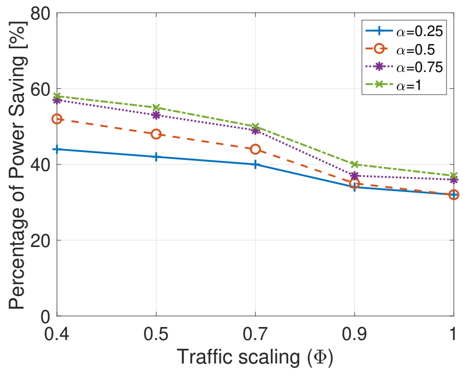
behavior in the power savings can be taken out with the increase in the traffic scaling factor. Therefore, a higher value of α is generally associated with power saving gains, especially for Geant. It is in the Nobel case (Fig. 11a) where the behavior is the opposite. The best results of power savings are obtained when $\alpha \leq 50\%$, as previously discussed when analyzing Fig. 9a. Once this threshold is overtaken, the power savings are considerably decreased, regardless of the network traffic load.

The explanation for the obtained power saving results is as follows. The proposed algorithm aims at either balancing the traffic load or to reduce the network power consumption. In the first stage, for the HSLA, the link weights are adjusted to create a large portion of the controllable traffic. As a consequence, the main role of the SDN nodes is to balance the traffic and not to reduce the power consumption (aim of the second stage, L3A). At the same time, the weight selection logic becomes less effective when the deployment ratio increases (approaching a full SDN network) since the amount of controllable traffic is already high. Thus, the proposed algorithm accomplishes its performance peak when the network is half IP-half SDN.

In the following, the other power saving approach that has been defined, namely, GA-LSO, is analyzed. Fig. 12 shows the power savings achieved by



(a) Nobel (b) Geant



(c) Germany50

Fig. 12: Percentage of power savings obtained by GA-LSO with respect to the traffic load for Nobel, Geant and Germany50 topologies.

GA-LSO as a function of the traffic scaling factor for the three considered topologies. In general, GA-LSO obtains better results than L3A, especially for small networks (see Figs. 11a and 12a for Nobel). The expected decreasing tendency of the power savings with the increase in the traffic load is confirmed for all cases, and the average power saving gain is 4.7% for a $50-50\%$ hybrid IP/SDN network ($\alpha = 0.5$), increasing up to 7.1% in the case of a full SDN migration ($\alpha = 1$). In the case of the Germany network, GA-LSO slightly struggles with regard to L3A for a small percentage of SDN nodes ($\alpha = 0.25$) and a low traffic load. This may be caused by the large size of the chromosomes in large networks (each link is represented by a gene) and the flexibility for the routing given by the low traffic flowing throughout the network.

4.6 Comparison with other solutions

In this subsection, we first introduce two benchmark solutions and then we compare their results with those obtained by the proposed heuristic. Finally, the results of the proposed solution are shown by means of a conducted performance analysis.

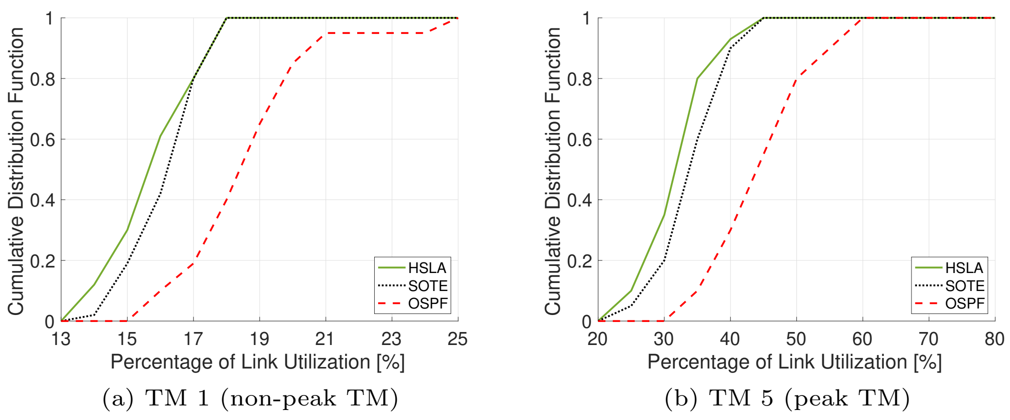


Fig. 13: Cumulative Distribution Function curves of link utilization for HSLA, SOTE and OSPF over Nobel topology considering TM 1 and TM 5.

The first algorithm we focus on to highlight the benefit allowed by using HLSA is SOTE [27]. SOTE tries to minimize the MLU of the hybrid IP/SDN network by jointly optimizing OSPF's weight settings for the whole network, thus balancing outgoing flows for IP equipment, as well as optimizing the traffic splitting ratio of flows aggregated at SDN nodes. However, SOTE does not consider energy efficiency optimization.

The second algorithm that is considered is HEATE [19]. HEATE's focus is the minimization of the power consumption of the hybrid IP/SDN network. Such a task is performed, similar to SOTE, by optimizing OSPF's link weight settings and SDN nodes' splitting ratio. However, HEATE does not consider the effects on the MLU caused by the minimization of energy consumption.

These two algorithms have been selected for comparison with the HSLA since they are both representative when facing the load balancing problem [27] and the energy consumption problem [19] in hybrid networks. Although works, such as [2] and [32], consider the joint problem of load balancing and energy efficiency in these scenarios, both of them are survey papers that review a set of works coping each type of problem ([2] considering also machine learning aspects), not the combined one. However, no proposals to compare with are provided.

Fig. 13 reports the results for the first analysis, where the Cumulative Distribution Function (CDF) curves of MLU are drawn for HSLA, SOTE and OSPF. The deployment ratio is set to $\alpha = 50\%$, i.e., a half hybrid IP/SDN network, and two scenarios with the nonpeak TM (TM 1, Fig. 13a) and the peak TM (TM 5, Fig. 13b) are considered. As shown in both figures, the HSLA achieves the lowest MLU compared with the rest of the solutions. SOTE, which is specifically designed to minimize the MLU, presents slightly worse results for the case of low traffic (Fig. 13a), with an average of a 1% difference in the MLU with regard to HSLA in approximately 80% of the links. In the case of a scenario with high traffic (see Fig. 13b), the outcomes obtained by HSLA are better than those reported by SOTE, with an average MLU reduction of

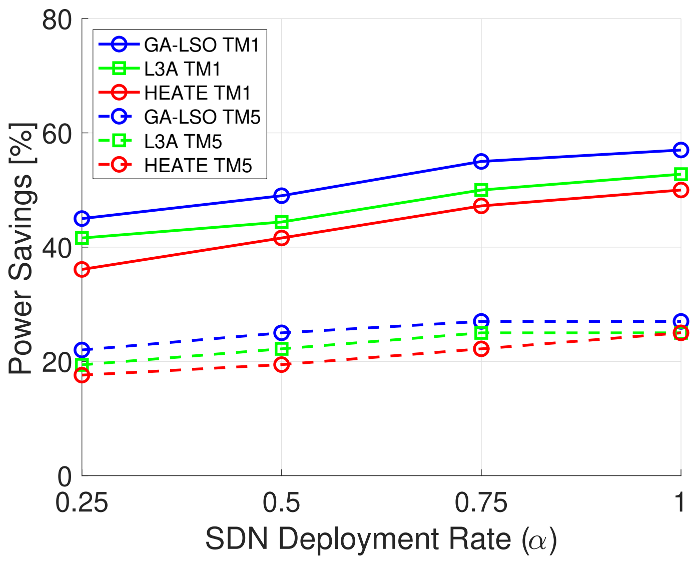


Fig. 14: Percentage of power savings achieved by GA-LSO, L3A and HEATE over Geant topology considering TM 1 and TM 5.

approximately 5% (note the difference in the x-axis between Figs. 13a and 13b). In general, the difference is that the HSLA allows for better load balancing in the less congested links with respect to SOTE for low traffic loads, while the gap is maintained throughout all the links in cases of high traffic loads. Furthermore, the OSPF protocol presents worse results than the HSLA and SOTE protocols because its objective is not to minimize the MLU, but to find the shortest path among each source-destination pair.

To compare the power saving gains among GA-LSO, L3A and HEATE, Fig. 14 shows the obtained results as a function of the SDN deployment rate for Geant topology. Two lines are reported for each solution for the peak and nonpeak TMs (TM 5 and TM 1). The figure shows that GA-LSO outperforms L3A and HEATE in all cases, with an average power saving gain of 7.7% with respect to HEATE and 4.2% when compared to L3A. In summary, the GA-based proposed solution presents better results than the ones obtained by ad hoc solutions that specifically cope a single objective.

5 Conclusion

The problems of load balancing and energy efficiency in hybrid IP/SDN networks are regarded by the state of the art as very relevant. However, the related literature treats them as completely separate problems, without considering the effects of powering off links on the MLU and vice versa. In this paper, we analyze the tradeoff between both problems. Starting from the opposite nature of the two optimization problems, a heuristic is proposed to jointly minimize the MLU and the network power consumption during the transition from IP to SDN networks. Through simulations on realistic network topologies, HSLA is compared with other state-of-the-art solutions, such as SOTE, which is intended only to minimize the MLU, while L3A and GA-LSO are compared to HEATE, which is aimed at solely minimizing the power consumption. Our proposed solution is shown to save more power than HEATE and outperforms SOTE in terms of link utilization. Moreover, it can provide up to 50% of MLU reduction and up to 60% of power savings. In the future, we plan to install the HSLA in a real SDN controller, such as Ryu or Faucet, to evaluate its impact on the network performance in a working test-bed.

Conflicts of interest

Not applicable.

Acknowledgments. This work has been partially funded by the Diputación de Cáceres (AV–7), the project RTI2018-094591-B-I00 (MCI/AEI/FEDER,UE), 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 and Infrastructure of the Government of Extremadura (GR21133, IB18030), and by the European Regional Development Fund.

References

[1] Cisco: Cisco Annual Internet Report (2018–2023). https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.pdf (2020)

[2] Etengu, R., Tan, S.C., Kwang, L.C., Abbou, F.M., Chuah, T.C.: Ai-assisted framework for green-routing and load balancing in hybrid software-defined networking: Proposal, challenges and future perspective. IEEE Access **8**, 166384–166441 (2020). https://doi.org/10.1109/ACCESS.2020.3022291

[3] Herrera, J.L., Polverini, M., Galán-Jiménez, J.: A machine learning-based framework to estimate the lifetime of network line cards. In: NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium, pp. 1–5 (2020). https://doi.org/10.1109/NOMS47738.2020.9110455

[4] Heegaard, P.E., Trivedi, K.S.: Network survivability modeling. Comput. Netw. **53**(8), 1215–1234 (2009). https://doi.org/10.1016/j.comnet.2009.02.014

[5] Bolla, R., Bruschi, R., Davoli, F., Cucchietti, F.: Energy efficiency in the future internet: A survey of existing approaches and trends in energy-aware fixed network infrastructures. IEEE Communications Surveys Tutorials **13**(2), 223–244 (2011). https://doi.org/10.1109/SURV.2011.071410.00073

[6] Chiaraviglio, L., Mellia, M., Neri, F.: Reducing power consumption in backbone networks. In: 2009 IEEE International Conference on Communications, pp. 1–6 (2009). https://doi.org/10.1109/ICC.2009.5199404

[7] Phillips, C., Gazo-Cervero, A., Galán-Jiménez, J., Chen, X.: Pro-active energy management for wide area networks. IET Conference Proceedings, 317–3225 (2011)

[8] Galán-Jiménez, J., Polverini, M., Cianfrani, A.: Reducing the reconfiguration cost of flow tables in energy-efficient software-defined networks. Computer Communications **128**, 95–105 (2018). https://doi.org/10.1016/j.comcom.2018.07.022

[9] Galán-Jiménez, J., Berrocal, J., Herrera, J.L., Polverini, M.: 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), pp. 33–37 (2020). https://doi.org/10.1109/NoF50125.2020.9249089

[10] Agarwal, S., Kodialam, M., Lakshman, T.V.: Traffic engineering in software defined networks. In: 2013 Proceedings IEEE INFOCOM, pp. 2211–2219 (2013). https://doi.org/10.1109/INFCOM.2013.6567024

[11] Akyildiz, I.F., Lee, A., Wang, P., Luo, M., Chou, W.: A roadmap for traffic engineering in sdn-openflow networks. Computer Networks **71**, 1–30 (2014). https://doi.org/10.1016/j.comnet.2014.06.002

[12] Mendiola, A., Astorga, J., Jacob, E., Higuero, M.: A survey on the contributions of software-defined networking to traffic engineering. IEEE Communications Surveys Tutorials **19**(2), 918–953 (2017). https://doi.org/10.1109/COMST.2016.2633579

[13] Hong, D.K., Ma, Y., Banerjee, S., Mao, Z.M.: Incremental deployment of sdn in hybrid enterprise and isp networks. In: Proceedings of the Symposium on SDN Research. SOSR '16. Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2890955.2890959

[14] Huang, X., Cheng, S., Cao, K., Cong, P., Wei, T., Hu, S.: A survey of deployment solutions and optimization strategies for hybrid sdn networks. IEEE Communications Surveys Tutorials **21**(2), 1483–1507 (2019). https://doi.org/10.1109/COMST.2018.2871061

[15] Amin, R., Reisslein, M., Shah, N.: Hybrid sdn networks: A survey of existing approaches. IEEE Communications Surveys Tutorials **20**(4), 3259–3306 (2018). https://doi.org/10.1109/COMST.2018.2837161

[16] Giroire, F., Moulierac, J., Phan, T.K.: Optimizing rule placement in

software-defined networks for energy-aware routing. In: 2014 IEEE Global Communications Conference, pp. 2523–2529 (2014). IEEE

[17] Wang, H., Li, Y., Jin, D., Hui, P., Wu, J.: Saving energy in partially deployed software defined networks. IEEE Transactions on Computers **65**(5), 1578–1592 (2015)

[18] Huin, N., Rifai, M., Giroire, F., Pacheco, D.L., Urvoy-Keller, G., Moulierac, J.: Bringing energy aware routing closer to reality with sdn hybrid networks. IEEE Transactions on Green Communications and Networking **2**(4), 1128–1139 (2018)

[19] Wei, Y., Zhang, X., Xie, L., Leng, S.: Energy-aware traffic engineering in hybrid sdn/ip backbone networks. Journal of Communications and Networks **18**(4), 559–566 (2016). https://doi.org/10.1109/JCN.2016.000079

[20] Saha, N., BERA, S., Misra, S.: Sway: Traffic-aware qos routing in software-defined iot. IEEE Transactions on Emerging Topics in Computing **9**(1), 390–401 (2021). https://doi.org/10.1109/TETC.2018.2847296

[21] Hopps, C.: Analysis of an equal-cost multi-path algorithm. RFC **2992**, 1–8 (2000)

[22] Luong, N.C., Hoang, D.T., Gong, S., Niyato, D., Wang, P., Liang, Y., Kim, D.I.: Applications of deep reinforcement learning in communications and networking: A survey. IEEE Communications Surveys Tutorials **21**(4), 3133–3174 (2019). https://doi.org/10.1109/COMST.2019.2916583

[23] Lin, T., Kuo, C., Chang, H., Chang, W., Lin, Y.: A parameterized wildcard method based on sdn for server load balancing. In: 2016 International Conference on Networking and Network Applications (NaNA), pp. 383–386 (2016). https://doi.org/10.1109/NaNA.2016.74

[24] Agarwal, S., Kodialam, M., Lakshman, T.: Traffic engineering in software defined networks. In: 2013 Proceedings IEEE INFOCOM, pp. 2211–2219 (2013). IEEE

[25] Ren, C., Bai, S., Wang, Y., Li, Y.: Achieving near-optimal traffic engineering using a distributed algorithm in hybrid sdn. IEEE Access **8**, 29111–29124 (2020)

[26] Guo, Z., Dou, S., Wang, Y., Liu, S., Feng, W., Xu, Y.: Hybridflow: Achieving load balancing in software-defined wans with scalable routing. IEEE Transactions on Communications **69**(8), 5255–5268 (2021)

[27] Guo, Y., Wang, Z., Liu, Z., Yin, X., Shi, X., Wu, J., Xu, Y., Chao, H.J.:

Sote: Traffic engineering in hybrid software defined networks. Computer Networks **154**, 60–72 (2019). https://doi.org/10.1016/j.comnet.2019.03.008

[28] Galán-Jiménez, J., Polverini, M., Lavacca, F.G., Herrera, J.L., Berrocal, J.: On the tradeoff between load balancing and energy-efficiency in hybrid ip/sdn networks. In: 2021 12th International Conference on Network of the Future (NoF), pp. 1–9 (2021). https://doi.org/10.1109/NoF52522.2021.9609876

[29] Cianfrani, A., Eramo, V., Listanti, M., Polverini, M., Vasilakos, A.V.: An ospf-integrated routing strategy for qos-aware energy saving in ip backbone networks. IEEE Transactions on Network and Service Management **9**(3), 254–267 (2012). https://doi.org/10.1109/TNSM.2012.031512.110165

[30] Ahn, C.W., Ramakrishna, R.S.: A genetic algorithm for shortest path routing problem and the sizing of populations. IEEE Transactions on Evolutionary Computation **6**(6), 566–579 (2002). https://doi.org/10.1109/TEVC.2002.804323

[31] Orlowski, S., Pióro, M., Tomaszewski, A., Wessäly, R.: SNDlib 1.0–Survivable Network Design Library. In: Proceedings of the 3rd International Network Optimization Conference (INOC 2007), Spa, Belgium (2007). http://sndlib.zib.de, extended version accepted in Networks, 2009.

[32] Etengu, R., Tan, S., Abbou, F., Lee, C., Yusoff, Z., Shahbe, M.: Hybrid software-defined networking traffic scheduling: Energy-aware load balancing perspective. In: Conference of Open Innovations Association, FRUCT, pp. 452–457 (2019). FRUCT Oy

Appendix E

Fog Node Placement in IoT Scenarios with Stringent QoS Requirements: Experimental Evaluation

ID	: **ICC 2021**
Authors	: Juan Luis Herrera, Paolo Bellavista, Luca Foschini, José García-Alonso, Jaime Galán-Jiménez, Javier Berrocal
Publication type	: Conference full-paper
Conference	: IEEE International Communications Conference (ICC) 2021
DOI	: `https://doi.org/10.1109/ICC42927.2021.9500900`
Conference rating	: International conference, GGS Class 2 (A)

Fog Node Placement in IoT Scenarios with Stringent QoS Requirements: Experimental Evaluation

Juan Luis Herrera*, Paolo Bellavista†, Luca Foschini†, José García-Alonso*,
Jaime Galán-Jiménez* and Javier Berrocal*
*Department of Computer Systems and Telematics Engineering, University of Extremadura, Spain.
†Dipartamento di Informatica-Scienza e Ingegneria, University of Bologna, Italy.
[jlherrerag, jaime, jgaralo, jberolm]@unex.es, [paolo.bellavista, luca.foschini]@unibo.it

***Abstract*—Leveraging the Internet of Things (IoT) in intensive domains, such as in the Industrial Internet of Things (IIoT) or Internet of Medical Things (IoMT), provides automation and sensing solutions for complex environments through the interconnection of different sensors and actuators. However, these scenarios usually demand to meet stringent Quality of Service (QoS) requirements to work properly. Fog computing, a paradigm that brings computation and storage closer to the edge, and Software-Defined Networking (SDN), a networking paradigm that enables for network scalability and flexibility, can be combined. To do so, fog nodes that integrate both, computation resources and SDN capabilities, are leveraged to meet these stringent needs. Clearly, the placement of such fog nodes plays a key role in the achieved QoS. In this paper, an optimal fog node placement formulation is evaluated in an emulated fog and SDN environment. Results show that an optimal fog node placement can achieve a reduction of up to 59% in the network latency with a minimal jitter compared with other well-known placement methods.**

I. Introduction

The Internet of Things (IoT) has emerged in recent years as a means to interact with the physical world from computer applications. Through sensors and actuators, IoT devices enable applications to receive inputs and to send outputs to the real world, respectively. The connection of IoT devices to the Internet allows for remote interactions. Applications may not run on the IoT devices themselves and may run at a remote computer instead, leveraging an Internet connection to communicate with these devices. These capabilities enable Cyber-Physical Systems (CPS): hybrid systems that apply *cyber* applications such as data processing or storage to *physical* environments, leveraged in domains such as industry or healthcare [1]. Applying IoT to these domains has also motivated the creation of terms such as the Industrial Internet of Things (IIoT) or the Internet of Medical Things (IoMT).

Leveraging CPS in IIoT or IoMT environments is not a simple task: the intensive nature of these domains require a stringent Quality of Service (QoS) for their IoT applications. For instance, IIoT Factory Automation applications [2] or IoMT machine learning applications [3] are very time-strict, sometimes even requiring to be executed in cycles of less than one millisecond. The cloud computing environment, which is widely used in user-grade IoT systems [4], involves the full execution of IoT applications in *cloud servers*, often physically placed far away from the set of end IoT devices. The physical distance between IoT devices and servers is translated into an increase in the network latency, which results in a restraint of the stringent QoS requirements. The main motivation of paradigms such as fog computing is precisely to leverage servers that are closer to IoT devices as a way to reduce the network latency, thus enhancing the QoS and enabling intensive, QoS-strict IoT applications [5].

In IIoT and IoMT environments, scalability and flexibility are also a major requirement [2]. The network needs to be scalable and flexible, as well as to meet the required QoS requirements. In this way, Software-Defined Networking (SDN) networks can act as a perfect enabler to provide IIoT and IoMT applications with the required scalability and flexibility [6].

Therefore, a potential solution to this problem is the combination of servers close to IoT devices, using paradigms such as fog computing, with an underlying SDN network in order to meet the requirements of these applications: stringent QoS levels, flexibility and scalability. Some of the authors of this work previously presented a proposal for a combination of fog computing with SDN: in [7], a specific Fog Node (FN) device, which combines an SDN switch with light virtualization capabilities, is presented. The FN is able to execute computational workloads for IoT applications with extensive workloads, and thus simplifies obtaining the required QoS. At the same time, its SDN capabilities [7] allow the FN to be integrated into infrastructures exploiting the SDN technology.

Tests of different QoS metrics such as jitter or latency are also included in [7], placing the FN in either the local or a remote network. However, no experimental evaluation is performed to compare the outcomes of the QoS metrics when the FN is placed at different parts of the local network. Moreover, no clarifications about the optimal placement of the FNs are provided in order to achieve optimal QoS. Furthermore, it does not consider that the FN, unlike an elastic cloud server, has a well-defined limit to its computational power and storage, i.e., a limited throughput. If an IoT application was to exceed this throughput, the FN would not be able to properly function.

The solution for this situation is to place additional FNs in the infrastructure and to divide the requests of IoT applications among these nodes, in such a way that the throughput of FNs is not exceeded. The division can be performed through a mapping: each IoT device can be *mapped* to a FN, so that the IoT device can send its requests exclusively to its mapped FN. This allows the infrastructure to become more robust, while at the same time requiring for the mapping to be performed in such a way so that QoS is optimized. Furthermore, this requires an optimal placement of multiple FNs.

Thus, while the needs of QoS-stringent IoT applications can be met through the use of fog computing and SDN, concretely by adding FNs to an SDN architecture, this is not enough to merely leverage these technologies to obtain an optimal QoS. FNs need to be placed in such a way that the QoS of each IoT application is optimized, and an optimal mapping between FNs and IoT devices is also required. The problem of optimally placing a set of FNs in an SDN network while optimally providing a device-to-FN mapping is defined as the Fog Node Placement Problem (FNPP). It is important to note that the FNPP is a problem that must be solved at the infrastructure level, not at the application level. In this work, we propose and experimentally evaluate the optimal solution for the FNPP.

Therefore, the main contributions of this paper are:

- The formalization of the FNPP.
- The evaluation of the solution in the Mininet network emulator.
- An analysis of the experimental evaluation results.
- The comparison of the FNPP optimal solution with alternative and well-known placement strategies.

The remainder of this paper is structured as follows. Sec. II presents the motivation for the FNPP. The system model used in the FNPP is explained in Sec. III. Following this model, a MIP formulation of the FNPP solution is presented in Sec. IV, followed by an experimental evaluation in Sec. V. Finally, Sec. VI concludes.

II. Motivation

The motivation behind solving the FNPP is rooted on the interest of bringing IoT to intensive domains, such as the advances in IIoT or IoMT scenarios. Although IoT, and its role as an enabler for CPS, can bring interesting improvement to industry or healthcare [2], [3], these systems need a very high QoS to work properly.

In order to meet these stringent QoS requirements, two factors should be accounted for: computing and networking QoS need to be high enough, i.e., low execution time and low latency, respectively. Through fog computing, computation resources can be brought closer to IoT devices, thus providing them access to a higher computing QoS with also a higher networking QoS [5]. Furthermore, it is desirable for the underlying network to be an SDN network in order to improve the scalability and flexibility of the overall infrastructure [2].

The FN proposed in [7] is designed specifically to meet these stringent QoS needs, while also bridging SDN and fog computing. The SDN capabilities of the FN enable for its introduction into flexible and scalable SDN networks, while its computational power allows for workloads to be executed in the local network, providing a level of QoS that would be difficult to replicate using cloud computing approaches.

Although this level of QoS is high, it is interesting to obtain an optimal QoS, since a sub-optimal level of QoS might not be enough for the stringent requirements of these applications. Since the placement of the FN affects the QoS of the applications that run on it, the FNPP must be solved to obtain an optimal QoS. This is the main motivation for this paper: to provide an optimization of the QoS provided by FNs, which are used as enablers for both fog computing and SDN in the infrastructures. Our goal is, thus, to provide a tool that enables administrators to place FNs optimally.

Some other works related to optimization of QoS in SDN and fog infrastructure have also been found in literature as research topics that remain active [4], [8], [9]. Some works have tackled the optimization of QoS through the placement of SDN network equipment, namely, an SDN controller [8]. Other approaches optimize the QoS by placing services in fog infrastructures, either in a way that least dissatisfies the QoS requirements of applications [9] or in one that optimizes QoS by finding optimal nodes to run services on [10]. The solutions of the latter approaches can be integrated with SDN in order to be transparently applied [10].

The main differences between the solution proposed in this work and the previously cited related works are centered around the focus. The FNPP is focused on optimally placing a set of FNs in an SDN infrastructure to support QoS-stringent IoT applications, instead of being focused on placing services in already placed FNs or optimizing the QoS of the SDN network.

III. System model

To illustrate the model used in the FNPP, its usage in a sample application is shown in this section. Concretely, an IoMT application for the processing of electrocardiogram (ECG) data is considered. Despite the example, the FNPP model is generic and applies to a broader range of applications than only IoMT ones. This application is based on the example model introduced in [11]. In such application, there is a service that contains a deep learning model that is able to detect anomalies in the heartbeat of an user. Users carry ECG monitors that constantly send their sensor information to the deep learning service. This service checks if the heartbeat is abnormal or not and, if an abnormal heartbeat is detected, the system is able to raise an alarm. In the particular example presented next, four ECG monitors are considered in the topology shown in Fig. 1.

This system involves almost real-time monitoring [11], and thus there must be minimal latency between the ECG sensors and the deep learning service, i.e., it has stringent QoS needs. This makes the deep learning service a very good candidate for running in a FN, since it is key for the QoS of the application and, furthermore, the original model also recommends to run it in the fog [11]. Thus, a FN must be placed in the infrastructure.

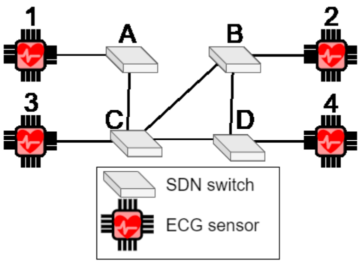


Figure 1: Example system model.

Since the FN contains an SDN switch, it is possible to replace an SDN switch by a FN to make the infrastructure fog/SDN-enabled. However, it should not be randomly placed. Application QoS requirements should be considered to assess its placement.

For simplicity, latency can be understood in terms of *hops*. The more hops traffic has to go through to reach the FN, the higher the latency is and, thus, the lower the QoS is. Since the ECG monitoring application has a very stringent QoS requirement [11], it is considered that it only admits latencies of one hop or less. If switch A is replaced by a FN, only sensors 1 and 3 would have a low enough latency. If B or D are replaced, sensors 2, 3 and 4 would meet the requirement, but sensor 1 would not. However, if switch C is replaced, all sensors are one hop or less away. Therefore, C is the optimal placement for the FN: the solution to the FNPP.

It is also possible to consider the placement of two FNs: some sensors will send their heartbeat to one FN, while the rest of the sensors will send it to the other one. While this approach makes it easier to place the FNs, there is an added complexity, since a mapping between sensors and FNs must also be obtained. For instance, if FNs are placed in switches A and B, but sensor 2 sends its heartbeat to the FN in A and the rest send their heartbeat to the FN in B, only sensor 4 would meet the latency requirement. For the same FN placement, however, it is possible to consider an alternate mapping so that sensors 1 and 3 send their heartbeat to the FN in A and sensors 2 and 4 send theirs to the FN in B. This mapping is the optimal one and meets the latency requirement as well.

As shown in the model, the placement of FN nodes and the mapping between FNs and IoT devices are relevant in the QoS obtained on IoT applications. When the QoS requirements are stringent, it is desirable to strive for an optimal placement and an optimal mapping, i.e., to obtain a solution by solving the FNPP. The FNPP is very similar to a well-known NP-hard problem: the facility location problem [12]. Thus, although it is feasible to solve the FNPP manually in small cases, an exponential growth is experienced for large scenarios. Therefore, a generic and automatic way to provide a solution is required.

IV. Problem formulation

Our proposal for a generic solution to the FNPP is a Mixed Integer Programming (MIP) formulation of the solution. This formulation can then be adapted to particular scenarios and fed to software MIP solvers, automating its usage. MIP formulations can be used to solve network problems that affect performance, as shown in [13].

The scenario in this FNPP formulation is a network topology, modeled as an undirected graph $G = \{V, L\}$, with V vertices and L edges. Links are modeled as edges, so that the link between vertices $i \in V$ and $j \in V$ is modeled as $l_{ij} \in L$. Link capacity and latency are also part of the model, so that C_{ij} is the capacity of l_{ij} and $\beta_{l_{ij}}$ is its latency. According to the system model presented in Sec. III, a vertex $v \in V$ can either be a IoT device (host) or an SDN switch.

In order to follow the system model presented in Sec. III and guarantee a feasible formulation, we assume that:

- FNs can only process a limited amount of traffic per unit of time, i.e., they have a capacity.
- The combined capacity of all FNs is enough to process all the traffic in the network.
- The capacity of a given FN is at least as high as the traffic produced by the IoT device that sends the highest volume of traffic.

Thus, vertices can be split so that $V = H \cup S; H \cap S = \emptyset$: H are IoT devices, i.e. hosts; and S are SDN switches. The objective of the FNPP is to select a subset of S, of a given size θ, to place FNs at, and to map a subset of H to each of these FNs. On the one hand, each switch is modeled to have a processing latency of β_S. On the other hand, and following Sec. III, IoT devices send an amount of traffic, which we label ϕ_h for a given host h. The FN mapped to h is the one in charge of processing its traffic, given that any FN has the capacity to process up to α traffic.

After these parameters have been defined, decision variables follow. To calculate optimal paths, flow variables are used: let $f^h_{ij} \forall h \in H; l_{ij} \in L$ be a binary variable that takes a value of 1 if the traffic sent by host h traverses link l_{ij} and 0 otherwise. In order to place the FNs as well, let $X_s, s \in S$ be a binary variable that is 1 if a FN is placed on switch s and 0 otherwise. Finally, to map hosts and FNs, let $Y_{hs}, h \in H; s \in S$ be a binary variable that becomes 1 if host h is mapped to the FN placed in switch s and 0 otherwise.

The objective of the FNPP is to minimize the average latency between hosts and FNs as a manner to optimize QoS. Thus, let $L(h) = (\sum_{l_{ij} \in L} f^h_{ij} \beta_{l_{ij} \beta_S}) - \beta_S$ be the latency between a host h and its mapped FN, accounting for the latency of all links and the processing latency of all switches in the route. A set of notations is summarized in Table I for easy reference.

Given these definitions, the FNPP can be formulated as:

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

subject to:

$$i \in V, h \in H : \sum_{j \in V} f^h_{ij} - f^h_{ji} = \begin{cases} 1 & \text{if } i = h \\ -Y_{hi} & \text{otherwise.} \end{cases} \tag{2}$$

Table I: List of 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
$\beta_{l_{ij}}$	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
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}

$$\forall l_{ij} \in L : \sum_{h\in H} f_{ij}^h \phi_h \leq C_{ij} \tag{3}$$

$$\sum_{s\in S} X_s = \theta \tag{4}$$

$$\forall s \in S : \sum_{h\in H} \phi_h Y_{hs} \leq \alpha X_s \tag{5}$$

$$\forall h \in H : \sum_{s\in S} Y_{hs} = 1 \tag{6}$$

$$\forall h \in H, s \in S, l_{ij} \in L : X_s, Y_{hs}, f_{ij}^h \in \{0, 1\} \tag{7}$$

The objective is to minimize the average latency between hosts and FNs, as per (1). Their paths are calculated using the classical flow conservation constraint of (2), while (3) guarantees that link capacity is not surpassed. The amount of nodes to be placed is the given amount θ, as (4) shows. To control the capacity of FNs as well as forcing IoT devices to only map themselves to FNs that are placed, (5) is also a constraint, knowing that, as of (6), each IoT device is mapped to a single FN. Finally, (7) makes variables binary.

This formulation of the FNPP can now be parameterized to cater to the needs of different network topologies with different FN capacities, different amounts of traffic sent by FNs or different latencies in their network components. It also guarantees an optimal placement, routing and mapping between FNs and IoT devices, and thus, an optimization for QoS-strict IoT applications.

V. Performance evaluation

This section presents a performance evaluation of the FNPP formulations under different circumstances. Evaluation has been carried out in an emulated environment, and includes results on latency, average path length, average FN usage and execution time with different number of FNs and different placement methods.

A. Evaluation environment

In order to analyze the scalability of the proposed solution, three different network topologies have been considered: small (25 switches and 24 links), medium (40 switches and 39 links) and large (70 switches and 69 links). These topologies were synthetically generated using the Erdős-Rényi model [14]. Traffic matrices were also synthetically generated using the model reported in [15]. Other details such as the latencies of the switches were empirically extracted from the Mininet SDN network emulator [16], while the value of α was set following Eq. (8) to guarantee a feasible solution, i.e., all FNs have enough throughput to manage all their incoming traffic.

$$\alpha = max(max(\phi_h \forall h \in H), \frac{\sum_{h\in H} \phi_h}{\theta}) \tag{8}$$

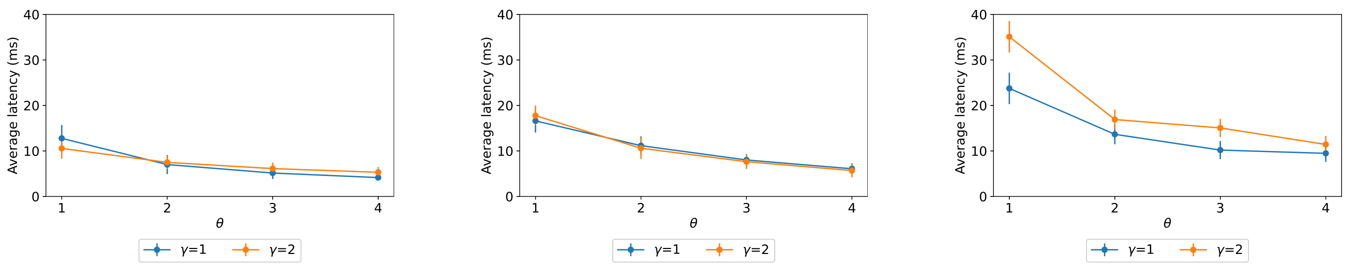
In order to emulate the set of FNs in the network, θ additional hosts have been added and placed according to the output of the corresponding placement solution. *Iperf* tool has been used to create client-server traffic flows from a host to its mapped FN according to the values of the traffic matrix. Latency has been assessed by sending 30 probes with the *ping* command from each host to its mapped FN. Simulations have been performed in Mininet's virtual machine [16], featuring 8 GB of RAM and 4 CPU cores of an Intel i7-8565U CPU.

In order to analyze the impact of adding FNs to the network, values of $\theta \in [1, 4] \in \mathbb{N}$ have been considered. At the same time, in order to analyze the impact of the variation in the number of hosts, a parameter that sets the number of hosts that are connected to each switch, namely γ, has been defined. In particular, we have carried out simulations for $\gamma = \{1, 2\}$. It is important to remark that, while different traffic matrices were generated for each scenario, the total amount of traffic in the network is always the same, regardless of γ.

B. Performance analysis

The objective of the first performed analysis is to evaluate the optimization objective, i.e. the average latency, by varying the number of FNs (θ) and hosts per switch (γ). At first, Fig. 2 shows the average latency as a function of θ for the small, medium and large topologies. Error bars represent the average jitter for each type of test.

By looking at the figures, a similar trend is experienced in all topologies, with the decrease of the average latency as soon as the number of FNs is increased. Concretely, latency lowers in a negative log likelihood as more FNs are added. This is because, although adding more FNs implies that their placement can get closer to their mapped IoT devices, thus reducing latency, this difference is most noticeable when FNs were far from their mapped devices and they can be placed closer. When FNs are already close, placing them even closer has an undeniable effect, but not as large as before. This is also reflected by the experienced jitter represented by the error bars: a higher value of θ is associated with a lower value of the experienced jitter. Clearly, the larger the topology is, the higher average latency is experienced. This outcome is caused due to the fact that a larger topology generally implies longer



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

Figure 2: Average latency vs θ.

paths from IoT devices to FNs. Another conclusion we extract from Fig. 2 is that the impact of γ is not very significant in the small and medium topologies, thus, the infrastructure remains scalable as soon as more hosts are added. However, a higher impact is seen in larger topologies, especially in low values of θ, while also showing the same trend.

The second analysis is a comparison between the results obtained using the MIP formulation defined in Sec. IV and other well-known placement methods based on graph algorithms, namely, placing FNs in the nodes with the Highest Betweenness Centrality (HBC) or the Highest Closeness Centrality (HCC). When these methods are used, the mappings between hosts and FNs are also assessed by using the formulation, and thus only the placement of the FN is compared. Results can be seen in Fig. 3, which again reports the average latency as a function of θ for the different topologies. Overall, results show that the MIP formulation, labeled *Optimal*, consistently achieves the best results in all the topologies and placement methods. Moreover, it also shows that, for the specific case of $\theta = 1$, some of these methods can show very similar results. HCC in Fig. 3a and HBC in Figs. 3b and 3c provide very similar latency values compared to the optimal solution, with less than 1 ms of difference. However, there are two main benefits to the usage of the optimal method. First, the optimal method consistently exhibits the lowest latency. While alternate methods may also yield a similar latency in some cases, it is not consistently the same one: sometimes it is HBC, sometimes it is HCC, as opposed to the formulation, which is consistent. The second benefit is that, for values of $\theta \geq 2$, the performance gap between the optimal solution and these alternate methods heavily increases. The largest gap can be found in $\theta = 3$ in Fig. 3c, with HCC providing a latency more than twice as large (144% larger) than the one provided by Optimal. Along with the previously shown benefits, the conclusion is that the MIP formulation is preferred in all cases because of its consistency and performance.

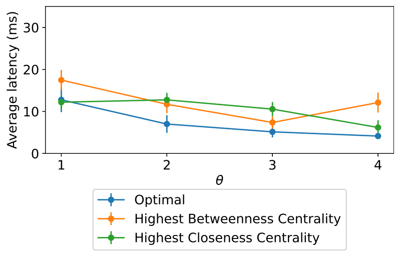
Fig. 4 shows an analysis of the path length between IoT devices and FNs. Due to lack of space, and since all topologies derive similar results, only the large topology is shown. The drawn conclusions are very similar to the ones extracted from Fig. 3c: Highest Closeness Centrality yields the worst results, followed by Highest Betweenness Centrality. Indeed, the optimal formulation clearly yields much better results: although the minimum path length is the same value for all three placement methods, even the maximum path length for Optimal (13 hops) is less than half the average path length for HBC (28 hops). Therefore, the impacts in latency shown in Fig. 3 are mainly caused by longer paths. Another interesting conclusion is that the MIP formulation assesses more balanced paths, as the differences between minimum, average and maximum path lengths are much lower in proportion than the ones reported for the rest of placement methods.

The analysis shown in Fig. 5 tests how load is balanced of all three methods in the large topology. To do so, the median FN usage is shown: optimally, it should decrease in a harmonic progression, while lower or higher values imply an unbalance. The lower or higher the values are compared to a harmonic progression, the more unbalanced it is. While all of the considered placement methods show equally balanced loads in most of the tests, the results in $\theta = 3$ shows that the median usage in Optimal is 30.5%, a nearly perfect balance with only 2.8% of difference to the harmonic progression. However, the rest of the methods are over well 40%, with more than a 10% of difference to the harmonic progression, showing a clearly unbalanced distribution. This means that, even with optimal mapping, sub-optimal FN placement derives in sub-optimal load balancing as well.

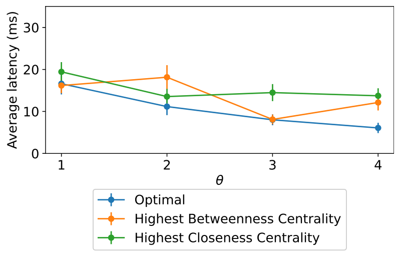
In a final analysis, Fig. 6 shows the time required to solve the FNPP with $\theta = 4$ in all the considered topologies with all the proposed methods. The main conclusion is that the MIP formulation is more time-consuming than the alternate methods, as expected. Furthermore, Fig. 6 also shows an almost exponential growth with the topology size, which is to be expected of an NP-hard problem optimization. Such growth is smaller in other methods, as the time gap between them also grows with the topology size. However, even in the largest topology, the formulation takes 5.60 seconds, which can be considered as tractable time.

VI. Conclusions and Future Work

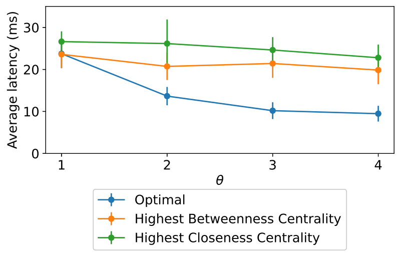
IoT has a great potential in intensive domains such as industry or healthcare, and with this potential come stringent QoS requirements that are difficult to meet. In order to leverage fog computing and SDN, FNs should be correctly placed to provide optimal placement, hence. In this paper, the FNPP has been introduced, formulated and solved. The performance of the solution has been evaluated, analyzed and compared against other methods in the Mininet network emulator.



(a) Small topology.



(b) Medium topology.



(c) Large topology.

Figure 3: Average latency with different placement methods.

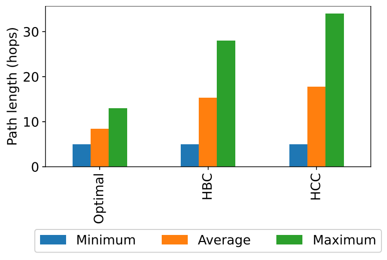


Figure 4: Hops between IoT devices and FNs.

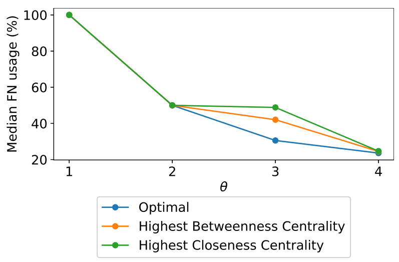


Figure 5: FN usage vs θ.

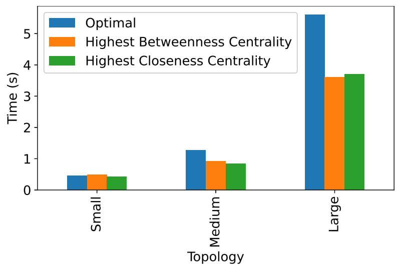


Figure 6: Execution time comparison.

The MIP formulation is able to optimally place FNs, as well as to optimally map IoT devices to FNs, in a reduced amount of time. Furthermore, MIP outperforms alternative placement methods in terms of average latency, path length and FN usage. However, the time consumption of the MIP formulation is shown to raise almost exponentially in large topologies. This implies that, while it is possible to use the formulation to initially place FNs optimally in an infrastructure, it may be difficult to quickly re-assess optimal placements over time in very large topologies.

In the future, we expect to develop heuristics that are able to place FNs nearly optimally while maintaining a low execution time. We also plan to analyze the cost of adding FNs in order to develop solutions that not only assess optimal placements for FNs, but also retrieve the optimal number of FNs to place. Moreover, we also plan to consider other factors that affect the performance of the system, such as coverage issues [17].

ACKNOWLEDGEMENT

This work has been partially funded by the project RTI2018-094591-B-I00 (MCI/AEI/FEDER,UE), 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 and Infrastructure of the Government of Extremadura (GR18112, IB18030), and by the European Regional Development Fund.

REFERENCES

[1] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-physical systems: The next computing revolution," in *Proceedings - Design Automation Conference*, 2010, pp. 731–736.

[2] 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.

[3] 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.

[4] 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, 2016.

[5] 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.

[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. Kassar, and T. Aguili, "Toward self-adaptive software defined fog networking architecture for IIoT and industry 4.0," in *Proceedings of IEEE CAMAD 2019*, 2019.

[8] T. Das, V. Sridharan, and M. Gurusamy, "A Survey on Controller Placement in SDN," *IEEE Communications Surveys & Tutorials*, pp. 472–503, 2019.

[9] O. Skarlat, M. Nardelli, S. Schulte, and S. Dustdar, "Towards QoS-Aware Fog Service Placement," in *Proceedings of IEEE ICFEC 2017*, 2017, pp. 89–96.

[10] R. P. Singh, J. Grover, and G. R. Murthy, "Self organizing software defined edge controller in IoT infrastructure," in *ACM International Conference Proceeding Series*, 2017, pp. 1–7.

[11] I. Azimi, J. Takalo-Mattila, A. Anzanpour, A. M. Rahmani, J. P. Soininen, and P. Liljeberg, "Empowering healthcare IoT systems with hierarchical edge-based deep learning," in *Proceedings of IEEE/ACM CHASE 2018*, vol. 18, 2019, pp. 63–68.

[12] P. B. Mirchandani and R. L. Francis, *Discrete location theory*, 1990.

[13] T. M. Ayenew, D. Xenakis, N. Passas, and L. Merakos, "A novel content placement strategy for heterogeneous cellular networks with small cells," *IEEE Networking Letters*, vol. 2, no. 1, pp. 10–13, 2019.

[14] P. Erdős and A. Rényi, "On Random Graphs," *Publicationes Mathematicae*, vol. 6, pp. 290–297, 1959.

[15] M. Roughan, "Simplifying the synthesis of internet traffic matrices," in *Computer Communication Review*, vol. 35, no. 5, oct 2005, pp. 93–96.

[16] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of Hotnets-9*, 2010, pp. 1–6.

[17] A. Tsiota, D. Xenakis, N. Passas, and L. Merakos, "On jamming and black hole attacks in heterogeneous wireless networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 11, pp. 10 761–10 774, 2019.

Appendix F

Multi-Objective Optimal Deployment of SDN-Fog Infrastructures and IoT Applications

ID	: **ICC 2022**
Authors	: Juan Luis Herrera, Jaime Galán-Jiménez, Paolo Bellavista, Luca Foschini, José García-Alonso, Juan Manuel Murillo, Javier Berrocal
Publication type	: Conference full-paper
Conference	: IEEE International Communications Conference (ICC) 2022
Status	: Under review.
Conference rating	: International conference, GGS Class 2 (A)

Multi-Objective Optimal Deployment of SDN-Fog Infrastructures and IoT Applications

Juan Luis Herrera*, Jaime Galán-Jiménez*, Paolo Bellavista†, Luca Foschini†, Jose Garcia-Alonso*, Juan M. Murillo* and Javier Berrocal*
*Department of Computer Systems and Telematics Engineering, University of Extremadura, Spain.
†Dipartamento di Informatica-Scienza e Ingegneria, University of Bologna, Italy.
[jlherrerag, jaime, jgaralo, juanmamu, jberolm]@unex.es, [paolo.bellavista, luca.foschini]@unibo.it

***Abstract*—The Internet of Things has brought digitalization to intensive domains through the automation of their real-world processes. However, the criticality of these processes is reflected in high Quality of Service (QoS) requirements for the application to work properly. Moreover, business-level QoS, such as the operational cost, are also key to the feasibility of these applications. This QoS depends on three, closely-related dimensions: the application software, the computing devices and the communication network, which provide high flexibility to obtain different performances at different costs. Thus, to achieve optimal QoS in these scenarios, the application, computing and networking dimensions must be optimized, considering their crucial interplay in a joint effort. Furthermore, this solution must allow multi-objective optimization, finding the optimal trade-off between operational cost and application performance. In this paper, we present Multi-Objective SDN Fog Optimization (MO-SFO), a holistic framework that allows for the optimization of both the response time and the deployment cost. MO-SFO is evaluated over an emulated smart city case study, showing the cost and performance trade-off achieved in different topologies.**

I. Introduction

The advent of the Internet of Things (IoT) paradigm has enabled for the bridging of computing applications and networks with real-world processes [1]. The potential of such computerization has attracted the interest of the research community in intensive domains, such as smart cities [1], industry or healthcare [2]. Nonetheless, the criticality of these domains is translated to strict Quality of Service (QoS) requirements for their IoT applications, such as low response times [1], [2]. Therefore, the integration of IoT-based systems in intensive domains is contingent on their ability to provide a high enough QoS, such as smart surveillance systems whose performance depends on the QoS [1], [2]. There are three dimensions that affect the QoS of IoT applications: application, computing, and networking [3]. All of them need to be considered in order to meet the strict QoS requirements of these intensive domains.

Regarding the application dimension, modern IoT applications require to be highly evolvable and interoperable, as well as allowing distributed deployments [4]. Thus, these applications are often designed following the Service-Oriented Computing paradigm, and, more concretely, the Microservices Architecture (MSA) design pattern [4]. In MSAs, applications are divided into small and loosely-coupled microservices, which perform simple tasks individually and collaborate to support more complex functionalities. Moreover, each of these microservices can be deployed individually, allowing the application to be deployed in a distributed manner and providing higher flexibility. Nonetheless, this is not a simplistic decision, as the application's QoS is affected by where each microservice is deployed. On the one hand, devices with higher computational power are able to provide lower execution times and, thus, lower response times. On the other hand, powerful devices tend to be further away from the data source, therefore having higher latencies, raising the response times. The problem of optimally deploying a set of microservices is labelled in literature as the Decentralized Computation Distribution Problem (DCDP) [5].

In the computer dimension, the QoS requirements, as well as the aforementioned focus on distribution, also motivate the used paradigms. Traditionally, cloud computing was used to deploy IoT applications [6]. Nonetheless, due to the complexity of guaranteeing a suitable QoS using cloud servers exclusively, which tend to be far away from IoT devices, intensive domains tend to move towards fog computing infrastructures [1]. In these infrastructures, servers that are closer to users (*fog nodes*, FNs) are used alongside with cloud servers [6]. Fog infrastructures generally enhance the QoS of the applications deployed in them due to their lower latencies. Nonetheless, the microservices running on both the cloud and FNs need to communicate between themselves, and therefore, the placement of these FNs is crucial in the QoS enhancement experienced by the application [2], [3]. In order to place them optimally in the infrastructure, the Fog Node Placement Problem (FNPP) [2] needs to be solved.

The QoS of the communications between microservices and, therefore, between devices, depends on the networking dimension. The network needs to be scalable and flexible, meeting the required QoS [1], [3]. Furthermore, in fog infrastructures, microservices are often replicated and deployed in different FNs [3]. Before consuming a certain microservice, the IoT device needs to learn where the nearest replica is deployed (*service discovery*) [7]. In order to perform this task separately from the IoT application's concerns, it would be desirable for the network to automatically route requests to the nearest replica. It is possible to perform service discovery at the network level while providing scalability and flexibility, by exploiting the Software-Defined Networking (SDN) paradigm [7],

A foundation of the SDN paradigm is to decouple the data plane, embodied in SDN switches, from the control plane, which is managed at the SDN controller. Therefore, the SDN switches and the SDN controller need to communicate. The communication QoS between them is crucial, as every time these planes need to interact, they will be subject to such QoS [8]. In order to optimize this QoS, the SDN Controller Placement Problem (CPP) [8] can be solved, placing the SDN controller wherever the QoS from the switches is best.

These three dimensions are inherently coupled, as each decision taken in a dimension affects the rest. Changing the placement of the controller affects the overall network QoS [8], which dictates the QoS between FNs and, consequently, the QoS between the microservices deployed in them [3]. Changing the placement of a FN affects the flow of traffic, as a node that is the source and destination of communications is moved [2]. The flow of traffic is key to the optimal placement of the SDN controller [8], and thus, the network QoS will be affected, therefore also affecting the QoS of the microservices that are either deployed in the moved FN or communicating with microservices deployed in the moved FN [3].

However, companies usually have limited resources. This constraint limits the deployment configurations that they can feasibly deploy [9]. In these cases, there are two QoS objectives in conflict: performance QoS, namely response time, and business QoS, in the form of deployment cost [9]. These two QoS objectives need to be traded off, performing a multi-objective optimization that suggests high-performance, low-cost deployment configurations that can be feasibly deployed.

In this work, we present Multi-Objective SDN Fog Optimization (MO-SFO), a multi-objective framework for the holistic solution of the CPP, the DCDP and the FNPP, optimizing response time and cost. In [3], some of the authors of this work proposed Umizatou, a framework able to optimize the response time by solving these three problems in a single effort. However, Umizatou only allows for the optimization of response time. This shortcoming complicates the use of Umizatou in scenarios in which the deployment cost is a QoS objective, as well as those in which cost and response time need to be traded off. MO-SFO extends Umizatou by considering multiple metrics, as well as multi-objective optimization. Furthermore, MO-SFO has been evaluated over an emulated SDN-fog testbed, whereas Umizatou's evaluation was only performed in a simulated environment [3]. The main contributions of this work are:

- The formalization of a problem that combines CPP, DCDP and FNPP for intensive IoT environments, aimed at optimizing the response time, the deployment cost, or both at the same time.
- The formulation of a multi-objective, Mixed Integer Linear Programming (MILP) optimization solution.
- The evaluation of MO-SFO over a realistic, emulated smart city case study.

The remainder of this paper is structured as follows. Sec. II explains the system model of MO-SFO. Sec. III details the multi-objective optimization formulation of MO-SFO making use of MILP. Sec. IV presents the evaluation of MO-SFO using an emulated smart city case study. Finally, Sec. V concludes the paper and highlights future challenges.

II. System model

To make it easier to understand the MO-SFO model, the problem is explained through a smart city case study. This case study is based on Intel's OpenVINO toolkit, an industrial framework for the execution of deep learning video and audio analysis models, designed for its use in fog environments and tailored towards smart cities [10]. Nonetheless, the MO-SFO framework is not exclusive to smart city scenarios, and could be used in other intensive IoT domains as well.

In the case study, the smart city has a set of cameras and microphones, which act as IoT devices and continuously send their information to a surveillance IoT application. This application allows for five different functionalities: detection of presence of people and vehicles, visual tracking of such people and vehicles, classification of the vehicles based in their attributes (e.g., the type of vehicle, their colors or their sizes), detection of the zone vehicles and people are on (e.g., whether they are on the sidewalk or on the road, on which lane of the road they are on), and environmental audio recognition (e.g., detection of car horns or car crashes through their sound) [10]. Each of these functionalities is implemented as a microservice. Based on their role (e.g., security surveillance cameras, traffic control cameras, emergency detection microphones), they can request these functionalities to be carried out with their video or audio inputs. This application is depicted in Fig. 1.

For the deployment of this surveillance application in the smart city, a SDN network, with the topology shown in Fig. 1, is envisioned. While this network connects the IoT devices, the placement of the SDN controller, as well as the fog nodes, is not yet decided. Moreover, both the application's response time and its deployment cost are considered crucial QoS metrics that need to be optimized [1]. Thus, the CPP, the FNPP, and the DCDP need to be solved to optimize the QoS of this application. The application operator defines the two QoS metrics to be optimized in order to find the best trade-off between response time and cost.

In this scenario, the objective of MO-SFO is to deploy SDN controllers, FNs and microservice instances within the FNs, in a manner that minimizes both the cost of the deployment and the application's response time. To do so, MO-SFO decides to use the deployment shown in Fig. 1: 2 FNs are deployed, one on the top side, with microservices for video processing, and one in the bottom, including audio recognition, and an SDN controller is deployed in the middle. MO-SFO deems this deployment optimal in terms of both objectives. While it would be possible to deploy more microservice replicas in each FN to further decrease response time, doing so would increase the energy consumption, and thus, the operational expenditures (OPEX) of the system. Similarly, deploying more FNs or SDN controllers would also decrease the response time, at the cost of higher capital expenditures (CAPEX). On the other hand, it would be possible to deploy a single FN, running a replica of each microservice, to reduce the CAPEX

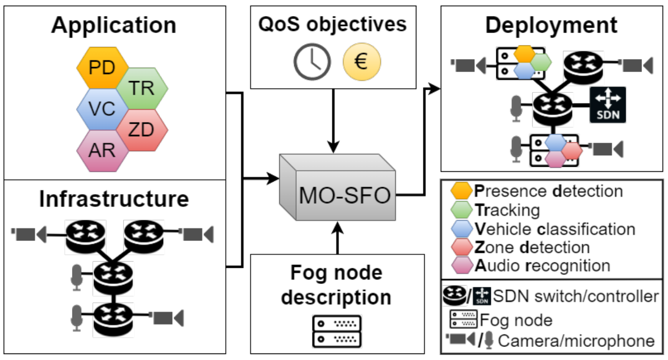


Figure 1: MO-SFO in the smart city case study.

and OPEX of the system. However, doing so would provoke an increase in the response time.

The decisions of MO-SFO are possible due to its multi-objective joint approach: in order to assess if deploying additional FNs is more cost-effective than deploying additional SDN controllers, MO-SFO needs control over both the computing and networking dimension. Furthermore, the cost-effectiveness of the equipment depends on the application dimension's communication patterns, computational workload and microservice placement. This sort of analysis can only be performed if both objectives are optimized with an holistic perspective that covers all three dimensions.

III. Problem formulation

In this section, the mathematical abstraction of the model of the joint CPP, DCDP and FNPP is presented as an MILP formulation. This formulation allows for the problem to be solved through the use of automatic MILP software solvers.

In this formulation, the infrastructure is represented by a directed graph $G = \{V, L\}$, where L represents the links and V represents the infrastructure's vertices. A link denoted as l_{ij} links together vertices i and j, with a propagation latency δ_{ij} and a maximum capacity θ_{ij}. The vertices of the infrastructure can be divided into two subsets: $V = H \cup S; S \cap H = \emptyset$, with H containing the vertices that are hosts, and S containing those that are SDN switches. Furthermore, every switch $s \in S$ has a certain processing latency δ_s. To simplify the understanding of the formulation, let $SW(v) \forall v \in V$ be a binary function, whose value is 1 if $v \in S$ and 0 otherwise. Moreover, each switch in the infrastructure has a given CAPEX ($CAPEX_s$) and OPEX per second due to energy consumption ($OPEX_s$).

Continuing with fog nodes, let r be the total RAM memory of an FN, which determines the number and kind of microservice replicas that can be deployed in it. In terms of cost, placing the FN has a certain CAPEX, defined as $CAPEX_{FN}$, and consumes an amount of energy, i.e., it has a certain OPEX. The energy consumption depends on the usage of the FN, as more intensive usage consumes more energy, and thus, we define $OPEX_{FN}$ as the OPEX per second for the corresponding FN. With respect to the SDN controller, we need to consider the size of control packets, which we label σ, to support multiple SDN protocols. Nonetheless, in this work, this size is taken from the OpenFlow 1.3 specification [11]. Similarly to the switches, each controller has a certain CAPEX ($CAPEX_C$) and OPEX per second due to energy consumption ($OPEX_C$).

To represent the application, we use a model based in the MSA pattern, i.e., the application is divided into a set of microservices we label M. Each microservice m consumes an amount of RAM r_m, and its input and output data have a size of I_m and O_m, respectively. Furthermore, in MSA-based applications, each client (i.e., IoT device) requests for functionalities in the form of *workflows* [4]. Each workflow can involve one or multiple collaborating microservices, which are executed in order, pipelining the output of a microservice to the input of the next one [9]. We represent the total set of workflows requested as W, and each workflow as an ordered set of microservices $w = \{c_1, c_2, ..., c_{|w|}\} \subseteq M$. It is important to note that each element is a microservice $c_i \in M$, and elements from different workflows can refer to the same microservice. For simplicity, we also define the binary function $WR(w, h) \forall w \in W, h \in H$, which takes the value of 1 if workflow w is requested by host h and 0 otherwise.

Finally, we introduce the parameter $\alpha \in [0, 1]$ as a means to select the objective or objectives of the optimization, as well as to decide the priority of each objective. If $\alpha = 0$, MO-SFO will only consider the deployment cost as the objective, useful for scenarios with very stringent budgets and not very time critical. Conversely, $\alpha = 1$ only enforces the response time objective, which may be used whenever the budget is not a constraint. A value of $\alpha = 0.5$ indicates MO-SFO should find a trade-off between deployment cost and response time, which is meant for problems where the budget constrains an application that requires high performance.

After defining the parameters for the problem, the decision variables need to be defined as well. To solve the FNPP, FNs must be placed, replacing SDN switches: let $n_s \forall s \in S$ be a binary variable that will take a value of 1 if a FN is placed in switch s. Continuing the joint approach, to solve the DCDP, microservice replicas need to be deployed to FNs. We define $z^w_{sc_a} \forall w \in W, s \in S, a \in [1, |w|]$ a binary variable, representing whether the replica of microservice c_a in workflow w is deployed to the FN in switch s or not. Moreover, the communication between microservices generates traffic flows, which are represented by the binary variables $f^{vwc_a}_{ij} \forall l_{ij} \in L, v \in V, w \in W, a \in [1, |w|]$. These variables take a value of 1 if the traffic generated by vertex v as a consequence of requesting microservice c_a of workflow w is routed through the link l_{ij}. It is important to note that the response from the last microservice in each workflow to the requesting IoT device must also be considered: let the binary variables $f'^{vw}_{ij} \forall l_{ij} \in L, v \in V, w \in W$ represent them with a value of 1 if the traffic generated by vertex v due to the response of workflow w is routed through the link l_{ij}.

Finally, the CPP needs to be solved as well. SDN controllers are also placed in switches, and thus, $x_s \forall s \in S$ is defined as a binary variable whose value is 1 if an SDN controller is placed on SDN switch s. Moreover, if multiple controllers are placed, each switch needs to know which controller they should communicate with: let $y_{ss'} \forall s, s' \in S$ be a binary variable that will be 1 if the SDN switch s is assigned to communicate with the controller from switch s'. These communications also generate traffic flows, which are represented through the binary variables $cf^s_{ij} \forall l_{ij} \in L, s \in S$. These variables take the value of 1 if the flow of control communications for SDN switch s is routed through the link l_{ij}. Then, the problem can be formalized as follows:

$$\min \alpha RT + (1-\alpha)(CAPEX + OPEX) \tag{1}$$

subject to:

$$\begin{aligned} RT = & \frac{1}{|W|} \sum_{w\in W} \sum_{s\in S} \sum_{l_{ij}\in L} (\\ & \sum_{a=1}^{|w|} (f^{swc_a}_{ij}) + f'^{sw}_{ij})(\delta_{ij} + SW(j)\delta_i) \\ & + SW(j) \sum_{l_{km}\in L} cf^j_{km}(\delta_{km} + SW(m)\delta_k) \end{aligned} \tag{2}$$

$$CAPEX = \sum_{s\in S}(CAPEX_{FN} n_s + CAPEX_C x_s + CAPEX_s u_s) \tag{3}$$

$$\begin{aligned} OPEX = \sum s \in S((\sum_{w\in W} \sum a = 1^{|w|} \frac{r_{c_a}}{r} OPEX_{FN} z^w_{sc_a}) \\ + OPEX_C x_s + OPEX_s u_s) \end{aligned} \tag{4}$$

$$\begin{aligned} u_s = \max(& \max_{l_{is}\in L, v\in V, w\in W, a\in[0,|w|]} (\max(f^{vwc_a}_{is}, f'^{vw}_{is})) \\ & , \max_{l_{is}\in L, s'\in S} (cf^{s'}_{is})) \end{aligned} \tag{5}$$

$$\forall w \in W, a \in [1,|w|] : \sum_{s\in S} z^w_{sc_a} = 1 \tag{6}$$

$$\forall w \in W, a \in [1,|w|], s \in S : z^w_{sc_a} \le n_s \tag{7}$$

$$\forall s \in S : \sum_{w\in W} \sum_{a=1}^{|w|} z^w_{sc_a} r_{c_a} \le r \tag{8}$$

$$\begin{aligned} \forall l_{ij} \in L : \sum_{s\in S}[cf^s_{ij}\sigma + \sum_{w\in W} [(\sum_{a=1}^{|w|} f^{swc_a}_{ij} I_{c_a}) + (f'^{sw}_{ij} O_{c_{|w|}})]] \\ \le \theta_{ij} \end{aligned} \tag{9}$$

$$\forall i, v \in V, w \in W :$$
$$\sum_{j\in V} f^{vwc_1}_{ij} - f^{vwc_1}_{ji} = \begin{cases} WR(w,v) & \text{if } i = v \\ -WR(w,v) z^w_{ic_1} & \text{if } v \in H \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

$$\forall i, v \in V, w \in W :$$
$$\sum_{j\in V} f'^{vw}_{ij} - f'^{vw}_{ji} = \begin{cases} z^w_{vc_{|w|}} & \text{if } i = v \\ -WR(w,i) & \text{otherwise.} \end{cases} \tag{11}$$

$$\forall i, v \in V, w \in W, a \in [0,|w|] : -z^w_{vc_{a-1}} + z'^{iw}_{vc_a} \le 0 \tag{12}$$

$$\forall i, v \in V, w \in W, a \in [0,|w|] : -1 + z^w_{ic_a} + z'^{iw}_{vc_a} \le 0 \tag{13}$$

$$\forall i, v \in V, w \in W, a \in [0,|w|] : z^w_{vc_{a-1}} + 1 - z^w_{ic_a} - z'^{iw}_{vc_a} \le 1 \tag{14}$$

$$\forall i, v \in V, w \in W, a \in [0,|w|] : -z^w_{ic_a} + z''^{iw}_{vc_a} \le 0 \tag{15}$$

$$\forall i, v \in V, w \in W, a \in [0,|w|] : -z^w_{vc_{a-1}} + z''^{iw}_{vc_a} \le 0 \tag{16}$$

$$\forall i, v \in V, w \in W, a \in [0,|w|] : z^w_{vc_{a-1}} + z^w_{ic_a} - z''^{iw}_{vc_a} \le 1 \tag{17}$$

$$\forall i, v \in V, w \in W, a \in [2,|w|] :$$
$$\sum_{j\in V} f^{vwc_a}_{ij} - f^{vwc_a}_{ji} = \begin{cases} z'^{iw}_{vc_a} & \text{if } i = v \\ 0 & \text{if } v \in H \\ -z''^{iw}_{vc_a} & \text{otherwise.} \end{cases} \tag{18}$$

$$\forall s \in S : \sum_{s'\in S} y_{ss'} = 1 \tag{19}$$

$$\forall s, s' \in S : y_{ss'} \le x_{s'} \tag{20}$$

$$\forall i \in V, s \in S :$$
$$\sum_{j\in V} f^{vwc_a}_{ij} - f^{vwc_a}_{ji} = \begin{cases} 0 & \text{if } i \in H \\ 1 - y_{si} & \text{if } i = s \\ -y_{si} & \text{otherwise.} \end{cases} \tag{21}$$

Eq. 1 expresses the optimization objective: to minimize the weighted sum of the average response time of workflows and the deployment cost, integrated by CAPEX and OPEX. Starting with response time, since all FNs are equal, each microservice has a constant execution time, and hence the objective is to minimize workflow latency, as defined in Eq. 2. Workflow latency includes both the latency of application traffic flows (i.e., traffic generated by requesting microservice execution) and the control latency of switches traversed by the application flows related to the workflow. With respect to the cost, CAPEX is defined in Eq. 3 as the sum of the CAPEX of used switches, FNs and SDN controllers. OPEX, defined at Eq. 4, integrates the energy consumption of used switches and controllers as well, while the energy consumption of FNs is integrated as dependent on the resource consumption of the microservices deployed in it. Nonetheless, to define them both, we need to know if a switch is being used or not, which is defined in Eq. 5.

Moving to constraints, Eq. 6 states that each microservice request in a workflow can only be fulfilled once. Eq. 7 ensures that only FNs that are actually placed can execute microservices. Eq. 8 states that the total RAM consumed by the microservices executed in an FN cannot be higher than the available RAM of the FN. Eq. 9 enforces link capacity. Eqs. 10-18 adapt the classic flow constraints to the traffic flows generated by workflows. Eq. 19 makes sure only one controller is assigned to each switch. Eq. 20 states that only placed controllers can be assigned to switches. Finally, Eq. 21 adapts the flow constraints to control flows.

This formulation, along with a defined scenario, can be used as an input to an software MILP solver to obtain a solution with optimal placements for SDN controllers, FNs and microservices, as well as optimal information routing, according to the optimization objective or objectives selected.

IV. Performance evaluation

The objective of this section is the evaluation of MO-SFO over a smart city case study using different topologies.

A. Evaluation environment

The evaluation of MO-SFO has been performed over emulated testbeds based on the case study presented in Sec. II. These emulated environments have been created using the Kathará emulation framework [12], which allows for the creation of emulated SDN networks using Docker containers. Regarding the networking dimension, the switches of the emulated networks are based on OpenVSwitch [13], while the SDN controller software is Faucet, a controller made for enterprise usage [14]. Moreover, the network conditions have been emulated using Linux's traffic scheduler through the *tc* command. The evaluation has been performed on topologies created using the Erdös-Rényi model for graph generation [15], transforming the nodes with a degree of one into IoT devices, while the rest are left as SDN switches. The tests have been performed over two topologies: the *Small* topology (7 SDN switches, 6 IoT devices), and the *Large* topology (20 SDN switches, 18 IoT devices), based on the topology sizes from [3]. On the application dimension, all the five microservices defined in the case study have been emulated using the official OpenVINO Docker image from Intel [1], which includes real implementations for all of them, requiring approximately 1.25 GB of RAM each. IoT devices are emulated through Alpine Linux-based containers that stream the videos provided by Intel for their use with OpenVINO [2]. Each IoT device requested the execution of a single workflow 10 times, which are used on the Docker image to measure the average response time. On the computing dimension, each FN has the specifications of a PICO-TGU4, a device for intensive domains with 32 GB of RAM. The emulated environments have been executed in an AWS $c1.xlarge$ instance.

The version of MO-SFO used for the evaluation applies the formulation from Sec. III, through the use of the MILP solving software Gurobi, to the smart city scenario. MO-SFO has been executed in a computer with an Intel i7-8565U CPU and 16 GB of RAM. Three values of α were tested in each topology: 1, 0.5 and 0, which are labeled under the objective they represent (*Response time*, *Trade-off* and *Cost*, respectively). It is important to note that, as MO-SFO expands on Umizatou, the *Response time* results can be seen as Umizatou results, hence enabling for their comparison. In these six scenarios, different analyses have been performed. First, an analysis to evaluate the cost of the deployment proposed by MO-SFO in each scenario is performed. In this analysis, the overall cost of each scenario, including the cost of each of the elements (e.g., SDN controllers, FNs) is evaluated, including the CAPEX and the OPEX over 5 years, which is a common amortization period [1]. Moreover, the distributions of the response time obtained in the emulated environments are also analyzed, assessing both the performance of MO-SFO solutions and allowing for the evaluation of the cost-effectiveness achieved by each objective.

[1]https://hub.docker.com/r/intel/video-analytics-serving
[2]https://github.com/intel-iot-devkit/sample-videos

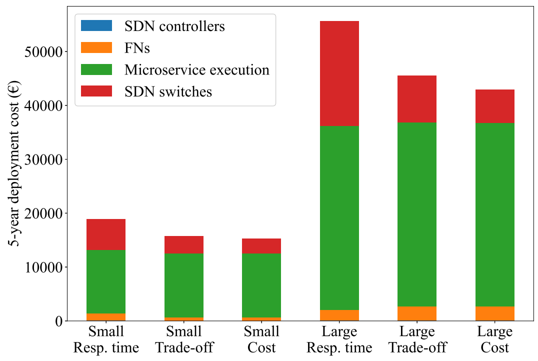


Figure 2: Deployment cost of each scenario, including CAPEX and OPEX during 5 years.

B. Performance analysis

In the first analysis, depicted by Fig. 2, the cost of the deployment during 5 years on each of the six scenarios is compared, as well as the contribution of each element to the total cost. Moreover, four elements are considered part of the costs: the SDN controllers (including their CAPEX and OPEX), the CAPEX of FNs, the OPEX due to the microservice execution, and the SDN switches. The first conclusion that can be drawn from the figure is that the main source of cost is the energy consumption of the microservices, accounting for 11,845€ (65-77% of the total cost, 6.49€/day) in the small topology and 34,123€ (67-79%, 18.69€/day) in the large topology, $2.88\times$ as much as in the small topology. Moreover, the cost due to microservice execution is the same across all objectives, due to the fact that the same number of microservice replicas were deployed in all cases. The deployment of the network infrastructure, i.e., SDN switches, is also a very important element in terms of cost, representing a total of 16-30% of the cost in the small topology, and a 13-35% of the large topology costs. Furthermore, the large topology has an overall higher cost than the small topology, and the response time objective requires for more costly deployments. The cost and trade-off objectives place a single controller in the small topology, while the response time objective places two. Nonetheless, three are placed in the large topology, regardless of the objective. Finally, the optimal cost objective achieves the best cost, while the multi-objective trade-off achieves a slightly (3-6%) higher cost. The response time objective, however, has the highest costs, 20-23% higher than the cost objective.

Fig. 3 details the distribution of the response time in each of the scenarios using a box plot, in which the medians are drawn as white lines in each box, and the means are depicted by cyan triangles. These response times are the result of 10 requests of analysis over a 1-minute-long video, using the multiple microservices of the application (e.g., audio recognition, vehicle classification), and thus, the response time cannot be less than 60 seconds. Moreover, due to incompatibilities, GPU acceleration was also disabled in the emulated testbed. Starting with the small topology, the response time objective achieves an average response time of 299 seconds, a median response time of 304 seconds and a standard deviation of

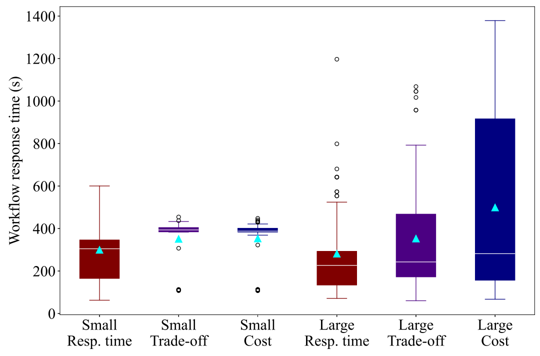


Figure 3: Distribution of response time in each scenario

163 seconds. These are the overall lowest response times for the topology, as the trade-off and cost objectives have similar mean response times, of 351 and 353 seconds (17% and 18% higher), respectively. While the median response time in the trade-off objective is slightly higher than in the cost objective, it is important to note the interquartile range is lower in the former (277 to 425 seconds) than in the latter (282 to 424 seconds), i.e., a *normal* IoT device experiences response times 1% lower if the trade-off multi-objective version is selected. Moving to the large topology, the response time objective exhibits the best response times again, with an average of 281 seconds, a median of 226 seconds, and a standard deviation of 224 seconds. Overall, the response times are lower than in the small topology, albeit the higher standard deviation indicates a larger spread. Similarly, the trade-off objective achieves an average of 352 seconds (57% higher) and a standard deviation of 276 (22% higher), whereas the cost objective exhibits an average of 499 seconds (78% higher) and an standard deviation of 413 (84% higher). On average, the trade-off multi-objective version exhibits 11% lower response times than the single-objective cost alternative, while they are a 37% higher than the response time-focused alternative.

Finally, if the results in terms of both cost and response time are considered, the multi-objective trade-off version is the most cost-effective, as it achieves a very similar deployment cost as the single-objective version, and lower response times. This phenomenon is due to the fact that the cost objective does not consider response time at all, and simply makes sure the application is able to function, whereas the trade-off objective arranges the application to optimize the response time while constraining the deployment to minimize its cost.

V. Conclusions and Future Work

The interactions between the physical and computing world brought by IoT make it interesting to apply its potential to intensive domains. However, the digitalization of critical processes comes with the cost of high QoS requirements for associated IoT applications. Furthermore, the QoS of intensive IoT applications depends on the application, computing and networking dimensions, as well as in the interplay between them, and is often multi-objective in nature. Optimizing these dimensions to achieve the required QoS calls for solutions able to consider the coupling between all three dimensions and trade off multiple QoS metrics to cater to specific IoT applications. In this paper, we presented MO-SFO, a multi-objective solution able to optimally place SDN controllers, FNs and microservices for intensive IoT environments. MO-SFO has been evaluated in an emulated smart city case study, focusing on the trade-off between the response time and deployment cost objectives. In the future, we expect to allow MO-SFO to use alternative multi-objective optimization algorithms, such as genetic algorithms, to improve the optimization times and provide a Pareto front. Finally, we also expect to integrate MO-SFO with state-of-the-art orchestrators, such as Kubernetes.

Acknowledgements

This work was partly funded by project PID2021-124054OB-C31 (MCI/AEI/FEDER,UE), by the European Regional Development Fund, by the Department of Economy, Science and Digital Agenda of the Government of Extremadura (GR21133), and by the Valhondo Calaff institution.

References

[1] S. Ketu and P. K. Mishra, "A contemporary survey on iot based smart cities: Architecture, applications, and open issues," *Wireless Personal Communications*, pp. 1–49, 2022.
[2] J. L. Herrera, P. Bellavista, L. Foschini, J. Galán-Jiménez, J. M. Murillo, and J. Berrocal, "Meeting stringent qos requirements in iiot-based scenarios," in *IEEE Global Communications Conference*, 2020, pp. 1–6.
[3] J. L. Herrera, J. Galán-Jiménez, P. Bellavista, L. Foschini, J. Garcia-Alonso, J. M. Murillo, and J. Berrocal, "Optimal deployment of fog nodes, microservices and sdn controllers in time-sensitive iot scenarios," in *IEEE Global Communications Conference*, 2021, pp. 1–6.
[4] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: State of the art and research challenges," *Computer*, vol. 40, no. 11, pp. 38–45, 2007.
[5] 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, dec 2019.
[6] 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.
[7] 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 and Tutorials*, vol. 19, no. 4, pp. 2359–2391, 2017.
[8] T. Das, V. Sridharan, and M. Gurusamy, "A Survey on Controller Placement in SDN," *IEEE Communications Surveys & Tutorials*, pp. 472–503, 2019.
[9] J. L. Herrera, J. Galán-Jiménez, J. García-Alonso, J. Berrocal, and J. M. Murillo, "Joint optimization of response time and deployment cost in next-gen iot applications," *IEEE Internet of Things Journal*, 2022.
[10] Intel, "OpenVINO Toolkit," 2019. [Online]. Available: https://software.intel.com/en-us/openvino-toolkit
[11] Open Networking Foundation, "OpenFlow Switch Specification 1.3.0," 2013. [Online]. Available: https://tinyurl.com/openflow-v13-spec
[12] M. Scazzariello, L. Ariemma, and T. Caiazzi, "Kathará: A lightweight network emulation system," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2020, pp. 1–2.
[13] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The design and implementation of openvswitch," in *12th USENIX symposium on networked systems design and implementation (NSDI 15)*, 2015, pp. 117–130.
[14] J. Bailey and S. Stuart, "Faucet: Deploying sdn in the enterprise," *Communications of the ACM*, vol. 60, no. 1, pp. 45–49, 2016.
[15] P. Erdös and A. Rényi, "On random graphs i," *Publ. math. debrecen*, vol. 6, no. 290-297, p. 18, 1959.

Appendix G

Privacy-Aware and Context-Sensitive Access Control for Opportunistic Data Sharing

ID	: **CCGrid 2021**
Authors	: Juan Luis Herrera, Hsiao-Yuan Chen, Javier Berrocal, Juan Manuel Murillo, Christine Julien
Publication type	: Conference full-paper
Conference	: IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID) 2021
DOI	: `https://doi.org/10.1109/CCGrid51090.2021.00092`
Conference rating	: International conference, GGS Class 2 (A)

Privacy-Aware and Context-Sensitive Access Control for Opportunistic Data Sharing

Juan Luis Herrera*, Hsiao-Yuan Chen†, Javier Berrocal*, Juan M. Murillo* and Christine Julien†
*University of Extremadura, Spain. [jlherrerag, jberolm, juanmamu]@unex.es
†University of Texas, Austin, United States. [littlecircle0730, c.julien]@utexas.edu

***Abstract*—Opportunistic data sharing allows users to receive real-time, dynamic data directly from peers. These systems not only allow large-scale cooperative sensing but they also empower users to fully control what information is sensed, stored, and shared, enhancing an individual's control over their own potentially private data. While there exist context-aware frameworks that allow individual users to define when and what shared information peers can consume, these approaches have limited expressiveness and do not allow data owners to modulate the granularity of the information released depending on a particular peer or situation. In addition, these frameworks do not consider the consuming peers' privacy, i.e., how much information they have to provide to get access to some desired data. In this paper, we present PADEC, a context-sensitive, privacy-aware framework that allows users to define rich access control rules over their resources and to attach levels of granularity to each rule in order to precisely define who has access to what data when and at what level of detail. Our evaluation shows that PADEC is more expressive than other access control mechanisms and protects the provider's privacy up to 90% more.**

***Index Terms*—opportunistic networks, access control, privacy**

I. Introduction

During the last few years, there has been a massive deployment of mobile devices. Almost every developed country has at least 90% mobile phone penetration and 80% smartphone penetration [1]. Similar trends exist for IoT and other wearable devices, with a projected penetration of 25% in the US by 2022 [2]. These devices carry a large number of on-board sensors that provide their owners with a rich view of the surrounding context and support a wide array of application functionalities. In addition, as these devices are commonly mobile, they accompany the user, providing an interface between their owner and a wider networked community [3].

This astounding increase in *companion devices* has enabled new types of systems and applications, such as mobile crowd sensing [4] or opportunistic data sharing [5], which both leverage the myriad devices to execute large-scale sensing tasks. Mobile crowd sensing recognizes that a user's sensing needs can often be fulfilled by others nearby [6]. Opportunistic data sharing, which can be combined with mobile crowd sensing, relies on transient wireless network connections [7] to distribute and fulfill a data sharing task using information from the local networked devices. As interactions are pushed into the opportunistic space, individuals need mechanisms to control the release data according to their privacy needs. Privacy management mechanisms must empower users to manage who, when, and how their personal information can be consumed [8], i.e., each user should be able to decide which information is shared with whom, in how much detail, and under which contextual conditions.

Context-aware access controls constrain access to data or resources based on specific contextual conditions [9]. In contrast, basic access control mechanisms, such as Role-Based Access Control (RBAC) [10] or Dynamic Sharing and Privacy-aware Role-Based Access Control (DySP-RBAC) [11], allow data or resource providers to control access to information based on the identity of a potential consumer. These mechanisms are not sufficiently expressive to support applications with opportunistic device-to-device connections, in which identities of most peers are unknown. Other proposals, such as Attribute-Based Access Control (ABAC) [12] or the access control mechanism from [13], allow users to define rules based on contextual conditions beyond identity. However, these mechanisms cannot modulate the precision of information released to each peer as a function of the shared context, which prevents providers from having fine-grained control of the access to their data or resources. Furthermore, these systems are designed for centralized (cloud-based) data sharing in which a coordinator negotiates data sharing and access control. This is not the case of in our scenarios, which are often completely decentralized.

This paper presents a Privacy-Aware and Context-Sensitive Access Control mechanism (PADEC), designed for completely decentralized data sharing scenarios. PADEC empowers users by increasing the expressiveness of access rules and protecting the privacy of both providers and consumers. PADEC allows resource owners to define rich access rules based on different kinds of contextual information, as well as to attach filters to these rules in order to provide the data or resources at different levels of granularity. At the same time, PADEC protects the privacy of resource consumers by minimizing the amount of contextual information that they need to share within their access requests. To that end, we introduce the concept of *keyhole* that provides consumers information about the context information they must provide in order to gain access.

This paper is organized as follows. Section II presents a case study that serves as motivation. Section III describes related works on context-sensitive and privacy-aware access control, while Section IV elicits the threat model for our opportunistic data sharing scenarios. Section V incrementally introduces the contributions of PADEC, which is then evaluated over the case study in Section VI, providing comparisons with other access control mechanisms. Finally, Section VII concludes.

II. Case Study Application

To motivate the problem, we present a case study of a mobile crowd sensing application that relies on opportunistic data sharing in a smart city. In this scenario, tourists arriving in a smart city want to find the most popular Points Of Interest (POIs) of the city according to local residents: restaurants that the locals frequent, bars and cafes that are popular, parks that local residents enjoy, etc. To aid visitors, the local government has released an application that leverages users' sensors to store a history of the POIs they have visited and to share this information with other nearby users by relying on hyper-local wireless links. The government has encouraged local residents to use this application as a means to promote tourism. Such a scenario is not just a thought experiment; similar applications have already been envisioned [14]. From an application architecture perspective, there are two roles in the application; the local residents are *providers* of information in the smart city, and the tourists are *consumers* of that information. Concretely, a provider's device exposes one or more *endpoints* that a consumer can access using opportunistic device-to-device communication to retrieve the information.

When the local residents share information directly with nearby tourists, privacy is crucial due to the personal nature of the data being handled. A malicious party could potentially use the history of a user to track their habits, so this information must be carefully protected. With a centralized approach, local residents would need to store their POI history to a centralized server, provided by a third party (e.g., the local government), that has access to the full POI history of every user. This means that users need to trust this third party, since they are transferring the ownership of their POI history. This is not the case with opportunistic data sharing.

Although privacy could be easily maintained by not allowing any user to access a resident's POI histories, this would render the application useless. Thus, while granting open access to everyone is undesirable because of the private nature of the POI history, so is consistently denying access. Users require access control mechanisms that grant access to individual users based on their context, as well as to allow data owners to share their private information at different levels of granularity, depending on that context. For instance, a local resident might not mind sharing a small portion of their history with nearby tourists. If this tourist is a friend of the resident, or part of their family, the resident is likely to be willing to share a much larger part of the history with them because the resident has some implicit trust in the consumer.

On the other side, tourists access the providers' endpoints by applying to the access control mechanism for authorization. To do so, they must provide their own contextual information. While the reduced amount of information released by the consumer makes it less critical than the data released by locals, the privacy of tourists should also be protected. Similar to how the data providers need a tool to select how much information they are willing to share based on the context, tourists need similar mechanisms to restrict their exposure. In this case study application, the visiting tourists will release their own context information for the purpose of getting access to providers' resources or data. In this process, the consumers can be guided by the policies defined by the data providers.

III. Background and Related Work

In the opportunistic data sharing approaches that motivate our work, the providers and consumers that are collaborating may be completely unknown to one another, we do not assume the presence of a central coordinator to help mediate privacy protection, and different parties may have dramatically different privacy sensitivities. In the remainder of this section, we examine existing work that addresses facets of the access control challenge for opportunistic data sharing applications.

Context-sensitive access control. During the last few years, different works focused on using personal context information to authenticate users. In [15], context information, such as nearby familiar devices or locations, is used to apply an access control policy and adapt the authentication method of mobile phones. In [16], the users' movements, like gait, are leveraged to validate the user's identity. While the identification of users using their context information and attributes has proven useful for unlocking and adapting the behavior of personal devices, it requires an *a priori* model of the attributes or behavior of each user, which is unfeasible for large-scale applications, especially those that need to authenticate and grant access to unknown users depending on their context.

Decentralized access control. Different works propose access control frameworks to access shared data under different contexts. For instance, Penumbra [17] proposes an expressive and decentralized access control system that empowers users and that can be used in opportunistic scenarios in which there is no central mediator. Nevertheless, Penumbra only considers identity in its access control decisions, and omits other contextual information. In [18], the authors propose the use of a distributed ledger, called Tangle [19], to store the policies and access rights for resource-constrained IoT devices. This ensures the distributed auditability of the process. Although this mechanism is also privacy-aware, privacy is only preserved if access control decisions are made by honest participants, which is difficult to ensure in opportunistic scenarios.

Privacy-aware access control. Opportunistic and pervasive scenarios require decentralized and context-sensitive access control mechanisms in which individual users can also modulate the information to release depending on the consumer's context. Existing access control mechanisms have been defined for collaborative environments. The NIST standard RBAC [10] authenticates users based on identity by grouping them with roles. However, RBAC is not context-sensitive nor privacy-aware. DySP-RBAC [11] extends RBAC into a context-sensitive and privacy-aware mechanism. However, DySP-RBAC is implemented applying a pure server/cloud-based architectural style, in which all users are known in advance and users do not retain ownership of their own data. Moreover, context in DySP-RBAC is based on the relationships between users, and it therefore is fully identity-based.

ABAC [12] is a potential model for context-sensitive access control. However, it is still not a privacy-aware mechanism, as it does not allow users to set different levels of detail for the shared information. The work in [13] proposes an alternative mechanism for pervasive scenarios. Making use of semantic technologies, users can define access control policies by providing conditions over context. However, this mechanism also requires a centralized knowledge base that contains contextual information from all users, which takes data ownership away from users and is unsuitable for opportunistic scenarios. Furthermore, it does not allow users to control access at different levels of granularity.

While privacy-aware and context-sensitive access control mechanisms are addressed in the literature, the fact that none of these mechanisms are designed for opportunistic scenarios makes none of them suitable for our envisioned pervasive environments. Efforts in privacy-aware access control are not often coupled with context-sensitivity and *vice versa*, and those that address both do not consider the special needs of opportunistic data sharing for empowering users to manage access to their own data.

IV. Threat Model

In our system model, a user, known as provider, voluntarily shares their data or resources with others, known as consumers, as long as both the provider and the consumer meet certain conditions set by the provider. These conditions may constrain the provider's own context, the consumer's context, or a combination of the two. The provider should also be able to filter or obfuscate the information it provides based on similar conditions over context. We assume a model of an attacker whose objective is to obtain information they are not granted access to by exploiting any weakness of the system. Attackers can have up to three roles: they can be consumers, and therefore they try to obtain information from providers; they can be providers, trying to obtain information from their contacts with consumers; or they can be third-party attackers, trying to obtain information from messages exchanged by other providers and consumers.

We build a threat model for decentralized access control for opportunistic data sharing incrementally, first considering general threats that are common to any access control mechanism and then moving to more concrete threats that are specific to our context-aware and opportunistic approach.

General threats:

- *Unauthorized access.* The most basic threat in which a consumer tries to obtain data when the provider desires to deny access.
- *Circumventing context constraints*, consumers get access to information because of their identity while it should be denied according to other contextual attributes.

Privacy threats:

- *Consumer over-exposure.* Providers obtain precise information from consumers using the context information they reveal to request access.
- *Provider over-exposure.* Consumers access information with a finer granularity or higher precision than the provider intended to grant access to.
- *Insider attack.* Consumers get finer-grained information than the provider's policies allow by correlating and aggregating the results of repeated allowed requests.

Third-party attacks:

- *Eavesdrop attack*. The attacker obtains the messages shared between providers and consumers, acquiring privileged information.
- *Replay attack*. The attacker obtains a legitimate message and replays it to get incorrectly granted access.

All of the above threats are directly addressed in our approach. There is another threat that is present in the current version of our approach, and it will be tackled in future work, the *insider multi-type correlation attack.* In this threat, a consumer could correlate different types of contextual information obtained legitimately to obtain other kinds of data not allowed. It is similar to an insider attack, but instead of correlating multiple queries to the same information type, the attacker correlates single queries to multiple information types.

V. The PADEC Conceptual Model

In this section, we present our approach to context-sensitive and privacy-aware access control for opportunistic data sharing. We do this incrementally, adding concepts to the model to address the threats elicited in the previous section. Throughout this section, we designate with underlines the key concepts that are the primary novel contributions of PADEC.

Devices in PADEC take on one of two roles: a data or resource *provider* and a data or resource *consumer*. A provider has some application-level data or service that a consumer desires to access; this data or service is provided through the exposure of an *API endpoint*For instance, in our case study, tourists can obtain the history of a local resident's device by calling an API endpoint that releases it.

A key tenet of PADEC is a strict *separation of concerns* between application-level functionality and PADEC's privacy and access control. From an application perspective, this means that the consumer should perceive that it directly calls the provider's application-level API endpoint; pragmatically, this request passes through PADEC components to determine access, but the application (on both sides) remains unaware of the details. Throughout this section, we use sequence diagrams to depict PADEC's details. The consumers are shown on the left of these diagrams, while the providers are shown to the right. Each side has two threads: the application threads (at the outside) and the access control mediating threads (in the center).

A. Step 0: Encrypted communication

To address the *eavesdrop attack* and the *replay attack*, we start with an assumption that all communications are protected by either symmetric or asymmetric cryptography (e.g., AES or RSA). We assume both devices have strong public and private

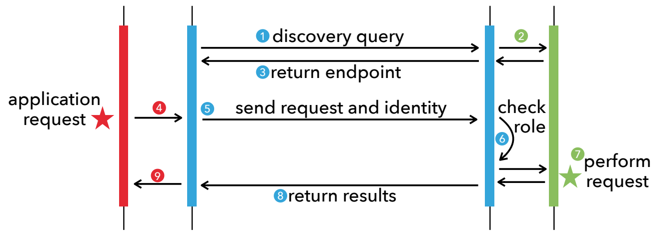


Fig. 1. *Step 1: Providing simple access control based on identity.* A consumer discovers an endpoint in the surroundings (1-3) and makes an application request (4), which is sent to the provider via the available network (e.g., an opportunistic device-to-device connection) (5). The provider checks the consumer's identity against the list of granted roles (6) and, if access is granted, the provider performs the request (7) and returns the results (8-9).

keys, or share a strong symmetric key. These keys are assumed to be shared through a secure, out-of-band channel.

Since messages are encrypted, eavesdropping is not a concern, since the attacker cannot examine the message content. The *replay attack* is also thwarted, since any answer the attacker might get by replaying messages will be as unreadable as any message they overhear via eavesdropping.

B. Step 1: Controlling access to API endpoints

The main, basic concept behind preventing *unauthorized access* is to differentiate users who are authorized to access an endpoint from those who are not. A simple approach is for each endpoint to have an allow list of user identities that are authorized to access the endpoint. Managing this mechanism on an endpoint-by-endpoint basis does not scale: managing multiple lists for different endpoints on an individual identity basis quickly explodes. Therefore, it can be desirable to create *groups*, and, instead of adding individual users to lists, users are added to groups and groups are added to allow lists. The reader might find this system familiar; if the term groups is replaced with *roles*, this is a simple version of RBAC [10]. This concept is shown in Fig. 1.

C. Step 2: Very basic context-aware access control

In many pervasive computing applications, it is necessary to support access control mechanisms that are sensitive to their context. In particular, it may be desirable to provide some level of access to complete strangers who have never before been encountered, as long as some condition on their context holds. Access lists are not sufficient because it is not possible to register an unknown person in a group. Furthermore, this type of access control needs to consider context attributes other than identity, such as the consumer's location or activity. A simple approach is to allow endpoint providers to specify more expressive rules that set conditions over context in order to grant access, e.g., *a consumer must be within 50 meters of the endpoint's position*. To support this approach, consumers must share their context information with the endpoint provider as part of their request, so that the rule can be checked, ensuring that the *circumventing context constraints* threat is mitigated. This system is also familiar, since the same concept is captured in ABAC [12]. This mechanism is depicted in Fig. 2.

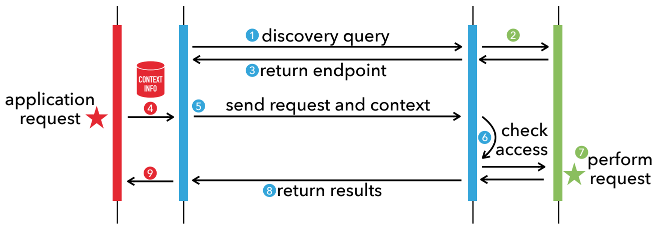


Fig. 2. *Step 2: Extending access control to consider the consumer's context.* The consumer provides its context information as part of the request (4-5), and the provider's process for determining access requires executing the more expressive rules associated with the endpoint (6).

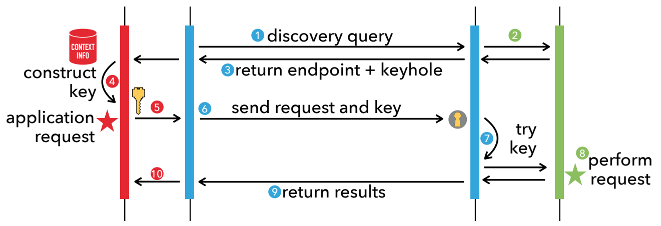


Fig. 3. *Step 3: Keys and keyholes.* In this step, PADEC adds *keyholes* (3), which define the context information required to access the endpoint, and *keys* (4-6), which a consumer application constructs to encapsulate the context information requested. The provider's keyhole defines a rule that validates the provided key (7) before granting access.

D. Step 3: Keys and keyholes

While the previous step enables a provider to consider context in access control, the approach has a few limitations. From a privacy standpoint, this requires the consumer to release *all* of context information that might be relevant without relying on any information about what might actually be needed or used. From a system efficiency standpoint, this is also an overhead in terms of communication cost. While this overhead may be negligible in a centralized system, where the context information may be stored and checked on a cloud server, it may prove prohibitive in opportunistic networks.

To reduce the amount of shared context information both for privacy and overhead reasons, the next step is to define a keyhole associated with an endpoint. The keyhole provides an access point for the API endpoint on the provider side and defines one or more contextual attributes that the consumer is required to provide to gain access. When the consumer discovers a nearby available endpoint (step (1) in Fig. 3), the consumer simultaneously discovers the associated keyhole. To access the endpoint, the consumer assembles a key for each attempted access via the keyhole; the key contains the consumer's current contextual values for the attributes requested by the keyhole. Key construction may also be constrained by the consumer's own local policies, which may limit or prevent the release of certain personal information.

On the provider's side, using only the name of the invoked endpoint and the data in the key, PADEC's access control mechanisms determine whether the access is granted. Practically, each keyhole is associated with a rule that is evaluated at run-time over the consumer's key. With this approach, only relevant contextual attributes have to be released, lowering the

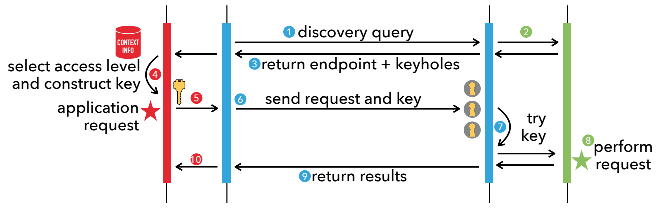


Fig. 4. *Step 4: Providing access options.* Discovery now includes multiple keyholes for each endpoint (3), and the consumer chooses one to target (4-6).

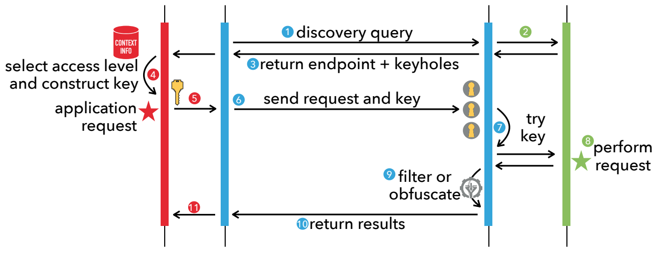


Fig. 5. *Step 5: Increasing providers' privacy flexibility.* After completion of the application-level API call, PADEC applies a filter associated with the selected keyhole (9) before returning the results to the consumer (10-11).

amount of revealed information and the size of keys that are communicated. This step aims to prevent the *consumer over-exposure* threat described previously. In addition, keys are not revealed to the application-level provider, but remain solely within PADEC, further protecting the consumer's privacy. On the other hand, keyholes impose an overhead, as consumers need to query for the keyhole of an endpoint before sending the key. These exchanges are depicted in Fig. 3.

E. Step 4: Providing access options

As a next iterative step, PADEC can allow a provider to set multiple rules (and hence, define multiple keyholes) to control access to a single endpoint. The goal is the corollary of the previous step, i.e., to prevent *provider over-exposure*. For instance, the rule *the consumer must be within 50 meters of my position* or *belong to the group "friends" to access this endpoint* can be decomposed into two, one that requires the consumer's location and the other that requires the consumer's identity. Before decomposition, the keyhole for the above rule would request *both* location and identity; after decomposition, the consumer sees two different keyholes and can choose which information to provide. In PADEC, we refer to these different keyholes as access levels; when a consumer discovers the endpoints, the provider also returns the keyholes of all the access levels, and the consumer can choose which one they want to attempt to access. This refined process is shown in Fig. 4. A consumer can choose the order in which it attempts to access the available keyholes, depending on the consumer's own sensitivity to private context information.

F. Step 5: Increasing providers' privacy flexibility

Up to this point, if a consumer gets access to an endpoint through any of its keyholes, the consumer gets access to all the information in that endpoint. The next layer places filters on top of access levels and their associated keyholes. In particular, a provider can attach a different filter to each access level exposed by an API endpoint; the filter can be used to abstract or obfuscate the results returned from the endpoint. Thus, rather than having to duplicate endpoints to provide different levels of access, the provider can use a single endpoint but post-process the results before returning them to the consumer. In this way, PADEC separates the application-level functionality of the provider endpoint from the concerns associated with privacy and access control, which are implemented within the keyholes and filters. This additional step is shown in Fig. 5.

In PADEC, a filter can be any technique used to limit the precision of the information returned by an endpoint. PADEC has basic built-in filters, but these are also user-definable by injecting small pieces of code or tailored parameters.

Finally, to assist consumers in selecting appropriate keyholes, the discovery information that is returned to the consumer (i.e., step (2) in Fig. 5) includes the achievable level of precision for each of the available keyholes. To avoid *insider attacks*, filters should be implemented in such a way that guarantees idempotence, i.e., repeated access grants to the endpoint should consistently provide access to the same information after obfuscation as long as the unobfuscated information does not change.

G. Step 6: Negotiating multi-level access

Finally, as we discussed above, there is a limitation in the design of PADEC's keyholes and access levels: a consumer cannot tell in advance which access levels will be granted and which will be denied, which may result in several back-and-forth interactions as the consumer tests different keyholes to attempt to gain the needed access. This has potential significant negative ramifications in terms of communication overhead, especially as the application scales in size.

To mitigate the impact of this limitation, PADEC includes a negotiation component. Rather than providing exactly a single matching key in each request, consumers in PADEC can provide a personalized key that the provider can try against multiple keyholes. Upon receiving the request, the provider will try to use the key in the access level with the highest precision. If access is not granted, the provider will automatically resort to the next access level, and so on. If no attempt succeeds, access is effectively denied.

H. A Final Note about encrypted communication

As we described at the beginning of this section, our iterative design of PADEC assumes that all communications are encrypted, to ensure protection against eavesdropping and replay attacks. These mechanisms rely on cryptographic keys exchanged among all pairs of devices out-of-band. However, to keep all interactions purely opportunistic, it is possible to embed the key exchange into the messages sent as part of PADEC by integrating a key exchange protocol similar to Diffie-Hellman [20] into the endpoint discovery process. Taking the protocol in Fig. 5 as the final PADEC model. The consumer can send its public key with its discovery query in (1). As part of returning the endpoint and keyholes, the provider can create a new symmetric key and encrypt it with the consumer's public key before returning it. The consumer (and only the consumer) has the associated private key, which it can use to retrieve the symmetric key. This symmetric key can be used to encrypt the subsequent communications in the exchange (in particular exchanges (6) and (10) in Fig. 5).

VI. Evaluation

To evaluate PADEC, we first measure the trade-off between the *privacy level* achieved by the system and the successful accesses to endpoints in each of the steps explained in Sec. V. We then quantify the *overhead* of the system in each of the steps. Third, we explicitly compare the *expressiveness* of PADEC with that of alternative mechanisms, namely RBAC and ABAC. In fact, the first sets of metrics – privacy, successful accesses, and overhead – also allow comparison across PADEC, RBAC, and ABAC, since Step 1 in the PADEC protocol is equivalent to RBAC and Step 2 is equivalent to ABAC. Finally, we provide a discussion of PADEC's success in addressing the *threat model* described in Section IV.

Our primary mode of evaluation is through the implementation of PADEC as an application layer in the ONE [21] simulator. We incorporated a streetmap of New York City from OpenStreetMap [22], in which the network nodes are of two kinds: tourists visiting the city and local residents. From a PADEC architectural perspective, tourists are consumers of data, and their movements are constrained to the touristic zones of the map (i.e., central downtown squares). The nodes representing local residents are PADEC providers that roam through touristic and residential zones of the city. As our evaluation moves into evaluating our threat model, a subset of each group is designated as attackers. The simulated scenarios consider 100 local residents and 50 tourists, and each simulation runs for 24 hours of simulated time. In the threat model evaluation, we consider 90 honest locals, 40 honest tourists, 10 malicious locals and 10 malicious tourists. Every 50 seconds, if a tourist is not waiting for a response to a request, the tourist chooses a random connected local and starts a PADEC communication, using a simple broadcast communication across the opportunistic network. It is important to note that both tourists and locals may keep moving during these interactions, and thus, their messages may be routed through the opportunistic network instead of being delivered directly.

As data to support the application scenario, we construct POI histories for the providers using a set of FourSquare check-ins in New York City between April 2012 to February 2013 [23]. We generate a realistic POI history for each simulated resident using the anonymized user IDs in the dataset. Each provider has an average of 210 check-ins, and the full dataset contains 227,428 check-ins. Each consumer collects three pieces of context that can be shared with providers to gain access to their POI histories. Each type of context has a different category of sensitivity from the consumer's perspective. Higher categories are considered to be more sensitive, i.e., to reveal more private information. The context types we use are: (1) *identity*, which each consumer perceives as the most sensitive attribute, which we refer to as category 3; (2) *location* (category 2); and (3) *sound level* (category 1). For our initial experiments, in Steps 1 through 3, we assume that all tourists are willing to reveal any context information to gain access to a provider's POI history endpoint. From Step 4 onwards, we assume that half of the tourists remain willing to reveal any information, while the other half will only reveal categories 1 and 2 (but not the category 3 identity information).

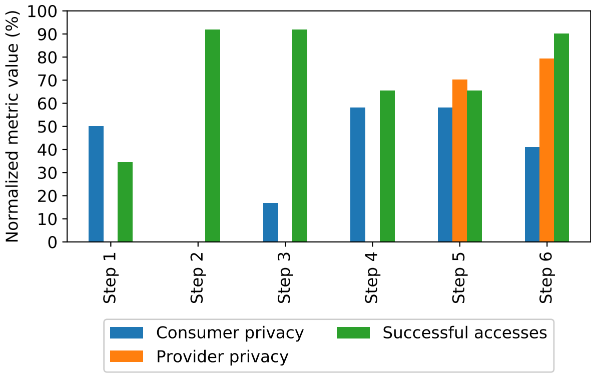


Fig. 6. Privacy vs. successful accesses trade-off in each PADEC step.

Each of the six steps is also associated with one or more rules defined by the provider to constrain access to the endpoint. For each of the six steps, we use the following rules:

Step 1 The provider grants access only to users whose identities place them in the *friend* group (14% of the tourists) or *family* group (6% of the tourists).

Step 2 The provider also grants access to *any* tourist within 500 meters of the provider.

Step 3 Same as Step 2.

Step 4 The rules are separated into two access levels (one for friends and family and a second for nearby strangers).

Step 5 Filters are associated the defined levels. The results for friends and family are not filtered, while those for nearby strangers only shows POIs visited at least three times between October and January.

Step 6 Same as Step 5.

Privacy vs. access. Our first evaluations consider the trade-offs that users navigate with respect to revealing their private information versus gaining or granting access. Fig. 6 shows these trade-offs for each step of PADEC. We present three metrics: consumer privacy, provider privacy, and number of successful accesses. Consumer privacy is based on the average sum of the categories of the attributes shared, so that 0% means all attributes are shared and 100% means no attributes are shared. Higher values for this metric therefore indicate a higher degree of consumer privacy. Provider privacy is based on the average precision of the information shared, so 0% means all of the raw POI information is shared (100% precision) and 100% means no POI information is shared (0% precision). Finally, successful accesses computes the percentage of the consumers' requests that were successfully answered.

Step 1 only considers identity, so the exposure of the con-

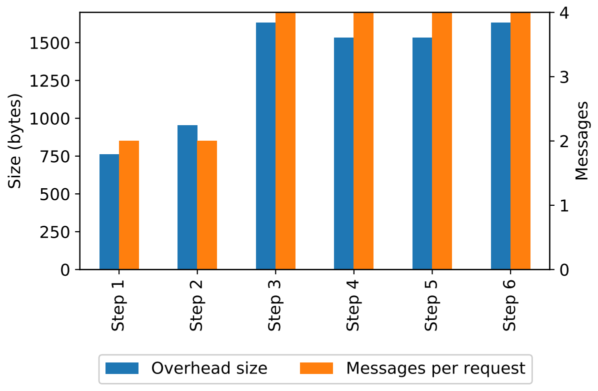


Fig. 7. Overhead in size and number of messages for each PADEC step.

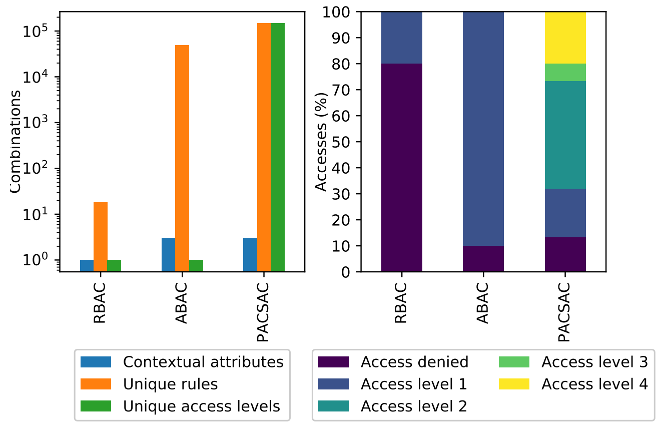


Fig. 8. Comparative expressiveness of RBAC, ABAC and PADEC.

sumers reflects the sensitivity of the context released, but also the fact that two context types are not shared. Because only family and friends are granted access, 65.57% of requests are denied. In Step 2, using other contextual attributes dramatically increases the percentage of fulfilled requests, but a consumer's privacy is completely exposed (i.e., all consumers share all of their context). In Step 3, the keyhole allows consumers to reveal slightly less information in their keys, increasing their privacy with no impact on successful accesses. The two access levels in Step 4 greatly increases the privacy of consumers, as they only have to reveal one contextual attribute; but consumers try the access level with the best precision given the context data they are willing to release. This means many try to access through the keyhole that requires identity, but they are not always friends and family, so 34.42% of the requests fail. Step 5 adds filters, which greatly improves the privacy of providers and has no impact on the other two metrics. Finally, the negotiation algorithm of Step 6 greatly increases successful accesses, getting up to 90.16% of the requests being satisfied, with a modest impact on consumer privacy.

Overall, while the increase of consumer privacy is not as large as the increase of provider privacy, there is an important feature of PADEC to consider: by design, the contextual information shared in the key can only be accessed by PADEC. The underlying applications are unable to access this information, so providers running malicious applications cannot collect data from the consumers. Compared to the information shared by providers, which can be read by underlying applications, consumer data is much more protected.

Overhead of PADEC. We consider two metrics for overhead: the number of messages sent during an exchange like the one shown in Fig. 5 and the total amount (in bytes) of data exchanged. Fig. 7 shows the overhead. While RBAC and ABAC (as Steps 1 and 2) have a low overhead, PADEC almost doubles this overhead in the later steps. PADEC has to share four messages (keyhole request, keyhole information, key, and response), whereas RBAC and ABAC only need two (authorization request and response). Although these two additional messages increase the overhead, the total overhead of PADEC in the context of this application scenario is 1.63 KB.This doubling of the communication overhead is the primary price an application pays to be able to leverage PADEC's capabilities. However, since the overhead of ABAC and RBAC is already very low considering current communication technologies, doubling the overhead is acceptable.

Comparative assessment of expressiveness. To evaluate the *expressiveness* of the relevant access control models, we compare the performance of ABAC and RBAC to a simulation in the same scenario featuring a PADEC endpoint with five access levels. Though PADEC allows for further access levels, these five levels are sufficient to demonstrate PADEC's expressiveness. We constrain ourselves to RBAC and ABAC because they are implementable in an opportunistic device-to-device network; DySP-RBAC, in contrast, requires a central registry with data that relates different users.

Our comparison is shown in Fig.8, which shows a theoretical comparison on the left and a practical demonstration on the right. From a theoretical perspective, we characterize the size of the rule space, i.e., how many different rules could be specified in each approach. We consider the use three context attributes as well as four operators and two combinational operators (and, or) to join rules. If we limit rules such that each combinational operator can be used only once (preventing infinitely long rules), we can compute the total possible number of attributes, rules, and access levels, as shown on the left of Fig. 8. Since ABAC and PADEC allow the use of contextual attributes other than identity, they are more expressive than RBAC. PADEC is, theoretically, the most expressive, nearly tripling the number of possible rules in comparison to ABAC. Furthermore, PADEC allows each rule to be associated with a filter, allowing multiple levels of privacy for each endpoint, while RBAC and ABAC can only set a single rule over an endpoint.

Threat model mitigation. Finally, we examine the degree to which PADEC addresses the threat model in Section IV. We used a scenario in which 10 consumers are attackers and 10 locals are attackers assigned each of the threats. Since it is not possible to access the key from an underlying application, *consumer over-exposure* attacks have been implemented in third-party nodes. In total, 37,109 attempts of various attacks were performed for *circumventing context constraints*, *consumer over-exposure*, *eavesdropping* and *replay* attacks. *Unauthorized access* attacks were proven to fail in the previous simulations, since accesses were denied. As for *insider attacks*, filters for each access level are, by design, idempotent, which makes it impossible to correlate responses from the same access level, as responses do not contain different information from one another. A total of 0 attempts were successful in the simulation, and therefore, we conclude that PADEC is resistant against attacks from the proposed threat model.

We have validated PADEC against the case study in Section II. PADEC provides tools for consumers and providers to share data opportunistically. The results show that PADEC can be leveraged to increase privacy with a negligible overhead compared to alternative access control mechanisms.

VII. Conclusion and future work

As interest in systems that leverage sensors of companion devices (such as the interest in mobile crowd sensing) continues to grow and develop, users become more wary about privacy of their data. To empower them and maintain data ownership, opportunistic data sharing systems have evolved, but they require expressive, context-sensitive, and privacy-aware access control mechanisms. We developed PADEC to directly address these concerns, protecting the privacy of providers and consumers in opportunistic scenarios with a much higher expressiveness and minimal overhead compared with alternatives such as ABAC or RBAC. In the future, we expect to address the *insider multi-type attack*.

Acknowledgment

Map data copyrighted OpenStreetMap contributors and available from https://www.openstreetmap.org. This work was supported by the projects RTI2018-094591-B-I00 (MCI/AEI/FEDER,UE), 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 European Regional Development Fund, and by the National Science Foundation (CNS-1703497). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the sponsors.

References

[1] C. Wigginton, M. Curran, and C. Brodeur, "Global mobile consumer trends, 2nd edition," 2020. [Online]. Available: https://www2.deloitte.com/content/dam/Deloitte/us/Documents/technology-media-telecommunications/us-global-mobile-consumer-survey-second-edition.pdf

[2] EMarketer, "US adult wearable penetration rate 2016-2022," EMarketer, Tech. Rep., 2019. [Online]. Available: https://www.statista.com/statistics/793800/us-adult-wearable-penetration/

[3] J. Guillén, J. Miranda, J. Berrocal, J. García-Alonso, J. M. Murillo, and C. Canal, "People as a service: A mobile-centric model for providing collective sociological profiles," *IEEE Softw.*, vol. 31, no. 2, pp. 48–53, 2014.

[4] B. Guo, Z. Yu, X. Zhou, and D. Zhang, "From participatory sensing to Mobile Crowd Sensing," in *2014 IEEE PerCom Workshops*. IEEE Computer Society, 2014, pp. 593–598.

[5] M. Abouaroek and K. Ahmad, "Foundations of opportunistic networks," *Opportunistic Networks: Mobility Models, Protocols, Security, and Privacy*, 2018.

[6] J. Huang, L. Kong, H. N. Dai, W. Ding, L. Cheng, G. Chen, X. Jin, and P. Zeng, "Blockchain-Based Mobile Crowd Sensing in Industrial Systems," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 10, pp. 6553–6563, oct 2020.

[7] L. Pelusi, A. Passarella, and M. Conti, "Opportunistic networking: Data forwarding in disconnected mobile ad hoc networks," *IEEE Communications Magazine*, vol. 44, no. 11, pp. 134–141, nov 2006.

[8] Y. Huang, A. Shema, and H. Xia, "A proposed genome of mobile and situated crowdsourcing and its design implications for encouraging contributions," *International Journal of Human-Computer Studies*, vol. 102, pp. 69 – 80, 2017, special Issue on Mobile and Situated Crowdsourcing.

[9] K. Olejnik, I. Dacosta, J. S. Machado, K. Huguenin, M. E. Khan, and J. Hubaux, "Smarper: Context-aware and automatic runtime-permissions for mobile devices," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 1058–1076.

[10] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Computer role-based access control models," pp. 38–47, feb 1996.

[11] A. K. Malik and S. Dustdar, "Enhanced sharing and privacy in distributed information sharing environments," in *2011 7th International Conference on Information Assurance and Security (IAS)*. IEEE, 2011, pp. 286–291.

[12] V. C. Hu, D. R. Kuhn, and D. F. Ferraiolo, "Attribute-based access control," *Computer*, vol. 48, no. 2, pp. 85–88, feb 2015.

[13] A. Toninelli, R. Montanari, L. Kagal, and O. Lassila, "A semantic context-aware access control framework for secure collaborations in pervasive computing environments," in *Lecture Notes in Computer Science*, vol. 4273. Springer, Berlin, Heidelberg, 2006, pp. 473–486.

[14] Withlocals B.V., "Withlocals - personal tours & travel experiences." [Online]. Available: https://play.google.com/store/apps/details?id=com.withlocals.Withlocals

[15] E. Hayashi, S. Das, S. Amini, J. Hong, and I. Oakley, "CASA: Context-aware scalable authentication," in *Proceedings of the 9th Symposium on Usable Privacy and Security*. New York, New York, USA: ACM Press, 2013, p. 1.

[16] D. Schurmann, A. Brusch, S. Sigg, and L. Wolf, "BANDANA - Body area network device-to-device authentication using natural gAit," in *PerCom 2017*. Institute of Electrical and Electronics Engineers Inc., may 2017, pp. 190–196.

[17] M. L. Mazurek, Y. Liang, W. Melicher, M. Sleeper, L. Bauer, G. R. Ganger, N. Gupta, and M. K. Reiter, "Toward strong, usable access control for shared distributed data," in *12th USENIX Conference on File and Storage Technologies*, 2014, pp. 89–103.

[18] S. Shafeeq, M. Alam, and A. Khan, "Privacy aware decentralized access control system," *Future Generation Computer Systems*, vol. 101, pp. 420 – 433, 2019.

[19] S. Popov, O. Saa, and P. Finardi, "Equilibria in the tangle," *Computers & Industrial Engineering*, vol. 136, pp. 160–172, 2019.

[20] E. Bresson, O. Chevassut, and D. Pointcheval, "Provably secure authenticated group diffie-hellman key exchange," *ACM Transactions on Information and System Security (TISSEC)*, vol. 10, no. 3, pp. 10–es, 2007.

[21] A. Keränen, J. Ott, and T. Kärkkäinen, "The ONE Simulator for DTN Protocol Evaluation," in *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*. New York, NY, USA: ICST, 2009.

[22] OpenStreetMap contributors, "Planet dump retrieved from https://planet.osm.org ," https://www.openstreetmap.org, 2017.

[23] D. Yang, D. Zhang, V. W. Zheng, and Z. Yu, "Modeling user activity preference by leveraging user spatial temporal characteristics in LBSNs," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 45, no. 1, pp. 129–142, 2015.

Appendix H

Meeting Stringent QoS requirements in IIoT-based Scenarios

ID	: **GLOBECOM 2020**
Authors	: Juan Luis Herrera, Paolo Bellavista, Luca Foschini, Jaime Galán-Jiménez, Juan Manuel Murillo, Javier Berrocal
Publication type	: Conference full-paper
Conference	: IEEE Global Communications Conference (GLOBECOM) 2020
DOI	: `https://doi.org/10.1109/GLOBECOM42002.2020.9322328`
Conference rating	: International conference, GGS Class 2 (A-)

Meeting Stringent QoS Requirements in IIoT-based Scenarios

Juan Luis Herrera*, Paolo Bellavista†, Luca Foschini†, Jaime Galán-Jiménez*, Juan M. Murillo* and Javier Berrocal*
*Department of Computer Systems and Telematics Engineering, University of Extremadura, Spain.
†Dipartamento di Informatica-Scienza e Ingegneria, University of Bologna, Italy.
[jlherrerag, jaime, juanmamu, jberolm]@unex.es, [paolo.bellavista, luca.foschini]@unibo.it

***Abstract*—The Industrial Internet of Things (IIoT) provides automation solutions for industrial processes through the interconnection of different sensors, actuators and robotic devices to the Internet, enabling for the automation of manufacturing processes through Factory Automation. However, IIoT processes are often critical, and require very high Quality of Service (QoS) to work properly, as well as network scalability and flexibility. Fog computing, a paradigm that brings computation and storage devices closer to the edge of the network to enhance QoS, as well as Software-Defined Networking (SDN), which enables for network scalability and flexibility, can be integrated into IIoT architectures in the form of fog nodes that integrate both, computation resources and SDN capabilities, to meet these needs. However, the QoS of the IIoT system depends on the placement of these fog nodes, creating a need to obtain placements that optimize QoS in order to meet the requirements by minimizing the latency between the fog nodes and the IIoT devices that consume their services. In this paper, this fog node placement problem is formalized and solved by means of Mixed Integer Programming. We also show relevant experimental results of our formulation and analyze its performance.**

I. Introduction

With the rise of the Internet of Things (IoT) in recent years, the interest on integrating Internet-connected devices into industry to simplify and automate industrial processes has also been increasing, leading to the creation of the Industrial Internet of Things (IIoT) [1]. Some of these IIoT applications, such as Factory Automation (FA), aim at fully automating manufacturing processes. However, IIoT applications such as FA are characterized by having very strict requirements for their Quality of Service (QoS), such as cycle times of 1 millisecond [1].

These QoS requirements call for architectures that are specifically built to support them. Currently, the cloud computing paradigm is extensively used in other IoT fields. However, meeting the strict QoS of critical IIoT applications such as FA can be complicated due to the physical distance between cloud servers and IIoT devices. This has brought paradigms such as fog computing, which brings computation and storage resources closer to IIoT devices, to the scene. Fog computing can be an important enabler for critical IIoT applications [2], mainly to meet response time requirements. Since these fog nodes are closer to end users, data has to go through shorter paths, and thus, it takes less time to transmit.

Another key aspect on IIoT is scalability. In order to support FA, the network needs to account for scalability and flexibility, while still delivering a high enough QoS to meet its requirements [1]. In this respect, the Software-Defined Networking (SDN) paradigm provides a decoupling between the data and the control planes, which enables high flexibility and scalability in these networks and can play a critical role in IIoT [1], [3].

It is because of this that architectures that combine both fog computing and SDN are a key enabler for IIoT, allowing to meet the stringent QoS requirements of FA while delivering flexibility and scalability. In [4], some of the authors of this work presented a proposal for a self-adaptive framework that combines both fog and SDN aimed at IIoT, as well as for a specific Fog Node (FN) for the architecture. The FN includes an SDN switch and a virtualization platform to provide IIoT services. The FN is designed to host IIoT services for applications such as FA to meet its stringent QoS requirements, and thus allows for an integration of fog computing and SDN aimed at IIoT QoS-strict applications.

However, as also tested in [4], the placement of this FN impacts on the QoS of the architecture. While these tests are limited to two scenarios, i.e. deploying the FN in the local network and deploying it in a remote network, it shows that the placement of the FN can affect the QoS of IIoT applications. Therefore, if these IIoT applications have strict QoS requirements, such as FA, it is key to place the FN in such a way that QoS is optimized. Otherwise, the possibility of using QoS-strict IIoT applications could be at risk.

While optimizing the placement is a way to optimize QoS, it is not always possible to make the distance between a single FN and IIoT devices short enough to meet the QoS needs of FA. In these cases, the placement of more than one FN has to be optimized. Moreover, there is also a need to optimize the routes between FNs and IIoT devices, so that each IIoT device consumes the services of the FN that provides the best QoS to it. These routes can then be implemented in the network using SDN, allowing for a transparent solution for FA and other IIoT applications [3]. The problem of placing a certain number of FNs optimally in the network, finding which FNs provide the best latency to which IIoT devices and finding the optimal routes between those, so that latency between them is minimized and QoS is therefore optimized, is what we label the Fog Node Placement Problem (FNPP). In this work, we provide a solution for the FNPP that, when applied on a certain architecture, is able to find all these results.

The main contributions of this paper are:

- The formalization of the FNPP.
- The formulation of the solution to the FNPP using Mixed Integer Programming (MIP).
- An analysis of the results of applying the solution to architectures under different circumstances.

The remainder of this paper is structured as follows. Sec. II motivates the FNPP. Sec. III explains the system model used in the FNPP. Sec. IV includes the problem formulation as a MIP optimization problem in order to find its solution. Sec. V presents details of our test environment and experimental results. Finally, Sec. VI concludes the paper.

II. Motivation

The FNPP is mainly motivated by IIoT applications that have very stringent QoS requirements, such as FA. In these applications, functionality is inherently related to QoS, and the QoS to meet is very strict, e.g. 1 ms response time [1]. To this respect, it is crucial to minimize response time in order to meet this requirement.

There are two methods to improve response time of a system, either the computing QoS (i.e. execution time) or the networking QoS (i.e. latency) have to be improved [5]. Fog computing provides a solution that enables for both, bringing computation resources closer to IIoT devices to provide them with services that require very low latency [2]. Moreover, the SDN paradigm provides scalability and flexibility to the network, two key requirements of FA [1]. Therefore, SDN and fog computing should be integrated in IIoT scenarios to meet the needs of FA.

The proposal in [4], namely the FN, enables for the integration of fog computing and SDN, while being designed specifically for IIoT. FNs can therefore meet both necessities of FA, by providing IIoT services close to the IIoT devices that consume them as well as enabling for a flexible and scalable network by including an SDN switch. However, the placement of FNs is key for the QoS of the IIoT services they host, with a sub-optimal placement being able to put the feasibility of a FA system at risk, i.e. the solution to the FNPP is key to meet the QoS needs of FA.

Although the need for QoS, flexibility and scalability of FA systems and similar IIoT applications, manifested in the form of a need for fog computing and SDN, can be solved by using FNs, this creates a need for optimizing their placement. This is the need that mainly motivates this paper. Therefore, the main goal of this paper is to provide a solution to the FNPP in form of a MIP formulation to optimally assess the placement of FNs in a specific IIoT scenario.

To the best of our knowledge, no prior works tackle the problem of placing a set of fog nodes in such a way so that QoS is optimized. However, both the optimization of QoS through the placement of equipment and the optimization of fog infrastructures in different ways are currently active research topics [6]–[8].

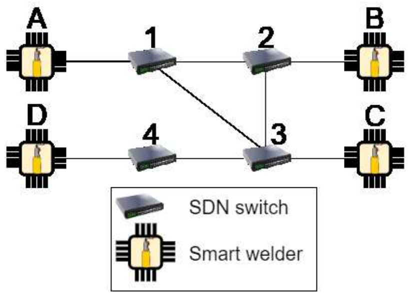


Figure 1. Example system model.

On the one hand, different works have conducted research on optimizing the placement of a SDN controller to optimize the QoS of a network [6]. On the other hand, optimization of QoS in fog infrastructures is an open topic in research. [7] proposes an optimization based on a placement of services in fog infrastructures that violates QoS requirements as least as possible. [8] optimizes QoS by finding the optimal node in a fog infrastructure to execute a certain service dynamically and to integrate these optimal solutions with a SDN controller so they are applied transparently.

The main differences between the works presented and this work are mainly related to their focus. The works related to SDN controllers aim at optimizing the QoS of the network, as opposed to optimizing the QoS of IIoT applications instead. On the other hand, related works on the optimization of fog infrastructures are focused on placing services on already placed FNs, not on placing the FNs themselves.

III. System model

In this section, the FNPP is explained in detail with an example model. For this model, a FA IIoT application for automated welding will be considered. In this application, each smart welder constantly senses the item in front of it. If it is a piece of metal, it notifies an smart welding service to retrieve a command on how to weld it. In this particular example, the system is comprised of four smart welders and four SDN switches, connected in a topology as shown in Fig. 1.

Since the smart welders have to quickly perform cycles in which they sense metal, send the information to the service and retrieve commands, the smart welding service runs on an FN to meet its strict latency requirements. This FN contains a SDN switch while also hosting IIoT services, in this case, the smart welding service. Therefore, one of the SDN switches shown in Fig. 1 should be replaced by an FN. However, any of the shown switches could potentially be replaced by an FN. To meet the strict latency requirement, the FN cannot be more than one hop away from a smart welder. Therefore, there is a need to solve the FNPP in order to meet the requirement.

If switch 1 were to be replaced by an FN, A, B and C would meet the requirement. D, however, would be two hops away, and thus, it would not meet the requirement. Similar results are obtained if the FN replaces switches 2 or 4. If it were to replace switch 3, however, it would be possible to meet the latency constraint, given that A would not route through switch 2. Therefore, it is needed to obtain the optimal placement for the FN and the optimal routes to be followed by the traffic of every smart welder in order to meet the stringent QoS requirement that has been set.

Another approach could be replacing two switches by FNs instead. In that case, some smart welders would consume the service from one FN, while the rest of smart welders would consume it from the other one. In that case, if one of the FNs is placed on switch 3 and another one is placed on switch 1, D should consume the service from the FN on switch 3. If it were to consume it from switch 1, even with an optimal route, it would be two hops away. Therefore, when multiple FNs are considered, not only the placement of the FNs and the routes followed by the traffic are important, the mapping between IIoT devices and FNs is too.

While the FNPP can be solved manually in small cases like these, scalability is crucial in IIoT [1]. The definition of the FNPP can be reduced to the definition of the facility location problem adding flow constraints, and is therefore an NP-hard problem [9]. Thus, when these architectures grow, testing every single placement, route and mapping manually can become unbearable in terms of time.

IV. Problem formulation

In order to solve the FNPP in larger architectures, we present a formulation using MIP, that can be used with automatic MIP solvers to solve the FNPP.

To define the FNPP, the network topology is represented as a directed graph $G = \{V, L\}$, where V are the vertices and L are the links. A link $l_{ij} \in L; i \neq j; i, j \in V$ links together vertices i and j. Let C_{ij} be the capacity of the link l_{ij}, so that $C_{ij} = 0 \forall l_{ij} \notin L$.

We also make certain assumptions on the network topology. Namely, we assume that:

- The vertices are either SDN switches or IIoT devices.
- The amount of SDN switches that should be replaced by FNs is given as input, and should be 1 or greater.
- All FNs have the same capacity.
- The combined capacity of all FNs is enough to deal at least with all the traffic in the network, and each FN has enough capacity to deal at least with the traffic from a single host.

With these assumptions in mind, let $H \in V$ be the set of vertices that are hosts and $S \in N$ be the set of vertices that are switches. Thus, $V = H \cup S$ and $H \cap S = \emptyset$. S will therefore be the set of possible placements for FNs. Similarly, every host $h \in H$ generates an amount of traffic $\phi_h \geq 0$ that has to be processed by its mapped FN. An FN has the capacity to process up to α traffic per unit of time.

Let $P(i,j) = \{L', V'\}$ be the shortest path from vertex i to vertex j, which traverses vertices V' and links L', so that $L' \subseteq L; V' \subseteq V$. Let $D(P(i,j)) = |V'| - 1$ be the amount of vertices traversed in path $P(i,j)$ (e.g. a path that directly connects i with j would have $D(P(i,j)) = 1$). Let each link have a propagation latency of $\beta_{l_{ij}} \geq 0$ and every switch have a processing latency of $\beta_S \geq 0$. Let $L(i,j) = \sum_{l_{ab} \in L'} \beta_{l_{ab}} + (D(P(i,j)) - 1)\beta_S$ be the latency of sending traffic from vertex i to vertex j.

Let θ be the upper limit for the number of FNs to be placed. Concerning the set of variables, let $X_i, i \in V$ be a binary variable that will be 1 if an FN is placed on vertex i and let $Y_{ij}, i, j \in V$ be a binary variable that will be 1 if vertex i is *mapped* to the FN placed in j. Finally, let f_{ij}^h be a binary variable that will be 1 if the traffic generated by h is routed through the link l_{ij}.

The set of notations is summarized in Table I.

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. IIoT 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 an FN per unit of time
$P(i,j)$	Shortest path from vertex i to vertex j
$\beta_{l_{ij}}$	Propagation latency of link l_{ij}
β_S	Processing latency of a SDN switch
$L(i,j)$	Latency of traffic sent from vertex i to vertex j
θ	Maximum number of FNs to be placed
Variable	**Meaning**
X_i	Boolean to determine if an FN is placed in vertex i
Y_{ij}	Boolean to determine if vertex i is mapped to the FN located in vertex j
f_{ij}^h	Boolean to determine if traffic generated by host h is routed through link l_{ij}

Table I
List of notations

Then, the FNPP solution can be formulated as follows:

$$\min \sum_{i \in V} \sum_{j \in V} L(i,j) Y_{ij} \tag{1}$$

subject to:

$$i \in V, h \in H : \sum_{j \in V} f_{ij}^h - f_{ji}^h = \begin{cases} 1 & \text{if } i = h \\ -Y_{hi} & \text{otherwise.} \end{cases} \tag{2}$$

$$\forall l_{ij} \in L : \sum_{h \in H} f_{ij}^h \phi_h \leq C_{ij} \tag{3}$$

$$\sum_{i \in V} X_i \leq \theta \tag{4}$$

$$\forall i \in V : \sum_{h \in H} \phi_h Y_{hi} \leq \alpha X_i \tag{5}$$

$$\forall h \in H : \sum_{i \in V} Y_{hi} = 1 \tag{6}$$

$$\forall s \in S : \sum_{i \in V} Y_{si} = 0 \tag{7}$$

$$\forall h \in H : X_h = 0 \tag{8}$$

$$\forall i,j \in V, h \in H, l_{i'j'} \in L : X_i, Y_{ij}, f^h_{i'j'} \in \{0,1\} \quad (9)$$

Eq. 1 expresses the optimization objective, i.e. to minimize the sum of the latencies from each host to its mapped FN. Eq. 2 represents an adaptation of the classical flow conservation constraint to this scenario, while Eq. 3 constrains the total traffic routed by a link to be less than its capacity. Eq. 4 constrains the number of FNs: there must not be more than θ. Eq. 5 controls the capacity of an FN, so the amount of traffic directed to that FN is never more that it can handle. Equations 6 and 7 will make sure that each host is mapped to exactly one FN and that switches are mapped to no FNs. Finally, Eq. 8 assures no FNs will be set up on host vertices. Eq. 9 simply makes these variables binary.

This mathematical formulation represents the FNPP, allowing it to be solved in different network topologies with different characteristics for the FN or the IIoT devices, providing the optimal placement, routes and mapping to meet the strict latency requirements of IIoT applications such as FA.

V. Performance evaluation

In this section, a performance evaluation of our solution to the FNPP to analyze its performance under different circumstances is presented. Analyses of latency varying the number of FNs placed, traffic, number of hosts in the network and the criteria used to place FNs have been performed to test the latency reduction of our formulation. Analyses of link load varying the amount of hosts in the network have also been performed to test its impact.

A. Evaluation environment

The formulation has been evaluated in four different topologies to analyze scalability and execution time. Namely, we have used small (12 switches, 15 links), medium (17 switches, 26 links), large (22 switches, 36 links), and extra-large (50 switches, 88 links) SDN topologies, generated synthetically.

We have considered different parameters for these simulations. Parameter α is the capacity of the FN in Mbps. Since different topologies have different amounts of traffic, and thus require different values for α, the comparisons for multiple topologies have considered α expressed as a percentage of the total traffic in the network instead of in Mbps. It is important to keep in mind that not every value for α is valid, since there must be enough overall capacity to process all the traffic on the network, as well as to have enough capacity on an FN to, at least, process all the traffic coming from a single node, as Eq. 10 shows.

$$\alpha \geq max(max(\phi_h \forall h \in H), \frac{sum_{h\in H}\phi_h}{\theta}) \quad (10)$$

Parameter β_S is the processing latency of a SDN switch in milliseconds. We have considered three values for it, concretely $\beta_S \in \{0.15, 0.76, 2.21\}$, retrieved from [10].

In order to evaluate the impact of the number of FNs to be placed in the network and the amount of hosts, the θ parameter as described in Sec. IV and a γ parameter that sets the amount of hosts per switch have been considered. The traffic on the network does not change regardless of γ, and is thus divided equally between the hosts of each switch. Traffic analyses have been performed by multiplying this traffic by factors between 0.5 and 1.0.

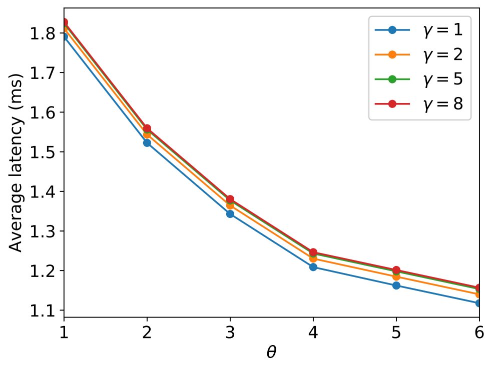


Figure 2. Average latency vs θ in medium topology.

B. Performance analysis

The first analysis that is proposed aims at evaluating the average latency on the medium topology varying θ and γ, while each fog node can process all the traffic in the network, as Fig. 2 shows. When θ increases, latency lowers in a negative log likelihood. This is because, when more FNs are placed, they can generally be placed closer to their mapped hosts, thus reducing latency, until they reach the switch they use to access the network. Once there, it is not possible to place FNs any closer. Therefore, the latency decrease is very significant when θ is low and FNs are in intermediate positions. However, it gets less steep as θ increases and FNs are closer to hosts, thus the negative log likelihood. As for the number of hosts per switch, it slightly influences latency, and its influence is not greatly affected by θ.

The second analysis aims at evaluating the average latency on the medium topology varying the traffic load, as Fig. 3 shows. In this analysis, the values for FN capacity and θ have been experimentally established in such a way that FNs are stressed when traffic rises. As we can see, when the network is not heavily loaded (i.e. traffic scaling between 0.5 and 0.9), latency is stable. However, in stress situations (i.e. traffic scaling of 1.0), there is a latency spike. This is because some areas produce more traffic than others, and thus, once the closest FN to the area is at its full capacity, the remaining hosts in that area have to be mapped to an FN in another area, and thus, further away, increasing the latency for these hosts. The main difference γ makes is about mapping few IIoT devices with high resource requirements or many IIoT devices with low resource requirements, which influences how many hosts have to be mapped to other areas and how they are mapped.

Fig. 4 shows the average latency on the medium topology as a function of α. The objective of this analysis is to evaluate

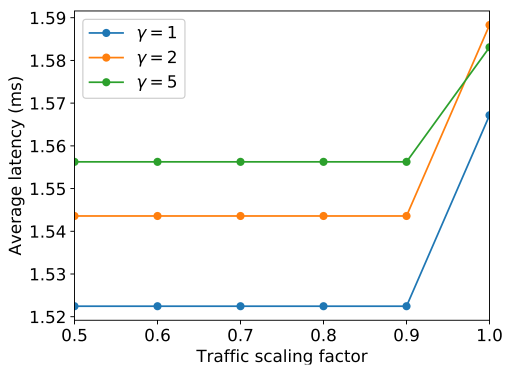


Figure 3. Average latency vs traffic in medium topology.

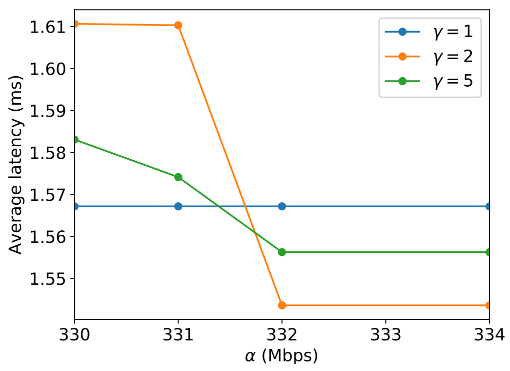


Figure 4. Average latency vs α in medium.

the average latency varying the FN capacity. It is clear that the number of hosts per switch affects the effects of α: the higher γ is, as long as it is larger than 1, the less steep the latency reduction is. On the other hand, it is also shown that a higher α generally improves latency. This is because of the effect commented earlier: a higher α makes each area be less overloaded with traffic, and thus, it makes the hosts in said area able to direct all their traffic to the FN in their area. The higher α is, the less traffic has to be directed to FNs in areas further away. However, each host can only direct its traffic to a single FN, hence, $\gamma > 1$ improves this effect, as different hosts can be mapped to different FNs. It can also be seen that lower γ values have a lower latency with low values of α but a higher latency with high values of α, compared to other values of γ.

In order to evaluate the scalability of our proposal, the next analysis evaluates the link load in different topologies. Figures 5, 6 and 7 show the empirical CDF of the link load in different topologies. The size of the topology and the number of hosts per switch are both key to this respect. In general, it can be seen that the links of larger topologies are less loaded than those of smaller topologies. These patterns are also replicated when considering γ: a higher γ makes the CDF steeper. γ also has a major role on the number of unused links: a lower γ

Table II
AVERAGE EXECUTION TIMES OF THE SIMULATIONS IN MEDIUM TOPOLOGY

γ	θ	Time (s)	γ	θ	Time (s)
1	1	0.708	5	1	36.069
	2	1.680		2	35.239
	3	0.878		3	30.325
	4	0.830		4	30.320
	5	1.139		5	32.957
	6	0.953		6	31.205
2	1	3.490	8	1	136.489
	2	7.582		2	143.020
	3	3.017		3	138.938
	4	2.996		4	128.652
	5	3.584		5	189.306
	6	2.906		6	135.589

leaves more links unused. One of the main reasons for this is that a higher γ adds an additional link for each host, that will always be used. Thus, these additional links make the proportion of overall unused links lower.

Fig. 8 shows a comparison between choosing the placement of the FNs by using different graph theory-based techniques and choosing the placement by using the formulation proposed in Sec. IV, labeled as *Optimal*. In all of the cases, the host-FN mapping and the traffic routing were performed by means of the formulation in order to minimize latency. As Fig. 8 shows, the minimum latency achieved by each technique is quite different when $\theta = 1$. However, the performance gap shrinks when θ is larger. In any case, placing the FNs in the nodes with the highest betweenness centrality gives the best results out of the three techniques. Nonetheless, there is a large performance gap between using any of these techniques and using the proposed formulation, although it shrinks when θ is large enough. This is because, when θ is larger, more FNs can be placed, and thus, it is more likely to place FNs at their optimal placements.

Finally, Tables II and III show the results of an analysis to evaluate the scalability of our solution on large networks based on the mean execution time of the simulations that considered our formulation. The results for the medium topology can be seen in Table II. In the medium topology, the MIP solver usually takes more time to solve the problem when $\theta = 2$, while it usually manages to keep similar times in the rest of cases, taking into account that γ is the most important parameter that influences execution time, since more hosts per switch means an overall larger topology, and, as it can be seen in Table III, the size of the topology is a key parameter on execution time. As for the times themselves, the worst time is for solving the FNPP in the medium topology with $\gamma = 8, \theta = 5$ takes roughly over 3 minutes. On the other hand, in smaller topologies, such as the small or medium topology with $\gamma = 1$, the FNPP can be solved in about one second. In the middle ground, it takes roughly 25 seconds to solve the FNPP in the extra-large topology.

In the IIoT domain, these results show that a high amount of FNs is crucial to minimize latency, and that assessing their placement, mapping and routing by using our proposal can

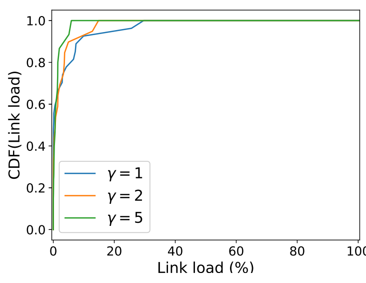


Figure 5. Empirical CDF of link load in small topology.

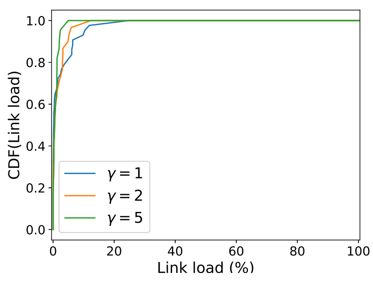


Figure 6. Empirical CDF of link load in medium topology.

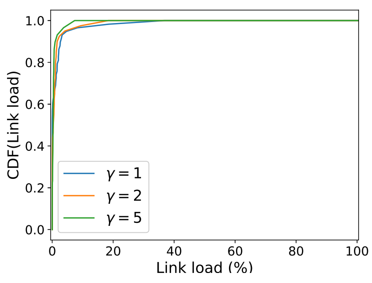


Figure 7. Empirical CDF of link load in large topology.

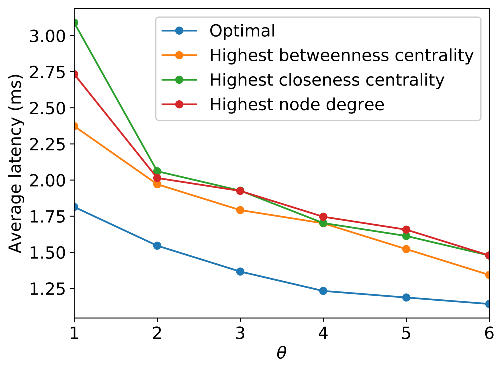


Figure 8. Average latency using different solutions in medium topology.

Table III
AVERAGE EXECUTION TIMES OF THE SIMULATIONS IN DIFFERENT TOPOLOGIES

Topology	$\lvert V \rvert$	$\lvert L \rvert$	Time (s)
Small	24	27	0.810
Medium	34	43	1.912
Large	44	58	1.902
Extra-large	100	138	25.100

further minimize this latency. By repeatedly solving the FNPP and installing its routes and mappings on the SDN controller, it is possible to adapt the placement over time to minimize latency, all while being transparent to the IIoT devices.

VI. Conclusions and Future Work

In this paper, we have introduced, formulated and proposed a solution for the problem of placing one or more fog nodes to optimize the latency of IIoT applications, i.e. a solution for the FNPP. Furthermore, we have tested the solution formulation on different network topologies with different parameters and we have analyzed the effects of these parameters in network latency. We have also tested the scalability of our solution in different circumstances.

Our formulation is able to optimize the placement and mapping of FNs, as well as to optimize the routing of the traffic that each node offloads to its fog node in tractable time. Therefore, it is possible to use the proposed solution over time, thus adapting FN placement, mapping and routing to fit dynamicity. The speed of these adaptations is dominated by execution time, and thus, it is not possible to adapt the placement, mapping and routing in larger networks in less than a few minutes.

In the future, we expect to analyze the impact of adding FNs in terms of cost, as well as to develop heuristics to improve scalability and execution time, so this problem can be solved in a shorter time in large networks, and thus its adaptation time can be shortened. We also expect to evaluate the performance of our formulation and these heuristics in real or emulated network test-beds.

VII. Acknowledgements

This work has been partially funded by the project RTI2018-094591-B-I00 (MCI/AEI/FEDER,UE), 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 and Infrastructure of the Government of Extremadura (GR18112, IB18030), and by the European Regional Development Fund.

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] 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.

[3] 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.

[4] I. Bedhief, L. Foschini, P. Bellavista, M. Kassar, and T. Aguili, "Toward self-adaptive software defined fog networking architecture for IIoT and industry 4.0," in *Proceedings of IEEE CAMAD 2019*, 2019.

[5] ITU-T, *Definitions of terms related to quality of service*, International Telecommunication Union Telecommunication Standarization Sector Std. E.800, Rev. 09/2008.

[6] T. Das, V. Sridharan, and M. Gurusamy, "A Survey on Controller Placement in SDN," *IEEE Communications Surveys & Tutorials*, pp. 472–503, 2019.

[7] O. Skarlat, M. Nardelli, S. Schulte, and S. Dustdar, "Towards QoS-Aware Fog Service Placement," in *Proceedings of IEEE ICFEC 2017*, aug 2017, pp. 89–96.

[8] R. P. Singh, J. Grover, and G. R. Murthy, "Self organizing software defined edge controller in IoT infrastructure," in *ACM International Conference Proceeding Series*, 2017, pp. 1–7.

[9] P. B. Mirchandani and R. L. Francis, *Discrete location theory*, 1990.

[10] P. Emmerich, D. Raumer, S. Gallenmüller, F. Wohlfart, and G. Carle, "Throughput and Latency of Virtual Switching with Open vSwitch: A Quantitative Analysis," *Journal of Network and Systems Management*, vol. 26, no. 2, pp. 314–338, 2018.

Appendix I

The Service Node Placement Problem in Software-Defined Fog Networks

ID	: **ISCC 2020**
Authors	: Juan Luis Herrera, Luca Foschini, Jaime Galán-Jiménez, Javier Berrocal
Publication type	: Conference full-paper
Conference	: IEEE International Symposium on Computers and Communications (ISCC) 2020
DOI	: `https://doi.org/10.1109/ISCC50000.2020.9219694`
Conference rating	: International conference, GGS Class 3 (B)

The Service Node Placement Problem in Software-Defined Fog Networks

Juan Luis Herrera*, Luca Foschini†, Jaime Galán-Jiménez* and Javier Berrocal*
*Department of Computer Systems and Telematics Engineering, University of Extremadura, Spain.
†Dipartamento di Informatica-Scienza e Ingegneria, University of Bologna, Italy.
[jlherrerag, jaime, jberolm]@unex.es, luca.foschini@unibo.it

Abstract—**Nowadays, cloud computing has become a key paradigm in distributed applications thanks to the rise of low-power Internet-connected devices as commonplace. However, stringent Quality of Service (QoS) requirements are complicated to achieve when a pure cloud computing paradigm is applied, due to the physical distance between end devices and cloud servers. This motivated the appearance of fog computing, a paradigm that adds computation and storage resources, named fog nodes, closer to the end devices in order to reduce response time and latency. However, the placement of fog nodes, as well as the relative placement of the end devices each fog node serves, can affect the QoS obtained. This can be crucial to those services that have stringent QoS requirements. In this work, we analyze the effects that different placements of fog nodes have on QoS and present the problem of placing fog nodes to obtain an optimal QoS, with a focus on the Industrial Internet of Things domain because of its strict QoS requirements. We conclude that an optimized placement of the fog nodes can minimize latency to support the QoS requirements of IIoT applications.**

Index Terms—**Software-Defined Networking, fog computing, Quality of Service, Internet of Things**

I. Introduction

Nowadays, the Internet of Things (IoT) and the rise of mobile devices has made these devices ubiquitous. A 59% of all Internet-connected devices in 2018 were smartphones or IoT devices [1]. However, these devices, specially IoT devices, usually have severe energy and cost constraints that make them lack the power to perform computational-heavy tasks. A common solution to this problem is to offload these tasks to the cloud [2]. While this paradigm is commonly used, the fact that cloud servers are usually in the core of the network can make high Quality of Service (QoS) complicated to obtain [3]. New paradigms have emerged, such as fog computing [3], that bring part of the computation and storage resources closer to the edge in an attempt to support these strict QoS requirements.

A key QoS requirement is low response time, since not all IoT applications are delay-tolerant. The main strategy used in fog computing to support it is to place fog servers closer to the users, so the path data has to go through is shorter, and thus, it takes the data less time to go through it.

While fog computing is an interesting paradigm for QoS-strict applications, it does not come without its challenges. Some of these challenges, such as seamless service delivery when IoT devices move to different parts of the network, can be tackled transparently for the applications if the network itself deals with them [4]. In the past, it would have been very difficult to adapt the behaviour of the network to tackle these challenges. However, the Software-Defined Networking (SDN) paradigm enables network administrators to take care of them. The core idea behind SDN is to decouple the data and control planes, by making SDN switches merely execute orders in the data plane and centralizing the control plane in the SDN controller. An SDN controller is an entity that defines rules for the switches to follow. The main feature of SDN controllers is that they can be programmed, thus allowing for the creation of *network-level applications*, that can alter the behaviour of the network under concrete circumstances for different purposes, such as taking care of the aforementioned challenges in fog computing. This makes architectures that combine fog computing with SDN interesting because of the extra benefits that come from combining them [4].

These architectures that combine SDN and fog computing can benefit the IoT domain, because of its strict QoS needs derived from its interaction with the physical world [3], [5], [6]. The Industrial Internet of Things (IIoT), which integrates IoT in industry, has some of the applications with most stringent QoS requirements (e.g. response time under 1 ms) [7]. This calls for robust physical architectures that are able to guarantee this high level QoS for IIoT applications.

In these combined SDN and fog architectures, fog nodes can be essentially divided in two kinds: those that provide services that require a high level of QoS, and those that only provide services that are tolerant to lower QoS. We label the former ones *Service Nodes* (SNs), because they generally provide services that are critical to IIoT applications. If a fog node provides both kinds of services, it should also be considered as an SN, since it provides critical services. SNs are especially interesting because their placement affects the QoS of the services they provide, and since these services are critical to IIoT applications, their placement may have important effects on these applications, and an optimal placement of the SNs may enhance the QoS of these IIoT applications. While the latter ones are also interesting, the QoS requirements of their services are not as strict, and thus, to place them optimally may not be as interesting. In this paper, we focus on SNs and the effects of their placement in QoS. Therefore, this paper focuses on architectures that only contain network equipment, SNs and IoT devices.

From the networking point of view, the end hosts of these architectures are IoT devices that are constantly gathering data through sensing and sending this sensed data to an SN for its processing. Since the processes performed at the SN have strict QoS requirements, mainly related to response time, the latency between the SN and the IoT device is of critical importance. In [7], a model for SNs is presented and its performance is evaluated, deriving the conclusion that the latency between an SN and end devices can be reduced by placing the SN closer to the end devices [7]. However, the experiments on [7] are limited to two possible placements for the SN: within the local network and in a remote network. While this is enough to support some QoS requirements, it may not be enough for IIoT applications with very strict QoS needs. In these applications, it would be desirable to place the SN optimally so the latency between the SN and the IoT device is minimal.

However, it may not always be possible to optimize latency by optimizing the placement of a single SN. Some networks might be large enough so that even the optimal placement provides a QoS lower than required. This can be solved by adding multiple SNs instead, and placing them all optimally. In essence, this would be similar to dividing the network into multiple zones and placing an SN optimally in each of the zones, so that IoT devices in that zone use that SN. The division in zones should also be performed in an optimal way in terms of QoS, so the optimal placement of the SNs in each of these zones is also the overall optimal placement for the SNs.

In this paper, we introduce the Service Node Placement Problem (SNPP), which consists on dividing a network in a set of optimal zones and placing an SN optimally in each zone. To the best of our knowledge, no prior works have introduced this problem.

The main contributions of this paper are the formulation of the Service Node Placement Problem, as well as an analysis of its effects over latency in a series of case studies. The remainder of this paper is structured as follows. Section II motivates the SNPP. Section III explains the model used in the SNPP. Section IV includes the different case studies. Section V discusses the results of the analyzed case studies. Section VI shows related works on the subject. Finally, section VII concludes the paper.

II. Motivation

The motivation of the SNPP is very related to IIoT. IIoT is the application of IoT for industrial purposes [8]. Thus, IIoT connects sensors and actuators to the Internet, with purposes such as smart manufacturing systems [8].

Within IIoT, one key application is factory automation, which consists on having IoT robotic systems to automate the production process [8]. This application requires to be executed periodically in very short time periods (1 ms according to [8]). Thus, it is key to minimize the execution time of factory automation processes so it can be executed in under 1 ms.

Execution time is one of the performance metrics of the system, and is thus related to the QoS of the system [9]. There are two methods to improve the QoS of an application: enhancing the network QoS and enhancing the computing QoS [9]. This can be difficult in cloud computing environments, which are very common in IoT applications [2], because cloud servers are usually far away from the devices.

Fog computing is a computing paradigm proposed by Cisco as complementary to cloud computing [10]. Fog computing is designed to mitigate these QoS issues by placing smaller, less powerful cloud-like nodes closer to the devices, so data has to travel shorter paths, and thus reducing network latency [3].

As explained in Section I, fog architectures can be complemented with Software-Defined Networks for additional benefits [4]. SDN decouples the data and control planes of a network, centralizing the control plane in the figure of the SDN controller, that defines the behaviour of SDN switches by installing rules in them. This brings benefits to fog computing, such as allowing a moving end device to start a request in one point of the network and retrieve the response in a different point in a transparent way for the applications, by installing rules on the switches of the network so the response is routed to the new location of the end device [4].

Among the different approaches and proposals for fog and SDN architectures, we found [7] to be interesting because of its IIoT-centered approach. [7] proposes the Service Node we work with as a self-adaptive fog node that includes SDN capabilities and provides IIoT services, as well as shows comparisons of latency depending on the placement of the Service Node.

In these comparisons, the main conclusion is that placing the SN close to the IIoT devices results in a better QoS [7]. These results, along with the need for QoS of IIoT and the powerful tools given by SDN and fog computing, are the core motivation for the Service Node Placement Problem. Since SNs enhance QoS, and placing SNs close to IIoT devices further improves latency, optimizing the placement of SNs allows for optimization of QoS, and thus, allows for a simpler method to obtain the stringent QoS required by IIoT applications such as factory automation.

III. Service Node Placement Problem

To model this problem, a networking-centric model is used. In this model, the network is a graph, in which vertices are either SDN switches, SNs or IoT devices. Links are the network links between them. The protocol these links use (e.g. Ethernet, Wi-Fi, ZigBee) is not of relevance. These links have a certain latency, that depends on different factors such as the length of the link.

The vertices of the graph are separated in four major categories: SDN switches, classic IP routers, SNs and IoT devices. IoT devices and SNs are assigned to each other, so that a certain IoT device is *mapped* to an SN. Thus, each IoT device produces a certain amount of traffic, that must be processed by its mapped SN. These mappings are also important, since QoS can also be affected by them.

As of the model shown in [7], the SN contains an SDN switch. Thus, we consider that placing an SN in the network consists on replacing an SDN switch with an SN. Since each SN contains an SDN switch, each SN can also perform the same actions as an SDN switch, additionally to the actions that are specific to SNs. SNs can only process a certain amount of traffic per unit of time. This amount is labeled *capacity* of the SN.

On the other hand, SDN switches and IP routers route traffic from IoT devices to SNs. Switches add a certain delay to the packets routed by them [11]. Thus, the SNPP must also consider this delay as a part of latency. While, as stated before, we consider SNs can substitute SDN switches, they cannot substitute classic IP routers. An IP router is not able to perform the exact same actions as an SDN switch (e.g. SDN controller placement should take SDN switches into account, but not IP routers [12]), and thus one should not be replaced by the other one and vice-versa. Moreover, the migration from IP networks to SDN is not within the objectives of the SNPP, and is thus out of its scope. Therefore, the SNPP can be applied to hybrid IP-SDN networks [13], [14] as well as to pure SDN networks, since the SDN switches in them can be replaced by SNs.

Figure 1 shows an example model in a hybrid IP-SDN network. In this example, the network would have 2 SNs (red and blue), 4 SDN switches, an IP router and 12 IoT devices. Each IoT device is mapped to a certain SN, and is highlighted with the same color for clarity. As we can see, each IoT device is mapped to its closest SN to minimize latency.

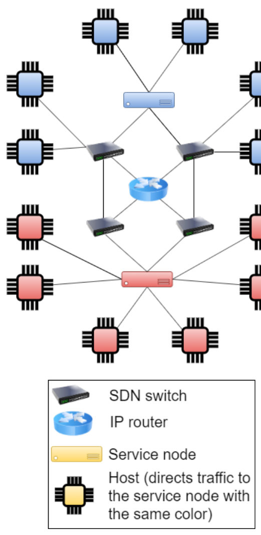

SDN switch
IP router
Service node
Host (directs traffic to the service node with the same color)

Figure 1. Example problem model in a hybrid IP-SDN network.

In this problem model, the SNPP has multiple objectives. If there is a single SN, a solution for the SNPP should find the optimal placement for the SN, as well as the optimal routes from every IoT device to the SN in order to maximize the QoS (e.g. to obtain minimal latency). However, if there are multiple SNs to be placed, the solution to the SNPP should first find the optimal mappings between IoT devices and potential SNs. In other words, the solution to the SNPP should first find an optimal strategy to divide the network in different areas, taking into account that each area will contain a single service node, that is, every IoT device in the same area will be mapped to the same SN. Once the division of the network in areas has been obtained, the solution to the SNPP should obtain the optimal placement for a single SN in each area, as well as the optimal routes from IoT devices in the area to the SN in the same area. Therefore, solving the SNPP with multiple SNs implies dividing the network in areas first to then solve an SNPP with a single SN on each area.

The SNPP should also be solved differently depending on the QoS dimension that is optimized. For example, the placement of an SN to minimize latency can be different from the placement of an SN to maximize reliability. It is also possible to solve the SNPP having as an objective to meet a certain constraint, instead of optimizing a QoS dimension (e.g. instead of solving the SNPP to minimize latency, it is possible to solve the SNPP so that latency is below a certain threshold, such as the maximum allowed latency of an IoT application).

IV. Case studies

In the previous sections, we define the SNPP and the system model we follow. In this section, we apply this model to IIoT case studies.

A. *Case study A: Small Factory Automation*

Factory Automation is an IIoT process that requires a high QoS, including a maximum latency of 1 millisecond [8]. In this case study, automation of an assembly line is considered. This assembly line has 10 robotic arms that are able to perform the same functions as a human operator [8], and each arm is also equipped with a set of sensors (e.g. cameras, proximity sensors, etc.) that allows the arm to sense the items that are within its reach. The considered network is an IP-SDN hybrid network [13], [14] with the topology shown in Figure 2. For the sake of simplicity, we assume that every link is 200 meters long, and thus, its propagation latency is of 0.0006 ms. The latency of IP routers is of 0.38 ms [15], while the one of SDN switches is of 0.15 ms [11].

Each arm works in a continuous cycle, in which it senses the items within its reach and sends the sensed information to a control service. The task of this service is to determine what are these items and how should the arm react. The control service will then send these commands to the arm so it reacts appropriately. In this case, a single SN will provide this service, and the effects of its placement are analyzed.

There are three SDN switches that could be replaced by this SN, namely, the SDN switch on the bottom, the one on the center and the one on the right. The effects on latency that depend on the SN placement are measured using two metrics: maximum latency among all arms to the SN and average latency between every arm and the SN [12].

First, the effects of placing the SN in the rightmost switch are analyzed. With this placement, the arms that are currently connected to this switch would be directly connected to the

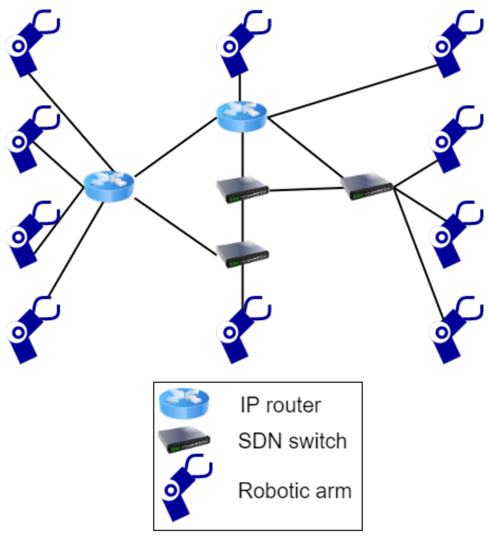


Figure 2. Model of case study A.

SN. Assuming the shortest routes are used, the arms connected to the IP router on the top would be one hop away from the SN, the arms connected to the IP router on the left would be two hops away and the arm connected to the SDN switch on the bottom would be two hops away as well. Since IP routers add more latency than SDN switches, the arms with the maximum latency would be those that are connected to the IP router on the left, which would have 0.76 ms of latency. If an alternate, three-hop route is taken through the SDN switches at the bottom and center, this latency is lowered to 0.68 ms. On the other hand, the average latency would be of 0.378 ms.

Next, the differences between these effects and the ones produced by placing the SN in the bottom switch are analyzed. Assuming the shortest routes are used, the arm on the bottom would be directly connected to the SN, the arms connected to the switch on the right would be two hops away, the arms connected to the router on the top would be two hops away as well and the arms connected to the router on the left would be one hop away. The arms with the highest latency in this case would be the ones connected to the router on the top, with 0.532 ms of latency, which is lower than the maximum latency when the SN is placed on the rightmost switch. The average latency is of 0.345 ms, 0.033 ms lower than when the SN is placed on the rightmost switch.

Finally, the effects of placing the SN in the switch on the center are analyzed. Assuming the shortest routes are used, the arms connected to the leftmost router would be two hops away, while every other arm would be one hop away. It must also be noted that the traffic of the arms connected to the leftmost router should be routed through the SDN switch on the bottom to minimize latency. Once that is considered, the arms with the highest latency would be the ones connected to the leftmost router, with 0.513 ms of latency. The average latency would, however, be of 0.349 ms, 0.004 ms higher than when the SN is placed on the bottom switch. Therefore, maximum latency is lowered with respect to the other placements for the SN, but the average latency is higher than the latency obtained by placing the SN in the bottom switch.

Overall, the conclusion is that the placement of the service node affects QoS, namely latency; and that the SNPP should be solved with a certain metric in mind. For instance, if the objective is to optimize the average latency, the SN should be placed on the bottom switch. However, if it is maximum latency that should be optimized, the SN should be placed on the switch on the center.

B. Case study B: Large Factory Automation

This case study is similar to the one presented in IV-A. The focus is on a Factory Automation system that automates an assembly line by using robotic arms, which work using the same cycle as the one presented in IV-A. However, in this case study, there are 20 robotic arms in a completely different topology, as Figure 3 shows.

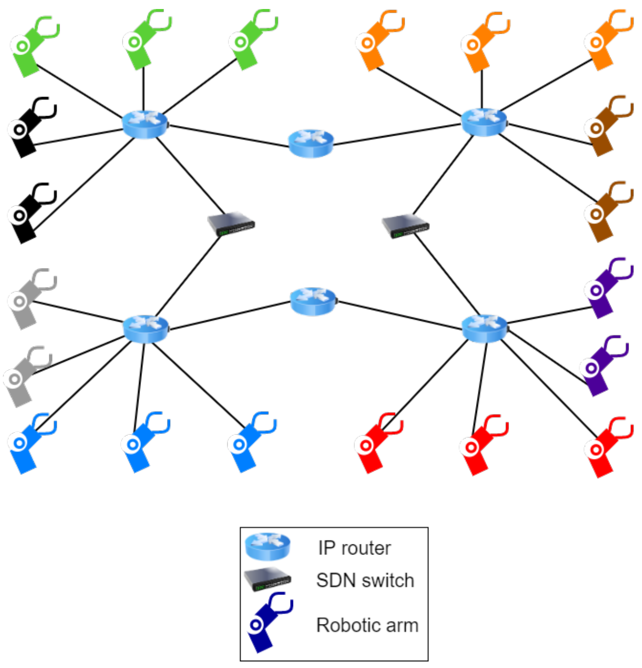


Figure 3. Model of case study B.

In this case study, the effects of different divisions of the network in areas is analyzed. In this case, the network is divided into two different areas. The placement for the SNs is clear, since there are only two SDN switches that can be replaced by SNs. There are many different ways of dividing this network topology into two areas, but in this case study two divisions are analyzed. For the sake of simplicity, the same values for latency and the same metrics as in IV-A are used, and traffic is assumed to be routed through the shortest paths.

The first division strategy is to divide the network into a top area and a bottom area. The top area would include the green, orange, black and brown robotic arms; while the gray, purple, blue and red robotic arms would be in the bottom area. The

SN for the top area would be on the leftmost switch, while the SN for the bottom area would be on the rightmost switch. In such case, the arms with the maximum latency of the top area would be the orange and brown ones, with a latency of 1.142 ms. For the bottom area, the arms with the maximum latency would also have 1.142 ms of latency, being the blue and grey ones. As for average latency, it is also the same in both areas, being 0.762 ms of average latency. Therefore, the maximum overall latency would be of 1.142 ms and the average overall latency would be of 0.762 ms.

The second division strategy is to divide the network into a left and a right area instead. The left area includes the green, black, gray and blue arms; while the left area contains the orange, brown, purple and red ones. The SN for the left area would be on the leftmost switch, and the one for the right area would me on the rightmost one. This strategy has an important impact on latency. With this division, every arm has the exact same latency: 0.381 ms.

The overall conclusion is that, just like placing the SN in one place or another has an effect on latency, the mapping between SNs and IoT devices also affects latency. Thus, even when the placement of SNs is obvious, the SNPP should also be considered so the division strategy followed optimizes latency.

V. Discussion

In Section II, it is shown that the QoS requirements for IIoT applications are very strict, and that fog computing is a paradigm designed to support these strict QoS requirements through the usage of service nodes. Therefore, it has been established that a possible solution to strict QoS needs can be to use fog computing.

However, in Section IV, it is shown that different placements for SNs, as well as different mappings between IoT devices and SNs, also affect QoS and should be considered. Concretely, in IV-A, it is shown that changing the placement of the SN can decrease worst-case latency by up to approximately 23% and average latency by up to approximately 9%. These results depend on the underlying network topology and the possible placements of SNs, but they show that the placement of SNs can increase or decrease the QoS enhancement that may come from using fog computing.

In IV-B, it is shown that the mapping between SNs and end devices is also an important factor on QoS. Concretely, changing the mapping between SNs and end devices lowers worst-case latency by approximately 77% and average latency by approximately 50%. While these decreases depend on the network topology, possible placements for SNs and possible mappings, these results show that the increase in QoS that may be experienced by placing additional SNs may be conditioned by the placement and mapping of these additional SNs.

The importance of these factors is, however, related to the objective of a solution for the SNPP. For instance, if the objective is to optimize average latency, it is key to obtain the best possible placement, such as the bottom switch on IV-A. If the objective of the SNPP is to have an average latency of under 1 ms, however, then any placement presented in IV-A would be a solution for the SNPP, and thus, SN placement might not be that important. Even in this last case, it is also important to note that placement is more important when QoS requirements are more strict. For instance, if the objective of the SNPP is to have an average latency of under 0.35 ms, then placement would be important again, since placing the SN on the rightmost switch on IV-A would not deliver the required QoS.

VI. Related work

The idea that the placement of elements that provide critical services to an infrastructure has effects on the QoS said infrastructure provides can be extensively seen in SDN research. The SDN Controller Placement Problem [12] is a problem similar to the SNPP, but focused on the network. Concretely, the SDN Controller Placement Problem aims at placing an SDN controller or a set of SDN controllers in a certain SDN network so a concrete dimension of QoS is optimized. This problem was first proposed in [12], where the effects over QoS of different placements for the SDN controller were analyzed. [16] presents a comprehensive survey of different works and approaches to this problem. [17] presents a solution for the problem that accounts for dynamic load, adapting the solution to the current traffic load.

On the other hand, the optimization of fog infrastructures has also been tackled by other works. [18] focuses on placing different services in a fog infrastructure with different fog and cloud nodes optimally to use their resources efficiently. [19] provides a solution to allocate the necessary resources for these services, maximizing the Quality of Experience for users by using fuzzy logic techniques. [20] also tackles an optimal placement of services in fog computing infrastructures that minimizes the QoS violations of these services. Other approaches such as [21] proposes to dynamically find the optimal node to provide a service and to integrate these solutions in an SDN controller so the network transparently transmits requests to the appropriate node.

The main differences between these related works and this work are twofold. First, the SNPP aims at optimizing the infrastructure by optimally placing Service Nodes for certain applications, and thus, optimizing the QoS of these applications; instead of optimizing the QoS of the network by placing the SDN controller optimally. Therefore, the focus of the SDN Controller Placement problem is similar, but on the networking dimension; while the focus of the SNPP is on the computing dimension. Secondly, the works on the optimization of fog infrastructures are mostly aimed at placing services optimally, either by explicitly finding the optimal nodes to provide a certain service or by allocating the resources for the services at appropriate nodes. Instead, this work aims at placing the nodes themselves optimally in the infrastructure. To the best of our knowledge, no prior works have tackled the optimization of QoS in a fog infrastructure by placing the fog nodes optimally.

VII. Conclusions

In this paper, the problem of placing a set of service nodes in a certain architecture that combines fog and SDN so that QoS is optimized has been proposed, being the Service Node Placement Problem or SNPP. The SNPP has also been studied in two IIoT case studies, showing the effects of different placements on latency.

We find the SNPP to be interesting on fields such as IIoT because of its stringent QoS requirements. Applying the SNPP to IIoT applications could possibly allow for an enhancement of the QoS of these applications by placing its service nodes optimally.

In the future, we expect to develop a generalized solution for the SNPP so that it can be applied to different cases and different network topologies.

VIII. Acknowledgements

This work has been partially funded by the project RTI2018-094591-B-I00 (MCI/AEI/FEDER,UE), 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 and Infrastructure of the Government of Extremadura (GR18112, IB18030), and by the European Regional Development Fund.

References

[1] Cisco, "Cisco Annual Internet Report (2018–2023)," *Cisco*, pp. 1–41, 2020.

[2] P. P. Ray, "A survey of IoT cloud platforms," *Future Computing and Informatics Journal*, vol. 1, no. 1-2, pp. 35–46, dec 2016.

[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] 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 and Tutorials*, vol. 19, no. 4, pp. 2359–2391, oct 2017.

[5] 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, dec 2019.

[6] S. Pallewatta, V. Kostakos, and R. Buyya, "Microservices-based IoT application placement within heterogeneous and resource constrained fog computing environments," in *UCC 2019 - Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*. New York, NY, USA: Association for Computing Machinery, Inc, dec 2019, pp. 71–81.

[7] I. Bedhief, L. Foschini, P. Bellavista, M. Kassar, and T. Aguili, "Toward self-adaptive software defined fog networking architecture for IIoT and industry 4.0," in *IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks, CAMAD*, vol. 2019-September. Institute of Electrical and Electronics Engineers Inc., sep 2019.

[8] 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.

[9] ITU-T, *Definitions of terms related to quality of service*, International Telecommunication Union Telecommunication Standarization Sector Std. E.800, Rev. 09/2008.

[10] 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.

[11] P. Emmerich, D. Raumer, S. Gallenmüller, F. Wohlfart, and G. Carle, "Throughput and Latency of Virtual Switching with Open vSwitch: A Quantitative Analysis," *Journal of Network and Systems Management*, vol. 26, no. 2, pp. 314–338, apr 2018.

[12] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *HotSDN'12 - Proceedings of the 1st ACM International Workshop on Hot Topics in Software Defined Networks*. New York, New York, USA: ACM Press, 2012, pp. 7–12.

[13] J. Galán-Jiménez, "Legacy ip-upgraded sdn nodes tradeoff in energy-efficient hybrid ip/sdn networks," *Computer Communications*, vol. 114, pp. 106 – 123, 2017.

[14] J. Galán-Jiménez, M. Polverini, and A. Cianfrani, "A scalable and error-tolerant solution for traffic matrix assessment in hybrid ip/sdn networks," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 251–264, 2020.

[15] M. Lee, N. Duffield, and R. R. Kompella, "High-fidelity per-flow delay measurements with reference latency interpolation," *IEEE/ACM Transactions on Networking*, vol. 21, no. 5, pp. 1567–1580, 2013.

[16] T. Das, V. Sridharan, and M. Gurusamy, "A Survey on Controller Placement in SDN," *IEEE Communications Surveys & Tutorials*, pp. 1–1, 2019.

[17] M. T. I. Ul Huque, G. Jourjon, and V. Gramoli, "Revisiting the controller placement problem," in *Proceedings - Conference on Local Computer Networks, LCN*, vol. 26-29-October-2015. IEEE Computer Society, dec 2015, pp. 450–453.

[18] M. Taneja and A. Davy, "Resource aware placement of IoT application modules in Fog-Cloud Computing Paradigm," in *Proceedings of the IM 2017 - 2017 IFIP/IEEE International Symposium on Integrated Network and Service Management*. Institute of Electrical and Electronics Engineers Inc., jul 2017, pp. 1222–1228.

[19] R. Mahmud, S. N. Srirama, K. Ramamohanarao, and R. Buyya, "Quality of Experience (QoE)-aware placement of applications in Fog computing environments," *Journal of Parallel and Distributed Computing*, vol. 132, pp. 190–203, oct 2019.

[20] O. Skarlat, M. Nardelli, S. Schulte, and S. Dustdar, "Towards QoS-Aware Fog Service Placement," in *Proceedings - 2017 IEEE 1st International Conference on Fog and Edge Computing, ICFEC 2017*. Institute of Electrical and Electronics Engineers Inc., aug 2017, pp. 89–96.

[21] R. P. Singh, J. Grover, and G. R. Murthy, "Self organizing software defined edge controller in IoT infrastructure," in *ACM International Conference Proceeding Series*. New York, New York, USA: Association for Computing Machinery, oct 2017, pp. 1–7. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3109761.3158390

Appendix J

Multi-Objective Genetic Algorithm for the Joint Optimization of Energy Efficiency and Rule Reduction in Software-Defined Networks

ID	: **NOF 2020**
Authors	: Jaime Galán-Jiménez, Javier Berrocal, Juan Luis Herrera, Marco Polverini
Publication type	: Conference full-paper
Conference	: International Conference on Network of the Future (NoF) 2020
DOI	: `https://doi.org/10.1109/NoF50125.2020.9249089`
Conference rating	: International conference

Multi-Objective Genetic Algorithm for the Joint Optimization of Energy Efficiency and Rule Reduction in Software-Defined Networks

Jaime Galán-Jiménez*, Javier Berrocal*, Juan Luis Herrera* and Marco Polverini†
*Dpt. of Computer Systems and Telematics Engineering, University of Extremadura, Spain. [jaime, jberolm, jlherrerag]@unex.es
†DIET Department, University of Rome "Sapienza", Rome, Italy. marco.polverini@uniroma1.it

***Abstract*—Although the use of Ternary Content-Addressable Memories (TCAMs) in the flow tables of the Software-Defined Network (SDN) switches increases the efficiency of packets matching procedure, drawbacks such as their large power consumption and the limitation on the number of flow rules that can be installed must be taken into account. This paper tackles the joint problem of power consumption and TCAM size limitation in SDN. By exploiting the Rate Adaptation technique and compression methods, a Multi-Objective Genetic Algorithm is proposed to practically solve it. Simulations on a real network topology show that our proposed solution outperforms other state-of-the-art approaches, both in terms of power saving gains (20% at non-peak TM) and maximum TCAM utilization (5%).**

***Index Terms*—SDN, energy efficiency, TCAM, compression, evolutionary algorithms.**

I. Introduction

THE power consumption problem in communication networks became of great interest for the research community during the last decade. A big volume of works intended to save energy in the wired network infrastructure while ensuring Quality of Service (QoS) requirements have been proposed [1]. The promising irruption of the Software-Defined Networking (SDN) paradigm, where an energy-efficient management can be carried out by means of the logically centralized SDN controller, opened a niche to propose algorithms able to dynamically re-route the traffic by using the less number of active network elements [2], [3].

In SDN, the flow tables of the SDN switches are implemented by means of Ternary Content-Addressable Memories (TCAMs). Their matching flexibility on complex matching patterns (including several fields of headers belonging to different layers) and high lookup performance make TCAMs appropriate for SDN environments. However, their high speed for classifying packets has several associated drawbacks, such as large power consumption [4] and high cost [5]. Consequently, these two factors have an impact on the relatively small number of rules a TCAM is able to store. In particular, commercial switches only support a small fraction of the total number of rules required to operate networks (ranging from 1k to 10k rules, as reported in [6]). This limitation is a challenging problem to be considered when proposing energy-efficient solutions on SDN-based networks [7], [8]. On the one hand, an energy-efficient routing solution is constrained by the upper limit of TCAM size. On the other hand, a reduction in the number of rules that are installed in the flow tables of the SDN switches implies scaling down the required parallel searches for packets matching, which would lead to a reduction in the TCAM power consumption.

For this reason, the TCAM size limitation problem is studied from different perspectives [9]: i) eviction mechanisms [10] are intended to create room in the flow table of an SDN switch by removing inactive or less important rules; ii) compression techniques [11] exploit the use of wildcards and priority mechanisms to shrink the content of the flow table; and iii) split-and-distribute solutions [12] divide the set of rules to be installed in the flow table into subsets which are distributed among the network nodes.

Although some works introduced the problem of taking into consideration TCAM size constraints in Energy-Aware Routing SDN [7], [8], only the ON-OFF (sleeping) approach on links has been considered to reduce the power consumption. In the present paper we also consider the Rate Adaptation technique, in which the power consumption of line cards is proportional to the operational mode (sending rate) it is being used [13]. A higher granularity in the number of operational modes of line cards has been demonstrated to be effective in terms of energy savings, regardless the type of energy functions and energy distributions that are considered in the energy consumption model [14]. In SDN networks, the controller can collect information about traffic load on each link and use it as input for the Rate Adaptation energy-efficient solution, redistribute traffic flows and re-adapt links rates to their new traffic load while preserving connectivity and satisfying link constraints.

In order to solve the joint problem of energy efficiency improvement and TCAM rules reduction, a multi-objective optimization problem has been defined and formalized. The two objectives to be optimized are: i) network power consumption, and ii) number of installed rules in the flow tables. Furthermore, a Multi-Objective Genetic Algorithm (MOGA) has been proposed to practically solve it in tractable times.

As a summary, the main contributions of this work are listed next:

- to consider the Rate Adaptation technique in the power consumption problem on SDN networks where the size of TCAMs is constrained;
- to define the multi-objective optimization problem for fairly considering energy efficiency and TCAM size constraints on SDN networks;
- to derive a multi-objective GA-based heuristic, namely Multi-Objective Genetic Algorithm for Energy Efficiency and Rules Reduction (MOGA-E2R2), to practically solve the problem;
- to compare MOGA-E2R2 with other state-of-the-art solutions on realistic scenarios;

The rest of the paper is organized as follows. Problem definition is described in Sec. II. Sec. III details the proposed MOGA-E2R2 heuristic. Results are reported and analysed in Sec. IV. Finally, Sec. V draws some conclusions and future works.

II. Problem Definition

Let us consider an SDN network modelled as a directed network graph $\mathcal{G} = (\mathcal{N}, \mathcal{L})$, where $\mathcal{N}$ is the set of nodes and $\mathcal{L}$ is the set of unidirectional links. While each link $l_{i,j} \in \mathcal{L}$ connecting node $i \in \mathcal{N}$ to node $j \in \mathcal{N}$ has a capacity of $C^k_{i,j}$ (where $k \in [0, K]$ represents the operational mode the link is currently working on) units to accommodate traffic flows, each node $n \in \mathcal{N}$ has a maximum capacity of C_n rules to describe its forwarding behaviour. Traffic description is given by a traffic matrix $\mathcal{T}$ comprising a set of demands $(s, d) \in \mathcal{T}$ which are sent from each source node $s \in \mathcal{N}$ to each destination node $d \in \mathcal{N}$ in the network. Taking into account the network features described above, the optimization problem this paper aims to solve is defined as follows. Given a network topology $\mathcal{G} = (\mathcal{N}, \mathcal{L})$ and a traffic matrix $\mathcal{T}$, the objective is to find an optimal network configuration which minimizes both the network power consumption and the number of rules to be installed in the flow tables of SDN nodes. For this purpose, a routing configuration which i) respects link capacity constraints; ii) respects TCAM capacity constraints on nodes; iii) results in a minimal network power consumption; and iv) the number of installed rules in the SDN nodes is also minimal.

In order to describe the Mixed Integer Linear Programming (MILP) formulation intended to solve this Multi-Objective and Multi-Commodity-Flow (MO-MCF) optimization problem, a set of variables must be previously defined:

- $x^k_{i,j}$ is a binary variable representing the operational mode of the link $l_{i,j}$, whose value is equal to 1 if the link $l_{i,j}$ is in the operational mode k. If $x^0_{i,j} = 1$, the link is switched off, while if $x^K_{i,j} = 1$ then it is working at full data rate.
- $f^{s,d}_{i,j}$ is a binary variable whose value is 1 if the traffic demand $(s, d) \in \mathcal{T}$ is routed on the link $l_{i,j}$, 0 otherwise.
- τ is a variable representing the number of rules installed on the most used TCAM.

Regarding the power consumption model adopted in the formulation, each link $l_{i,j}$ has an associated power consumption which depends on its operational mode $x^k_{i,j}$. Specifically, the power consumption of the link $l_{i,j}$ when it is in operational mode k is $p^k_{i,j}$ (clearly, $p^0_{i,j} = 0$ and $p^K_{i,j} = p^{\text{MAX}}_{i,j}$). Note that the number of operational modes of a link and their distribution depend on the hardware technology they are implemented with [14]. The classical sleeping behaviour [15], where links can be either powered off or working at their full rate, is therefore modelled by $K = 1$.

After explaining the required variables, the problem formulation is described by eqs. (1-8). In particular, the two objective functions that must be jointly optimized in the MO-MCF problem are defined by eqs. (1-2):

$$f_1 = min \sum_{l_{i,j} \in \mathcal{L}} \sum_{k \in [0,K]} x^k_{i,j} p^k_{i,j} \tag{1}$$

whose main goal is to minimize the global network power consumption, and:

$$f_2 = min \quad \tau \tag{2}$$

which aims at minimizing the maximum TCAM utilization (τ), i.e., the maximum number of rules installed in the most loaded SDN node for traffic steering. Thus, the joint optimization of both functions composes the MO-MCF problem to be optimally solved:

$$min\,[f_1; f_2] \tag{3}$$

subject to

$$\sum_{j \in \mathcal{N}^-_i} f^{s,d}_{i,j} - \sum_{j \in \mathcal{N}^+_i} f^{s,d}_{j,i} = \begin{cases} 1 & \text{if } i = s \\ -1 & \text{if } i = d \quad \forall i \in \mathcal{N}, (s,d) \in \mathcal{T} \\ 0 & \text{if } i \neq s, d \end{cases} \tag{4}$$

$$\sum_{(s,d) \in \mathcal{T}} f^{s,d}_{i,j} \mathcal{T}_{s,d} \leq u_{i,j} \cdot \sum_{k \in [0,K]} x^k_{i,j} \cdot C^k_{i,j} \qquad \forall l_{i,j} \in \mathcal{L} \tag{5}$$

$$\sum_{k \in [0,K]} x^k_{i,j} = 1 \qquad \forall l_{i,j} \in \mathcal{L} \tag{6}$$

$$\frac{1}{C_i} \sum_{(s,d) \in \mathcal{T}} \sum_{j \in \mathcal{N}^-_i} f^{s,d}_{i,j} \leq \tau \qquad \forall i \in \mathcal{N} \tag{7}$$

$$\tau \leq 1 \tag{8}$$

Eq. 4 describes the classical flow conservation constraints. Eq. 5 represents the capacity constraint on links, where the total traffic routed on a link must be less than the available capacity. In particular, the available capacity depends on the current operational mode that is configured in the link $l_{i,j}$. Moreover, term $u_{i,j}$ represents the Maximum Link Utilization (MLU) that can be tolerated by each link in the network. Rule capacity constraint on nodes is defined by eqs. 7-8. The first imposes that the TCAM utilization at node i must be lower than the value of the τ variable. The TCAM utilization is calculated as the ratio between the number of flow rules that are installed at node i (under the assumption that for each flow a single flow rule is installed) and the size of its TCAM. Then, by means of eq. 8, the TCAM utilization on each node is forced to be, at most, equal to 1, guaranteeing the respect of the TCAM size constraint.

As previously introduced, the problem formulation described above falls into the category of MCF problems, which are known to be NP-hard. In order to provide a solution in networks of all size in tractable times, a multi-objective heuristic based on GAs is proposed in the following section.

III. Multi-Objective Genetic Algorithm for Energy Efficiency and Rules Reduction

Evolutionary Algorithms are well suited for solving multi-objective optimization problems [16], whose aim is to find or to approximate a group of trade-off solutions known as Pareto-optimal solution set. In this way, a multi-objective evolutionary algorithm, namely Multi-Objective Genetic Algorithm for Energy Efficiency and Rules Reduction (MOGA-E2R2), is proposed to jointly solve the energy consumption problem and TCAM size limitation in SDN networks. This section outlines the specific considerations of the proposed heuristic, starting from the definition of an individual in the population, the fitness functions used to evaluate their suitability to the problem, the routing scheme adopted to steer the traffic and the methods used to compress the flow tables. Finally, a complexity analysis is reported.

A. MOGA-E2R2 Description

In the next, a description of the main components of the proposed MOGA-E2R2 is provided.

1) *Individual Structure:* Let us denote with $c \in \mathcal{P}$ an individual in the population, representing a potential network configuration as a succession of L genes where the k-th gene, $g_k \in c$, describes the operational mode $x_{i,j}$ of link $k = l_{i,j}$.

2) *Fitness Functions:* Regarding the objective function defined in eq. (3) of Sec. II, two fitness functions, f_1 and f_2, are required to evaluate the suitability of each individual in the population. Concerning the first objective, i.e., network power consumption, eq. (9) is applied for each individual in the population $c \in \mathcal{P}$. Function $P(c)$ assesses the aggregated power consumption of the network configuration represented by the potential solution c, according to the current operational mode p of each link k in the network. The resulting value of the sum is multiplied by ϕ, which is set to 1 if the network configuration mapped by c is feasible, i.e., the TM can be correctly routed without violating any of the constraints reported in eqs. (4)-(8). Otherwise, ϕ takes a value high enough to penalise the corresponding fitness value to such unfeasible chromosome.

$$P(c) = \left(\sum_{g_k \in c} g_k \cdot p_k^{\mathrm{MAX}} \right) \phi, \quad \forall k \in \mathcal{L} \tag{9}$$

The second function in the multi-objective optimization is $f2$ (reported in eq. 2), whose aim is to minimize τ, i.e., the maximum number of installed rules in the most loaded SDN node in the network. In MOGA-E2R2, function Ω of eq. 10 represents the projection of chromosome c on the set R, which contains the number of rules that must be installed in each node of the network if the configuration of chromosome c is used. In the equation, $\lambda(i)$ returns the number of rules installed at node i. Once set R is obtained, eq. 11 returns the number of installed rules in the most loaded SDN node, τ.

$$\begin{aligned} &\Omega : c \rightarrow R \\ &\Omega(g_{i,j}) = \{\lambda(i), \lambda(j)\}, \quad \forall g_{i,j} \in c; \forall i,j \in \mathcal{N} \end{aligned} \tag{10}$$

$$\begin{aligned} &\max(R) = \tau \in R \\ &\text{if} \quad \forall x \in R, x \leq \tau \end{aligned} \tag{11}$$

3) *Routing Scheme:* In the following, the routing scheme adopted to steer the traffic on the network configuration represented by the considered chromosome is explained. For each traffic demand, $(s,d) \in \mathcal{T}$, the selected path to steer such demand is computed following the next steps: i) Dijkstra algorithm is executed to use the shortest path between source and destination[1]; ii) among the set of available paths outputted by Dijkstra with same cost, an utilization variable, $w_{s,d}$, is computed, which represents the utilization of the path $p_{s,d}$. The best path to select is the one with the highest utilization value, computed as the maximum between the most loaded link in the path and the node with the highest free space in its TCAM.

3) *Flow Rules Compression:* Nodes that are approaching their saturation in terms of number of installed rules execute a function with the aim of increasing the space in their TCAMs for installing new rules in the future. In this way, two techniques have been considered, namely *default rule compression* and *wildcard rule compression*. The idea of the former method is to aggregate rules into different subsets according to their outgoing port. Then, the outgoing port with the biggest subset of rules, o, will be considered as the *default rule* in the table, with action equal to *forward to o*. Similar to *default rule compression*, the goal of *wildcard rule compression* is to exploit the flexibility provided by wildcards (noted as *) to aggregate a higher number of rules in order to increase the free space in the TCAMs.

MOGA-E2R2 is executed at the SDN controller side, which has to perform next actions: i) collect the set of input parameters; ii) run the MOGA-E2R2 heuristic; and iii) configure the set of rules in a proactive way. This process is triggered every time there is a significant variation in the traffic of the network. For this purpose, a set of warnings are defined to trigger its execution (e.g., a deviation of 5% in the load of a link).

[1]In general, if shorter paths are used, a lower number of links are involved, and hence less energy is consumed.

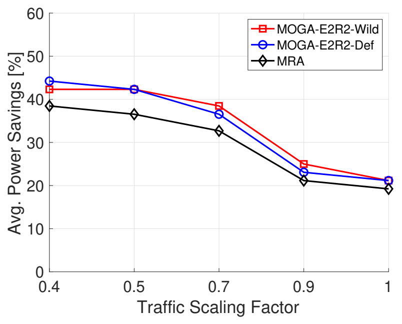


Figure 1. Average Power Saving for Nobel with $X = 2$.

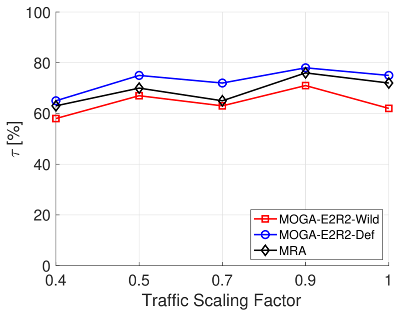


Figure 2. Maximum TCAM utilization for Nobel with $X = 2$.

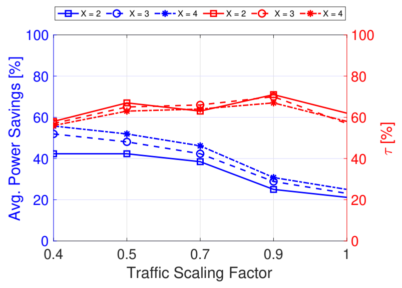


Figure 3. MOGA-E2R2 outcomes for different values of X.

B. *Complexity Analysis*

In order to analyze MOGA-E2R2 complexity, two variables must be considered: the size of the population, P, and the chromosome length, i.e., the number of links in the network, L. Apart from these two factors, the algorithm complexity mainly depends on the fitness functions to be optimized (eqs. 9-11), as well as on the involved operators (selection, crossover and mutation). In our case, the fitness functions take $\mathcal{O}(PL^2)$ for the evaluation of the population. Complexity associated to the roulette wheel criterion used to select the individuals that will survive and will become part of the next generation is $\mathcal{O}(P)$. Finally, crossover and mutation require $\mathcal{O}(PL)$, which means that the resulting complexity for MOGA-E2R2 is $\mathcal{O}(PL^2)$.

IV. Experimental Results

In this section, the possible benefits of applying MOGA-E2R2 are analyzed, as well as their potential drawbacks are discussed. At first, the simulation environment is presented. Then, two performance analyses have been carried out: i) a comparison between our solution and a benchmark algorithm, and ii) an analysis of the impact of applying the Rate Adaptation technique on the network power savings and on the TCAM utilization.

The network topology that has been considered is Nobel, available at [17], and composed of 17 nodes and 52 links. Regarding the traffic pattern, 5 TMs with different traffic load are considered. The selection of this set of TMs is described next: from the 288 TMs available at [17] for Nobel, the peak TM is taken as baseline. Then, it is scaled down by multiplying it by $\{0.4, 0.5, 0.7, 0.9\}$ to obtain the remaining set of TMs. In this way, the traffic scaling factor is varied on the X axis of the figures. In order to set the TCAM size limit, we apply next criterion. We first steer the traffic of the peak TM according to Dijkstra and obtain the maximum TCAM utilization in the network, τ_i. Then, we set the TCAM size of each node in the network as $C_n = \frac{\tau_i}{2}$. With respect to link constraints, MLU is set to $u_{i,j} = 0.8$. Regarding the links power consumption model, we adopt the energy functions and energy distributions reported in [14].

Concerning the criterion to handle the set of traffic demands, it is worth to say that in this work it is assumed that traffic can be known ahead of time, thanks to estimations that can be exploited, e.g., by using Machine Learning techniques. In particular, the set of requests is ordered in descending order to first steer the set of traffic flows that are bigger in terms of bandwidth. The evolutionary parameters for MOGA-E2R2 were empirically set to *Population* $= 20$, *Generations* $= 50$, roulette wheel criterion is used for selection, crossover is of type single-point with 10% of crossover rate, and uniform mutation with 5% of mutation rate is applied to create new offspring in the subsequent generation.

The first analysis we propose aims at comparing our proposal with a benchmark algorithm, namely Max Rules Aware (MRA), which is proposed in [7], [8]. Since the power consumption model considered in such work is of type sleeping (on-off), MOGA-E2R2 is run considering that links can be either powered off or working at their full rate, i.e., two operational modes ($X = 2$). Fig. 1 reports the average power savings obtained by our solution considering *default compression* and *wildcard compression*, which are compared to MRA. As expected, it can be noticed that the power savings decrease with the traffic load. Remarkably, our proposed algorithm outperforms MRA regardless the type of compression applied and the traffic load. In fact, the compression method exploiting wildcards is slightly better than the one only considering the default rule, while both of them achieve about 4% of power saving gains compared to MRA.

If we now move our attention to the maximum TCAM utilization, τ, we remark by inspecting Fig. 2 that MOGA-E2R2 presents better results than MRA for the case of wildcard compression. However, our solution suffers in the case of applying the default compression technique. Moreover, the traffic scaling factor does not impact on the value of τ, achieving an average around 70% of occupation in the most loaded node. This situation can be explained as follows: for a high traffic load (e.g., traffic scaling factor of 1 or peak TM) there are less chances to put certain links to sleep. Therefore, the set of flows are fairly distributed over the network and a potential low value of number of rules installed at each node can be expected. On the other hand, if the traffic is low (e.g., traffic scaling factor of 0.4), several links are put to sleep, which results in an increase in the number of installed rules.

Indeed, since this number increases, the compression method (default or wildcard) can be applied, therefore reducing the final number of installed rules. As a summary, the value of τ can be similar for peak and non-peak TMs, but due to different reasons: fairly distribution of flows in the former case, and application of compression methods in the latter.

Once the comparison between our proposal and MRA clearly shows that our solution outperforms MRA considering two operational modes for the line cards, Fig. 3 shows the impact of relaxing the number of operational modes that can be used (i.e., by applying the Rate Adaptation technique [14] with $X = 3$ and $X = 4$). In the left Y axis, the average power savings is represented as a function of the traffic scaling factor, while the maximum TCAM utilization is referred to the right Y axis. Clearly, the higher number of operational modes a line card can use, the higher power savings MOGA-E2R2 is able to achieve. The best outcomes are obtained when the traffic load is low, with a difference of 20% comparing the case of $X = 4$ with the sleeping one ($X = 2$). However, although such flexibility also impacts on the value of τ, the gap is not as high as in the case of the power savings achieved.

V. Conclusion and Future Works

In this paper, the power consumption and TCAM size limitation problems in SDN networks have been jointly studied. In particular, a multi-objective optimization problem for the joint minimization of power consumption and the number of installed rules at SDN switches is defined, while a MOGA is proposed to practically solve it. Simulations on a real topology show that MOGA-E2R2 outperforms other state-of-the-art solutions, both in terms of power saving gains (20% at non-peak TM) and maximum TCAM utilization (5%). As future steps, MOGA-E2R2 is planned to be installed in a real SDN controller to evaluate its impact on the network performance in a working test-bed.

VI. Acknowledgements

This work has been partially funded by the project RTI2018-094591-B-I00 (MCI/AEI/FEDER,UE), 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 and Infrastructure of the Government of Extremadura (GR18112, IB18030), and by the European Regional Development Fund.

References

[1] F. Idzikowski, L. Chiaraviglio, A. Cianfrani, J. L. Vizcaíno, M. Polverini, and Y. Ye, "A survey on energy-aware design and operation of core networks," *IEEE Communications Surveys Tutorials*, vol. 18, no. 2, pp. 1453–1499, Secondquarter 2016.

[2] J. Galán-Jiménez, M. Polverini, and A. Cianfrani, "Reducing the reconfiguration cost of flow tables in energy-efficient software-defined networks," *Computer Communications*, vol. 128, pp. 95 – 105, 2018.

[3] J. Galán-Jiménez, "Minimization of energy consumption in ip/sdn hybrid networks using genetic algorithms," in *2017 Sustainable Internet and ICT for Sustainability (SustainIT)*, 2017, pp. 1–5.

[4] C. R. Meiners, A. X. Liu, and E. Torng, "Bit weaving: A non-prefix approach to compressing packet classifiers in tcams," *IEEE/ACM Transactions on Networking*, vol. 20, no. 2, pp. 488–500, April 2012.

[5] P. C. Lekkas, *Network Processors: Architectures, Protocols, and Platforms*. New York, NY, USA: McGraw-Hill, 2003.

[6] M. Kuźniar, P. Perešíni, D. Kostić, and M. Canini, "Methodology, measurement and analysis of flow table update characteristics in hardware openflow switches," *Computer Networks*, vol. 136, pp. 22–36, 2018.

[7] F. Giroire, J. Moulierac, and T. K. Phan, "Optimizing rule placement in software-defined networks for energy-aware routing," in *2014 IEEE Global Communications Conference*, Dec 2014, pp. 2523–2529.

[8] F. Giroire, N. Huin, J. Moulierac, and T. K. Phan, "Energy-aware routing in software-defined network using compression," *The Computer Journal*, 2018.

[9] M. Polverini, J. Galán-Jiménez, F. G. Lavacca, A. Cianfrani, and V. Eramo, "Dynamic in-network classification for service function chaining ready sdn networks," in *2019 10th International Conference on Networks of the Future (NoF)*, 2019, pp. 74–81.

[10] H. Zhu, H. Fan, X. Luo, and Y. Jin, "Intelligent timeout master: Dynamic timeout for sdn-based data centers," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2015, pp. 734–737.

[11] E. Norige, A. X. Liu, and E. Torng, "A ternary unification framework for optimizing TCAM-based packet classification systems," *IEEE/ACM Transactions on Networking*, vol. 26, no. 2, pp. 657–670, April 2018.

[12] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with difane," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 351–362, 2010.

[13] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, and D. Wetherall, "Reducing network energy consumption via sleeping and rate-adaptation," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 323–336.

[14] J. Galán-Jiménez and A. Gazo-Cervero, "Using bio-inspired algorithms for energy levels assessment in energy efficient wired communication networks," *Journal of Network and Computer Applications*, vol. 37, pp. 171 – 185, 2014.

[15] L. Chiaraviglio, M. Mellia, and F. Neri, "Minimizing isp network energy cost: Formulation and solutions," *IEEE/ACM Transactions on Networking*, vol. 20, no. 2, pp. 463–476, April 2012.

[16] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang, "Multiobjective evolutionary algorithms: A survey of the state of the art," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 32 – 49, 2011.

[17] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly, "SNDlib 1.0–Survivable Network Design Library," in *Proceedings of the 3rd International Network Optimization Conference (INOC 2007), Spa, Belgium*, April 2007, http://sndlib.zib.de, extended version accepted in Networks, 2009.

Appendix K

A Privacy-Aware Architecture to Share Device-to-Device Contextual Information

ID	: **SMARTCOMP 2020**
Authors	: Juan Luis Herrera, Javier Berrocal, Juan Manuel Murillo, Hsiao-Yuan Chen, Christine Julien
Publication type	: Conference full-paper
Conference	: IEEE International Conference on Smart Computing (SMARTCOMP)
DOI	: `https://doi.org/10.1109/SMARTCOMP50058.2020.00044`
Conference rating	: International conference

A Privacy-Aware Architecture to Share Device-to-Device Contextual Information

Juan Luis Herrera*, Javier Berrocal*, Juan M. Murillo*, Hsiao-Yuan Chen† and Christine Julien†
*University of Extremadura, Spain. [jlherrerag, jberolm, juanmamu]@unex.es
†University of Texas, Austin, United States. [littlecircle0730, c.julien]@utexas.edu

***Abstract*—Smartphones have become the perfect companion devices. They have myriad sensors for gathering the context of their owners in order to adapt the behaviour of different applications to the device's situation. This information can also be of great help in enabling the development of social applications that, otherwise, would require a costly and intractable deployment of sensors. Mobile Crowd Sensing systems highly reduce this cost, but realizing this vision using traditional centralized networking primitives requires a constant stream of the sensed data to the cloud in order to store and process it, which in turn leads to the individuals about whom the data is sensed losing control over the privacy of the data. In this paper, we propose an architecture for a device-to-device Mobile Crowd Sensing system and we deepen on a new privacy model that allows users to define access control policies based on their context and the consumer's context.**

***Index Terms*—Privacy, Mobile Crowd Sensing, Device-to-Device, Pervasive computing, mobile applications.**

I. Introduction

Context-aware applications are increasing in their importance on mobile phones because of their adaptability and ease of use. These applications behave differently depending on the user's context, thus adapting themselves to provide services that can be most useful to their users. To do so, these applications feed on contextual information, which includes data such as the location or identity of the user, time-related data or the activity being performed [1]. Context-aware applications can even infer higher-level information from combinations of lower-level data in order to offer the most relevant information or services to a specific user.

These applications are nowhere more evident or important than in new smart city infrastructures [2]. In these environments, context-aware applications must also be social, that is, the contextual information from different users must be shared to enable collaboration in the smart city. One approach would be to deploy infrastructure-owned sensors to collect information from the inhabitants of the smart city. However, such an approach requires the deployment and maintenance of many sensors—an approach that is costly and intractable. Another possibility is to leverage the sensors that the city's inhabitants carry with them in any case—sensors connected to their wearable devices and smartphones. We refer to these devices throughout as *companion devices*. These devices are already imbued with myriad sensors, from accelerometers to GPS receivers. Existing applications use these personal devices to support individual user's needs, but it is also possible to offer them and the data they collect to be consumed by other nearby users and their devices [3]. For instance, the user's location, which is normally used by the smartphone to provide the user services about where they parked or information about the bar, shop, or restaurant where the user currently is, can also be shared so that other users can know if a bar is crowded or to let the city council to know which are busiest streets so that public policies can be better planned, for instance to make pedestrian routes more accessible for disabled people.

While this device-to-device cooperation is technically feasible, there remain unsolved challenges. First, there is a need for a technological architecture that allows companion devices to offer the information they sense to other devices, providing a gateway to access the surrounding context information. What's more, the architecture must provide mechanisms to allow different devices to communicate opportunistically. Second, these virtual context profiles potentially contain particularly sensitive and private information, such as the location or trajectory of the user or the activities they perform, that could be used for malicious purposes. Therefore, it is not enough to have an architecture that allows devices to offer their contextual information to others, but the architecture should also ensure the privacy of the user in a way that is tunable by the user directly.

An initial approach could be to constantly upload the information sensed by user's devices to a cloud service that is able to create virtual profiles and distribute context information, but this would imply a constant data flow, would require a constant Internet connection from every device, and would result in an increased response time due to the latency added by the communication with the cloud [4]. In addition, such an approach requires all of the communicating parties to place their trust in a shared third party service provider. We explore an alternative that mitigates these concerns by storing the virtual context profiles in the users' companion devices, and providing services for allowing other devices to consume that information by means of different sharing mechanisms (device-to-device, opportunistic networks, etc.). An additional access control layer is required to empower users to explicitly control what and how other devices and users are authorized to access the data. In the end, a full system will likely entail a hybrid of both approaches, one that mixes cloud interactions with opportunistic ones. In this paper, however, we isolate the latter and focus only on enabling privacy-preserving device-to-device context sharing.

In this paper, we present PADEC (Privacy-Aware DEvice

Communication), an architecture that allows companion devices to create and provide context information so that other devices can query that data to enable their applications to adapt their behaviour to their users' needs and surroundings. This architecture includes elements to store this data in mobile devices, offer this information to neighboring devices, communicate with other devices through different communications mechanisms, and control the access to this data by using user-set policies through a novel system based on contextual and dynamic keys, keyholes and locks.

The remainder of this paper is structured as follows: Section II presents the motivation behind the PADEC architecture. Section III shows the high-level architecture of PADEC. Section IV provides the details of an example communication abstraction, while Section V describes our proposed access control privacy layer in detail. Finally, Section VII concludes the paper.

II. Motivation and Background

Mobile Crowd Sensing (MCS) refers to the reliance on personal device resources to support gathering information about humans and their surroundings [5]. This information is commonly used to create a collective intelligence to provide services to the users based on the collected information.

In the following subsection, we present a running example of an MCS and our motivations behind a privacy-aware framework to provide personalized MCS services.

A. Running example

Imagine a smart city that provides an application to be used by both residents and tourists. The app provides information about restaurants and bars by collecting and sharing crowd-sensed information. The app collects and stores users' context information, such as locations, activities (e.g., eating, walking, working), presence of others (e.g., friends and family), etc. The app can use this personal information to provide recommendations, nudges, or other services to the user based on the user's own contextual history [6].

Other users could benefit from the availability of this crowd-sensed contextual information. For instance, a tourist visiting the city could use the app to query nearby residents for a restaurant recommendation. Also, the app can be used to query other users' devices (and their stored context information) to find what is popular nearby, potentially with a day-specific special or happy hour. Additional contextual information (e.g., the tourist's food preferences, the size of the tourist's dinner party, or where the tourist ate lunch) could also be added to the query to improve the results. Residents can also benefit from being able to query the contextual information sensed by other residents (or even visitors). For instance, individuals could query for information about where their friends or family members might prefer to get drinks on a Friday evening so they can make a recommendation for a group outing.

B. Motivation

The presented running example requires a large amount of contextual information from many different devices. To get and process this information, one approach can be to offload all of the context information to the cloud, an approach akin to that of Google Cloud's Places [7]. For instance, MCS platforms such as PartcipAct [8] and Vita [9] gather the users' contextual information using their smartphones and store it on a server so that complex algorithms can be executed. However, this style of approach requires users' devices to constantly stream information to a cloud server and enables the third party owner of the cloud server to know and track the situation of each individual.

When the crowd-sensed data is offloaded, the owner of the data loses control over it. The third party becomes responsible for protecting and sharing the data. In these situations, users tend to eschew sharing their data with others. However, research has shown that, while users might not be willing to share their spatiotemporal data publicly, they are more prone to share the information with others who are physically nearby [10], [11].

An alternative approach is to *on-load* the contextual information to the user's device, keeping all stored mobile crowd-sensed information persistent only on the device belonging to the user. [6], [12], [13]. Once the information is on-loaded to a user's companion device, it can be shared opportunistically with other users nearby. This approach comes with its own set of challenges. First, because companion devices like smartphones may be resource constrained in terms of battery, computation, and storage, the device needs an intelligent mechanism to constantly sense contextual information and maintain an accurate yet lightweight representation of that context. Second, each companion device needs to offer some kind of interface to other devices so they can consume the stored information on demand. Third, both devices need a common communication medium.Finally, the shared information is still sensitive and private, and the users should retain control over what other nearby devices have access to.

The first challenge has been addressed by Paco [6], an on-loading context storage mechanism that uses a companion device's location sensor and other on-board context sensors to create a spatiotemporal context database stored entirely on the user's companion device. Paco leverages the spatiotemporal tags associated with contextual information to determine how novel a sensed data item is before storing it; spatiotemporally reducing the memory footprint of the sensed context. Paco exposes a query interface that allows applications to access the contextual data at varying levels of abstraction, opening the possibility of defining access rules for snapshots of a user's contextual data.

For the second challenge, i.e., offering interfaces to make the contextual data available to other devices, APIGEND [14] allows for easy code generation and deployment of Application Programming Interfaces (APIs) in companion devices. These APIs are composed of a series of endpoints that can be exposed and called from other devices. These endpoints wrap some business logic, which, for this work, will be the ability to access some context data and return it to the device that called the endpoint.

In this paper, we tackle the remaining two challenges: unifying the communication medium and providing context-sensitive approaches to controlling access to the stored contextual data. For defining the communication mechanism, there are a variety of protocols and platforms that can be use for device-to-device communication. Classic request-response protocols such as HTTP are not ideal in this situation due to the highly dynamic an unpredictable nature of the opportunistic device-to-device environment. Simply put, request-response protocols are host-based, and thus, one must know the address (normally the IP address) of a device to perform a request on it. This is not so simple when the device is mobile, since these devices often have private IPs that are unreachable from the outside.

There are a variety of opportunistic or publish-subscribe protocols that can be used to solve these problems. In this paper we focus on publish-subscribe protocols, but different techniques can be used. This communication model is promising and privacy-preserving, since it is possible to use a secure but distributed intermediate broker to mediate requests and responses [15]. To be clear, this is different than a fully centralized approach because the collaborating parties are interacting directly with one another and can secure their end-to-end communications. In the centralized approaches, information is released in the clear to the third party to allow the third party to perform data processing and analysis.

Within publish-subscribe communication models, Google's Firebase [16] is one popular approach. Another possible approach is to use the open MQTT protocol [17], but it requires setting up a dedicated broker rather than relying on the central broker provided in Firebase. On the other hand, this approach can reduce the reliance on the third party to mediate communication. In this work, we explore allowing applications to optionally use either of these publish-subscribe approaches.

The meat of this paper focuses on the final challenge: addressing and protecting the privacy of users' shared contextual information. The contextual information that must be exposed to enable the application uses described above is potentially highly sensitive. This information can be used by malicious parties, for instance, to spy on the user [18]. This necessitates a privacy layer that provides user-tunable access control for the contextual information. A wide variety of access control models exist, one of the most interesting being the NIST-standard Role-Based Access Control (RBAC) [19]. Other works have extended RBAC to consider context, such as Team-Based Access Control (TMAC) [20] or Dynamic Sharing and Privacy-Aware RBAC (DySP-RBAC) [21]. However, these models were designed for collaborative working environments. Smart cities with decentralized device-to-device communications demand new techniques that allow access control to be dynamic and to consider the identity (and context) of both the provider and the consumer of information. In this work, we present a new access control model designed for these environments: Dynamic Context and Identity-Based Access Control (DCIBAC).

III. Architecture Overview

In this section, we provide a high-level overview of the complete PADEC architecture, before drilling into the details of each of its constituent components. Figure 1 provides a pictorial representation.

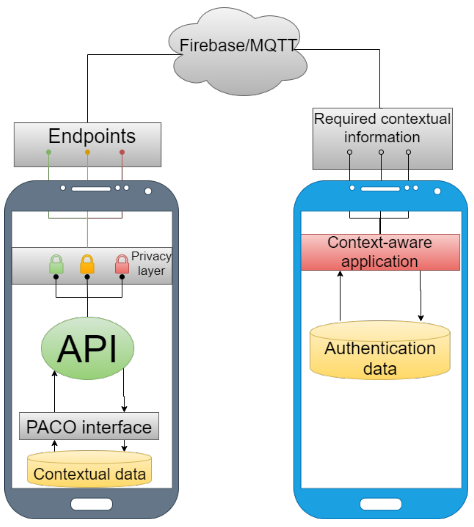


Figure 1. PADEC high-level architecture

In PADEC, a user's companion device can collect, store, and share contextual data. In addition, a context-aware application executing on a companion device can request contextual data from other nearby devices. In practice, devices will commonly perform both tasks. For simplicity, Figure 1 distinguishes these two roles; the device on the left is acting as a context collector and provider, while the device on the right consumes that context information via a remote connection. In PADEC, a device can only consume information if it also provides endpoints; this "sharing economy" style is PADEC's approach to incentivizing device participation. To describe the PADEC architecture, we walk through each element in the figure, starting at the bottom left of the figure and working up, to the right, and then back down.

The first layer focuses on storing the contextual information gathered by both internal and external sensors. In PADEC, we implement this layer using the Paco contextual data storage framework [6]. In Paco the gathered information is indexed within a database based a spatiotemporal context stamp that includes the coordinates and time at which the context information was collected. This spatiotemporal information is associated with the relevant semantic information (e.g., reviews, ratings, etc.) or data sensed by others sensors (e.g., temperature, noise, etc.). The key contribution of Paco is reducing the size and complexity of the stored contextual information so that it can be stored and queried entirely on the companion device. For this purpose, Paco also provides an API for issuing spatiotemporal queries for the stored information.

The second layer allows other devices to consume the information stored by Paco (API in Figure 1). This layer allows the deployment of OpenAPI-based APIs, to ensure that the endpoints exposed to applications are properly defined and documented so that remote applications can easily access them. These endpoints serve as the gateway between external devices and the local Paco data store. They serialize all of the exchanged information and they provide a first line of defense protecting the private spatiotemporal information stored within the Paco data store. To reduce the effort required to implement these APIs, we rely on the APIGEND tool [14]. This tool takes as input the API specification and generates the skeleton of an API for Android mobile devices and microcontrollers, so that developers only have to implement the business logic to invoke the query methods exposed in Paco's interface. For the time being, we simply expose the generic Paco interface, which allows queries asking the *probability of knowledge* of the Paco data store about a region, to retrieve the context data items contributing to that knowledge, or to trace a trajectory of the data owner through space and time. In the future, we could use APIGEND to expose more application-specific endpoints to further simplify the use of PADEC. For instance the endpoints used for a smart traffic application might focus queries on the density of cars on roadways, while endpoints for a smart tourism application might expose thematic walking tours.

Without additional protections, the deployed endpoints can be consumed by any PADEC-enabled connected device, which can be a problem for the privacy of the stored data. The next layer in the device on the left of Figure 1 implements a dynamic and multi-dimensional access control system for every endpoint (labeled "Privacy layer" in the figure). This layer enables the definition of access control policies that restricts a remote party's access to the context information. Access can be allowed at various levels of abstraction based on whether the requester and the provider meet concrete user-defined rules. These rules can define different contextual situations that have to be satisfied for the information to be shared. For instance, for our smart city example, a tourist may only be able to access stored information about the best restaurants if he is currently in the city. Furthermore, the level of detail of the information provided may also be different depending on different contextual situations. In our example, a tourist might be able to uncover a part of town that is likely to have many good restaurants rather than discovering exactly which restaurants the context provider has frequented.

The very top of Figure 1 shows the layer that manages the communication between the devices. In the current implementation of PADEC, we support two different publish-subscribe communication protocols (Firebase and MQTT) that can be employed individually or in combination. As future work, we plan to support other technologies such as direct device-to-device communication or opportunistic networks. By providing multiple options under the same umbrella, PADEC makes it possible for different devices in different situations to consume the provided information in the way most suited for the situation. The required infrastructure for implementing these protocols is also generated using the APIGEND tool.

In the following sections we will focus on the addressed challenges: how the communication medium is unified and the contextual access control policies are defined.

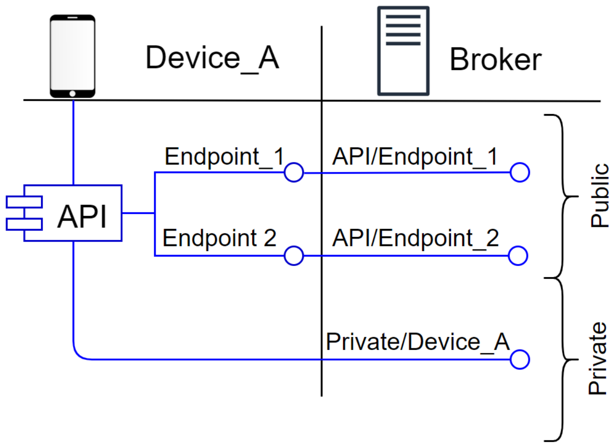


Figure 2. Communication Channel structure.

IV. Communication model

The communication layer at the top of Figure 1 mediates information sharing among devices to allow the exposed APIs to be consumed by other devices. The main responsibilities of the communication layer are to support different communication protocols for invoking the endpoints and to provide a first line of defense protecting the exchanged data.

In the current implementation of PADEC, the communication layer unifies diverse publish/subscribe protocols. Concretely, PADEC currently supports MQTT and Firebase Cloud Messaging. Communication through such protocols is commonly based on a central element, i.e., a "broker", that manages the entire communication. Devices can interact with the broker by establishing *topics* that create communication channels. With entities that *publish* information to a given channel, and *subscribers* receiving that information.

These roles are used in PADEC to request or send information by publishing in a channel and to receive the published information by being subscribed to that channel. Different channels are used to simulate the request-response communication pattern. To organize the communication among the myriad different entities and to enable strict access control, PADEC defines a topic structure that is automatically initialized when a device connects to the broker or when the device creates a new API endpoint or application component. This structure is shown in Figure 2.

PADEC creates two types of topics: private and public. The first time a device registers with the broker, a private channel is created, dedicated to communication destined to that device. Any device can publish in this channel, but only the registered device (the "owner" of the topic) is subscribed to it and is the only one that is notified of any published information. This private channel can be useful for multiple purposes. For instance, an endpoint provider can use its private channel to receive the private messages required to exchange the information used to determine access control of a requesting device. Likewise, a device invoking an endpoint can use its private channel to get the results of the invoked endpoint. More generally, when any device knows the ID of another device and wants to establish a device-to-device communication, it can use this private channel for the communication exchange. For devices not knowing the private channels, public channels for each provided endpoint are also created.

PADEC automatically creates public channels when a new API is deployed; the system creates a new topic for each provided endpoint. These channels are open, and any device can publish a message to them. In general, a message in one of these channels indicates a desire to invoke the particular endpoint. Only the device providing the endpoint is able to subscribe to and consume messages published to the topic. This provides one level of privacy for the devices invoking the endpoints; only devices hosting the target API will be able to see the request, preventing other devices from reading the content of the device's endpoint invocation. At the same time, this architecture provides a form of *spatial decoupling* in which the device wanting to invoke the endpoint does not need to know the details of the hosting device, like its address or location. These channels only support the invocation of endpoints, any subsequent communication for exchanging the intermediate messages or the requested data is maintained through private topics with the aim of not leaking private data. In addition to the decoupling that this provides, it also enables PADEC to create more sophisticated endpoints out of the aggregation of base endpoints to provide higher level services. For example, to ascertain whether a specific bar or area of the city is crowded, a requesting device can invoke a public endpoint offering information about the target location. Any device within this area willing to share crowd information can provide the endpoint and be subscribed to the channel. Upon receiving a notification of the published invocation, all of the devices providing the endpoint will reply, and the requesting device can aggregate the information to acquire a more robust measure of crowd density.

In PADEC, we implement this same functionality with both MQTT and Firebase Cloud Messaging. The former can be deployed in networks without any external connectivity to the Internet, while the latter relies on access to the Firebase cloud servers. Currently, the broker deployment is out of the scope of the paper. Nevertheless, for MQTT a hierarchical model can be used in order to improve the scalability. In the next section, we describe the PADEC privacy model defined to dynamically control the access to the information.

V. Privacy model

For PADEC's privacy layer, we propose a new access control model named Dynamic Context and Identity-Based Access Control (DCIBAC) that is designed with decentralized smart environments in mind, while inheriting some elements from access control models such as RBAC [19] or DySP-RBAC [21]. As its name implies, DCIBAC uses information about the identities and context of coordinating parties to determine whether access is granted. In PADEC, this means that information is used about both the endpoint provider and the requester to determine whether the invocation is allowed and how the invocation result can be presented, e.g., in terms of abstraction or granularity of the data returned. In contrast to approaches for access control of data shared in a cloud-based system, because PADEC performs access control locally, it can compare a requester's contextual information to the data owner's very private contextual information without requiring the provider to release the personal data to the third party that completes the access control decision.

In DCIBAC, pieces of information that need to be protected are called *objects*, and the agents that either own or try to access objects are *entities*. DCIBAC is based on putting *locks* over objects, so only authorized entities can access them. The entities who try to access objects are named *requestors*. The authority controlling if some specific information can be provided to the requestors is distributed and delegated to the providers. In PADEC, the data is the contextual data kept in a Paco data store, and the entities are the devices requesting remote access to information stored in Paco.

DCIBAC's *locks* are implemented via user-defined rules that can impose conditions, mainly on the context or identity of the owner of the object or on those of the requester. For instance, a user may allow access to information about the specific restaurant they are in at the moment to be accessed only by their near family. A user may allow their boss to access their position but only during working hours. These conditions can be combined using logical *or*, *and* and *not*, thus allowing for complex and rich conditions. For instance, it is possible to allow friends to access information about the restaurant the user is in only during Friday nights and there is not a scheduled appointment in the user's calendar for Friday night. DCIBAC also allows locks to include conditions on the intended use of the information, for instance the particular application consuming the information. A user's instantaneous location data may not normally be released to a stranger's device, but it might be released to smart traffic app if there is a nearby traffic accident.

To enable devices to request access to a protected object, DCIBAC uses, as far as we know, a novel concept called *keyhole*. Before requesting access to the object, the requester must first get the keyhole of the object. This keyhole identifies the information that the requester must send to open the lock request access. For instance, if a lock allows access to friends as long as they are nearby and their purpose is not marketing, its keyhole would be the position of the requester, their identity, and their purpose. This interaction does not reveal the exact conditions of the lock, only the type of information on which the decision is based. This message sent from the requester with the needed information is termed a *key*.

Figure 3 summarizes the workflow that has to be followed in order to validate an endpoint's defined access control policies.

1) The requesting device invokes an endpoint publishing a message in the associated topic.
2) The provider checks whether the endpoint is available in

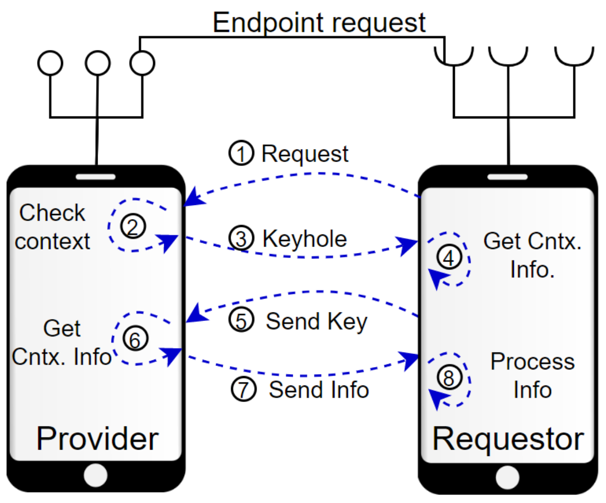


Figure 3. Communication workflow.

the provider's current context. For instance, a user may disallow access to an endpoint that shares their recently taken photographs while on an outing with their family.

3) If the endpoint can be consumed, the provider sends to the requester the keyhole that specifies any needed contextual information.
4) The requesting device acquires the contextual information required by the keyhole using the device's own local data store, on board sensors, or by invoking a remote third party service.
5) The requester packages the obtained information as a key that can be used to attempt to open the lock protecting the endpoint.
6) The endpoint provider executes the lock to check that the provided information is sufficient to grant the requester access to the protected endpoint. The provider invokes the endpoint if the lock is successfully opened.
7) After invoking the endpoint, the provider publishes a notification on the requester's private channel with the information requested, if access is allowed, or with a rejection notice if not.
8) Finally, the requesting device processes the obtained information.

Only the first message is published on a public channel, specifically the public channel associated with the target endpoint. The subsequent messages use the devices' private channels. This process allows us to create and support a dynamic and multidimensional privacy model able to provide or reject the consumption of information depending on the context of both the provider and the requester.

The final concept in DCIBAC is the definition of *access levels* of a lock. In Paco, each request to retrieve information from the Paco data store is made with a certain profile. This profile determines the degree of spatiotemporal granularity with which the request is allowed to view the contextual data store and the maximum number of requests an individual requester is allowed to make. These restrictions further protect the privacy of the owner's spatiotemporal information.

These profiles are mapped to access levels in DCIBAC locks. A single lock has by default a single access level, but it can also have many levels. These access levels are ordered, with the first being the level that releases data from the Paco data store with the least precision. Each level can have its own conditions, meaning a lock with multiple levels may be associated with each lock. Each level has a keyhole associated.

Since each access level can use different contextual information, it is possible that a single lock is associated with multiple keyholes. When a lock exposes multiple keyholes, it exposes some information about how its access decisions are structured. While this can aid requesters in understanding the conditions of access it also potentially exposes decision logic of the endpoint owner. Navigating this trade-off is left flexible for the endpoint owners to navigate.

When a user invokes an endpoint, all the keyholes associated with the endpoint's lock are sent to the requester together with the precision of the information they will get. The requester can analyze this precision in order to create the key for the keyhole with the required precision. This also allows the requester to leak, with the key, the minimal contextual information to get the needed information.

As an example, consider a lock with two access levels: level 1 that provides abstract, granular information for unknown nearby devices and level 2 that provides more detailed information for close family. Two different keyholes would be created, one per access level. A keyhole for level 1 (position) and another one for level 2 (position and identity). While this makes sure the requester knows the information required to open each level of the lock (meaning the requester is able to protect its own private contextual information more), the requester can also discern some of the logic relating to the lock's access control policies.

Figure 4 shows this example. In this example, DCIBAC secures a single endpoint that gives access to the restaurants visited by the user. To control access to this endpoint, the user uses a lock with two access levels. Nearby strangers can request access information about the restaurants that the endpoint owner likes (level 1) and only nearby close family members can access information about how often the endpoint owner frequents particular restaurants. Two keyholes are created,such that the keyhole for level 1 of the lock requires the requester to provide position information, while the keyhole for level two requires both position and identity. In the example process depicted in Figure 4, the device of a tourist at the same bus stop as the endpoint owner will try to access good places to have a drink. To do so, the tourist's device first publishes the endpoint request to the endpoint's public topic (*1*) and receives a message generated by DCIBAC containing the keyholes of the lock (*2*). When this response is received, the tourist's phone will decide which level it needs to get access to, will collect its current position using its GPS sensor and place this information in a message that will serve as the tourist's key (*3*). The key is then be sent to the secure endpoint (*4*) via the

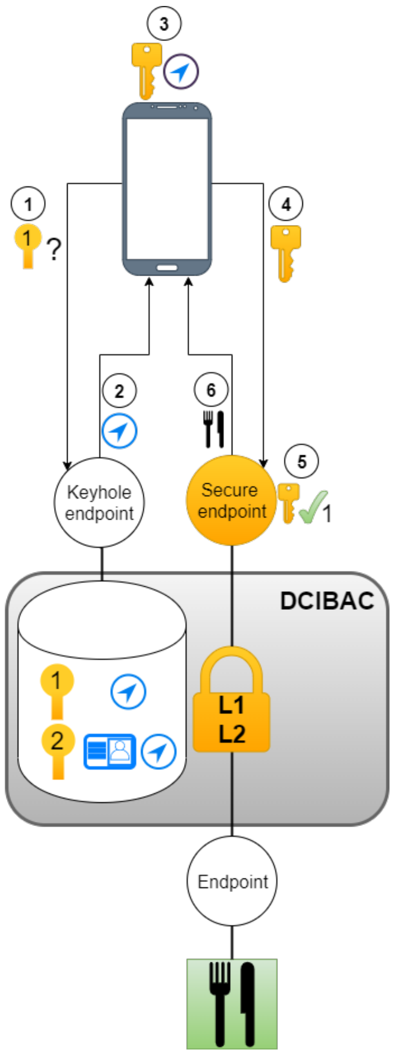


Figure 4. DCIBAC example usage.

endpoint's private topic. Once received, DCIBAC will check if the key is valid according to the lock and access level. Since the tourist is near the user and access level 1 was requested, access is granted (*5*). The secured endpoint will then invoke the necessary request on the endpoint's Paco data store and then send the information about the favourite restaurants of the user to the tourist's phone (*6*).

The permission to access an object in DCIBAC is dynamically checked and granted during execution time. This means that every time a requester wants to access some information, it must send its key, and there is no guarantee that a key will stay valid for any period of time, since the number of queries might be restricted depending on the access level.

VI. Discussion

Context-aware and MCS applications have proven useful during the last few years to distribute sensing and computation tasks in order to develop social and collaborative environments at an affordable way. For instance, [22] presents a framework to opportunistically offload computation in smart vehicles by accounting for context information. This framework uses the context of nearby vehicles, such as their speed or direction, to determine the best candidate to execute some tasks. [23] presents another framework that offloads data from social networking services to mobile devices based on their social context to maximize the quality of service of those services. These are also commercial examples of social applications that need contextual information from nearby users, such as Facebook Safety-Check [24].

A common approach to develop these platforms is to offload the data, or most of the data, to the cloud. This entails added resource consumption and a significant potential loss of privacy when the user's private contextual data resides in the Internet. These approaches also require a constant data flow, which can be problematic when the user has a limited mobile data plan. Such an approach also requires the user's device to be permanently connected to the Internet, which may not be possible in rural areas or even in urban canyons. This approach also relies on the third party infrastructure provider for protection and storage of data, and this third party provider is able to know and track every single device using the application. These challenges motivate PADEC's approach to onloading computing and data storage, which in turn enables device-to-device opportunistic provision of data access endpoints.

However, even when the data is stored locally, the user's privacy can be at risk, since other devices could potentially access that data if endpoints are not protected. To protect this information from malicious parties, storing and sharing contextual information is not enough, and an architecture that also accounts for privacy is required.

PADEC addresses these concerns head-on by providing an architecture that empowers users to store sensed contextual information in their own devices and to define dynamic and multidimensional access control policies that allow them to indicate what information can be accessed, by whom, and under what contextual circumstances.

This architecture also resolves other associated challenges, such as making it easier for developers to build mobile crowd sensing systems and applications that are especially complex due to device-to-device communication. PADEC's mechanisms for resolving these ancillary challenges comprise a series of modules that abstract developers from the implementation details on how to store information efficiently, how to expose the stored information to other users, and how to implement the communication workflow.

An open remaining challenge is to create methods to help and guide users in the definition of privacy policies, making the keyholes and locks that are fundamental to PADEC easier to use and more robust. Ideally, application users will be guided in establishing these rules in a clear, simple, and accurate way. Further, users should also be given a clear image of what information they are exposing or not depending on the established rules.

VII. Conclusions

In this paper, we presented PADEC to simplify the consumption of contextual information from other devices for context-aware and Mobile Crowd Sensing systems, which are important not only for the development of social applications, but also in paradigms such as the Internet of Things, Web of Things, or Human-in-the-Loop applications that require some devices to collaborate and adapt their behaviour depending on the context and the users' needs.

PADEC on-loads spatiotemporally tagged contextual information to the owner's device, and then enables this stored data to be queried by other devices. PADEC relies on the Paco framework for onloading the contextual data and APIGEND to define flexible API endpoints that expose Paco interfaces for remote queries. In this paper, we focused on providing user-tunable protection of the exposed endpoints by defining DCIBAC, a flexible access control mechanism that relies on the users' identities and context to determine whether access is allowed and what level of granularity of spatiotemporal information should be exposed. DCIBAC allows the definition of access control policies that protect the privacy of both coordinating parties.

Currently, we continue to work on evaluating the effectiveness of the PADEC architecture with respect to both the consumption of resources needed for the companion devices to on-load the computation and the communication load that the defined medium presents. On the other hand, we are also working on how to provide accurate information to users about the information they are exposing and the risks that this entails for their privacy. Finally, as future work, we will also define incentives and policies in order to foster information sharing.

Acknowledgment

This work was supported by the project RTI2018-094591-B-I00 (MCI/AEI/FEDER, UE), by 4IE project (0045-4IE-4-P) 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.

References

[1] R. Löwe, P. Mandl, and M. Weber, in *2012 IEEE International Conference on Pervasive Computing and Communications Workshops, PERCOM Workshops 2012*, 2012, pp. 76–81.

[2] R. Kamberov, V. Santos, and C. Granell, "Toward social paradigms for mobile context-Aware computing in smart cities: Position paper," in *Iberian Conference on Information Systems and Technologies, CISTI*, vol. 2016-July. IEEE Computer Society, jul 2016.

[3] J. Guillen, J. Miranda, J. Berrocal, J. Garcia-Alonso, J. M. Murillo, and C. Canal, "People as a service: A mobile-centric model for providing collective sociological profiles," *IEEE Software*, vol. 31, no. 2, pp. 48–53, 2014.

[4] J. Berrocal, J. García-Alonso, C. Vicente-Chicote, J. H. Núñez, T. Mikkonen, C. Canal, and J. M. Murillo, "Early analysis of resource consumption patterns in mobile applications," *Pervasive Mob. Comput.*, vol. 35, pp. 32–50, 2017. [Online]. Available: https://doi.org/10.1016/j.pmcj.2016.06.011

[5] H. Vahdat-Nejad, E. Asani, Z. Mahmoodian, and M. H. Mohseni, "Context-aware computing for mobile crowd sensing: A survey," *Future Generation Computer Systems*, vol. 99, pp. 321 – 332, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X18329583

[6] N. Wendt and C. Julien, "Paco: A System-Level Abstraction for On-Loading Contextual Data to Mobile Devices," *IEEE Transactions on Mobile Computing*, vol. 17, no. 9, pp. 2127–2140, sep 2018.

[7] Google-Cloud, "Places — Google Maps Platform — Google Cloud," 2019. [Online]. Available: https://cloud.google.com/maps-platform/places/

[8] G. Cardone, A. Corradi, L. Foschini, and R. Ianniello, "Participact: A large-scale crowdsensing platform," *IEEE Transactions on Emerging Topics in Computing*, vol. 4, no. 1, pp. 21–32, 2015.

[9] X. Hu, T. H. Chu, H. C. Chan, and V. C. Leung, "Vita: A crowdsensing-oriented mobile cyber-physical system," *IEEE Transactions on Emerging Topics in Computing*, vol. 1, no. 1, pp. 148–165, 2013.

[10] Q. Jones, S. A. Grandhi, S. Karam, S. Whittaker, C. Zhou, and L. Terveen, "Geographic place and community information preferences," *Computer Supported Cooperative Work (CSCW)*, vol. 17, no. 2-3, pp. 137–167, 2008.

[11] E. de Matos, R. T. Tiburski, C. R. Moratelli, S. Johann Filho, L. A. Amaral, G. Ramachandran, B. Krishnamachari, and F. Hessel, "Context information sharing for the internet of things: A survey," *Computer Networks*, vol. 166, p. 106988, 2020.

[12] S. Han and M. Philipose, "The case for onloading continuous high-datarate perception to the phone," in *Presented as part of the 14th Workshop on Hot Topics in Operating Systems*, 2013.

[13] N. Vallina-Rodriguez, V. Erramilli, Y. Grunenberger, L. Gyarmati, N. Laoutaris, R. Stanojevic, and K. Papagiannaki, "When david helps goliath: the case for 3g onloading," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, 2012, pp. 85–90.

[14] S. Laso, M. Linaje, J. Garcia-alonso, J. M. Murillo, and J. Berrocal, "Deployment of APIs on Android Mobile Devices and Microcontrollers," in *2020 IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2020*, 2020, pp. 12–14.

[15] S. Arnautov, A. Brito, P. Felber, C. Fetzer, F. Gregor, R. Krahn, W. Ozga, A. Martin, V. Schiavoni, F. Silva *et al.*, "Pubsub-sgx: Exploiting trusted execution environments for privacy-preserving publish/subscribe systems," in *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2018, pp. 123–132.

[16] Google, "Firebase API Reference." [Online]. Available: https://firebase.google.com/docs/reference?hl=es

[17] MQTT, "MQTT - Message Queue Telemetry Transport," Apr. 2020. [Online]. Available: http://mqtt.org/

[18] IEEE ComSoc, "Newsweek: We're Surrounded by Billions of Internet-connected (IoT) Devices. Can We Trust Them? – Technology Blog," 2019. [Online]. Available: https://techblog.comsoc.org/2019/10/26/newsweek-were-surrounded-by-billions-of-internet-connected-iot-devices-can-we-trust-them/

[19] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Computer role-based access control models," vol. 29, no. 2, pp. 38–47, feb 1996.

[20] C. K. Georgiadis, I. Mavridis, G. Pangalos, and R. K. Thomas, "Flexible team-based access control using contexts," in *Proceedings of Sixth ACM Symposium on Access Control Models and Technologies (SACMAT 2001)*. New York, New York, USA: ACM Press, 2001, pp. 21–27. [Online]. Available: http://portal.acm.org/citation.cfm?doid=373256.373259

[21] A. K. Malik and S. Dustdar, "Enhanced sharing and privacy in distributed information sharing environments," in *Proceedings of the 2011 7th International Conference on Information Assurance and Security, IAS 2011*, 2011, pp. 286–291.

[22] A. U. Rahman, A. W. Malik, V. Sati, A. Chopra, and S. D. Ravana, "Context-aware opportunistic computing in vehicle-to-vehicle networks," *Vehicular Communications*, vol. 24, p. 100236, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2214209620300073

[23] H. R. Cheon and J. H. Kim, "Social Context-Aware Mobile Data Offloading Algorithm via Small Cell Backhaul Networks," *IEEE Access*, vol. 7, pp. 39 030–39 040, 2019.

[24] "Facebook safety check." [Online]. Available: https://www.facebook.com/about/safetycheck/

Appendix L

Predicting Response Time in SDN-Fog Environments for IIoT Applications

ID	: **LATINCOM 2021**
Authors	: Guilherme Werneck de Oliveira, Rodrigo Toscano Ney, Juan Luis Herrera, Daniel Macêdo Batista, Roberto Hirata, Jaime Galán-Jiménez, Javier Berrocal, Juan Manuel Murillo, Aldri Luiz dos Santos, Michele Nogueira
Publication type	: Conference full-paper
Conference	: IEEE Latin-American Conference on Communications (LATINCOM)
DOI	: `https://doi.org/10.1109/LATINCOM53176.2021.9647803`
Conference rating	: International conference

Predicting Response Time in SDN-Fog Environments for IIoT Applications

Guilherme Werneck de Oliveira*, Rodrigo Toscano Ney*, Juan Luis Herrera†,
Daniel Macêdo Batista*, R. Hirata Jr.*, Jaime Galán-Jiménez†, Javier Berrocal†, Juan Manuel Murillo†,
Aldri Luiz dos Santos‡, Michele Nogueira‡
*Department of Computer Science
University of São Paulo (USP), Brazil
{werneck, rodrigo.ney, batista, hirata}@ime.usp.br
†Department of Computer Science and Communications Engineering
University of Extremadura (UEx), Spain
{jlherrerag, jaime, jberolm, juanmamu}@unex.es
‡Department of Computer Science
Federal University of Minas Gerais (UFMG), Brazil
{aldri, michele}@dcc.ufmg.br

***Abstract*—In IoT application scenarios, the response time is one of the attributes that most require attention and, for this reason, the paradigm of decentralized (or fog) computation has gained ground. Moreover, to help reduce the response time of decentralized IoT networks, routing optimization approaches can be employed using software-defined networking (SDN). When both contexts are combined, a new one called SDN-Fog Environments appears. This work presents a solution to predict the response time of Industrial Internet of Things (IIoT) applications using supervised and unsupervised learning for SDN-Fog Environments. Results show that the prediction of the response time of IIoT scenarios was close to the times obtained by solving the problem in the literature. Furthermore, according to the best-performing models, the prediction framework had less than 50 milliseconds of variation, executed in less than one second.**

***Index Terms*—prediction, response time, IoT, Fog, SDN, machine learning**

I. Introduction

The Internet of Things (IoT) has achieved rapid popularization, reaching a wide range of applications in healthcare, environmental monitoring, home automation, smart mobility, and Industry 4.0 [1], [2]. Thus, more and more IoT devices are deployed in various public and private environments, progressively becoming common objects of everyday life. The IoT has been seen by many as the next stage of the Internet, and its implementation enables the evolution not only of industrial infrastructures but also of smart cities because, with IoT, the development of transport systems, waste control, energy, among others, becomes more efficient and improves the quality of life of citizens. On the other hand, the physical infrastructure of heterogeneous systems is complex and requires efficient and dynamic solutions for deploying, configuring, and managing IoT networks [3], [4].

In whatever IoT application scenario, the response time for a given activity is one of the attributes that most require attention [5] and, for this reason, the paradigm of decentralized computation processing has gained ground in this context. *Fog computing* is an example that supports applications that require strict-response time, separating the processing of a given task into nodes of distinct processing capacity interconnected by network and closer to end-devices [6]. The latency is another factor to be seen in the composition of the response time and, to make an optimal decision about the location of processing devices, the decentralized computation distribution problem (DCDP) [7] should be considered. To reduce the latency of fog computing, routing optimization approaches can be employed [8] by using software-defined networking (SDN) [9]. In other words, SDN controller placement strategies are used to minimize the latency among devices. However, the latency between SDN controllers and switches must also be considered when computing network response time. The problem of placing SDN controllers on top of switches is known as Controller Placement Problem (CPP) [10].

The combination of both problems, DCDP and CPP, results in another one, the SDN-Fog deployment problem, posed initially by Herrera *et al.* [11] to describe a single effort approach to solve them together. The proposed approach consists of a Distributed Application Deployment Optimization (DADO) framework, which contains Mixed Integer Linear Program (MILP) into its core for identifying the deployment of both services and SDN controller in order to meet specific requirements. The results presented by the authors showed that the framework is suitable for small network topologies, infrastructures with under 300 nodes, in which the optimization problem is solved in a few seconds. However, the solution takes between 4664 and 20982 seconds and needs approximately 18 exabytes of memory in larger networks because the SDN-Fog deployment problem is NP-hard. One way to optimize MILP models is to use heuristics like branch-and-bound (BB) and branch-and-cut (BC) [12]. However, the resolution complexity tends not to reduce face to dynamic environment, in other words, when there is an excessive number of variables and restrictions, such as complex IoT network environments and real-world traffic flows [13].

Prediction of network behavior can bring advantages for many applications that have strict Quality of Service (QoS) requirements [14], such as placement of devices, resource allocation, besides allowing service designers to predict the performance of their applications for different load conditions [15]. For example, for the optimization problem presented above, predicting variables that make up the network can be helpful instead of just considering the problem's solution modeled in MILP, which requires a long execution time and high computational power. Furthermore, predicting the network response time can be very useful when planning network capacity, as it can elucidate the behavior of a given configuration without necessarily implementing it. Thus, infrastructure managers can see if their strategy, network components, and devices are enough to ensure a good QoS to end-users.

Therefore, this work presents a solution to predict the response time of Industrial Internet of Things (IIoT) applications considering specific scenarios in SDN-Fog Environments. The main contributions of this work are: *i)* an analysis of the relationship among Fog IIoT Factory Scenario, DCDP, and CPP; *ii)* a new network behavior prediction approach considering the context of SDN-Fog for IIoT; *iii)* the sharing of a framework with pre-trained machine learning models to predict the response time of IIoT networks in the SDN-Fog context.

The remaining of the paper is structured as follows: Section II presents related work. Section III details the work proposal. Section IV presents the experimental design with the description of the Fog IIoT Factory Scenario, the presentation of the machine learning framework, and the details of the data set and the experiments. Section V shows the results obtained and discussions about them. Finally, Section VI concludes the paper and highlights future work.

II. Related Work

In order to predict network behavior, Hardegen *et al.* [13] implemented a model based on Deep Neural Networks (DNNs) to perform 240 flow characteristics prediction experiments. In addition, they presented a case-based study where throughput, flow, and topology were the target predictor variables for specific paths. For predicting network traffic, Aldhyani *et al.* [16] proposed a method using sequence mining, combining non-crisp Fuzzy-C-Means (FCM) clustering and the weight exponential method to improve deep learning Long Short-Time Memory (LSTM) and Adaptive Neuro-Fuzzy Inference System (ANFIS) time series models. When the scenario is specific to cloud-based networks, Nourikhah *et al.* [17] analyzed the response time of 10 real-world services and proposed a prediction model using time series analysis, such as, naïve, mean, auto regressive integrated moving average (ARIMA), and auto regressive fractionally integrated moving average (ARFIMA) methods.

Considering networks for IoT devices, White *et al.* [18] presented the IoTPredict algorithm, which consists of a neighborhood-based prediction approach taking into account the response time and the throughput of a network released by Zheng *et al.* [19]. In addition, Yang and Zhang [20] considered time similarity to predict the response time of IoT service. The method proposed by the authors combines three main steps: (1) a calculation method of service response time similarity between different users; (2) the initial similarity values are adjusted, and a similarity threshold selects similar neighbors to improve prediction accuracy; (3) using a densified user-item matrix, service response time is predicted by collaborative filtering for currently active users. Table I summarizes the main differences between this paper and related work on the nature of the network prediction features, on the scenario, cloud-based, or not, on the context, IoT, or not, and on the scenario's solution, Fog-SDN, or not.

TABLE I
Comparison of Related Works

Work	**Network prediction feature**	**Cloud based**	**IoT context**	**SDN-Fog problem**
[13]	Flow	No	No	No
[16]	Traffic	No	No	No
[18]	Response time	No	Yes	No
[20]	Response time	No	Yes	No
[17]	Response time	Yes	No	No
This work	Response time	Yes	Yes	Yes

III. Work Proposal

Based on supervised and unsupervised learning, we created a Machine Learning (ML) framework to predict the response time of SDN-Fog environments before performing all possible network setups. Fig. 1 illustrates how the ML approach can compose the MILP-based solution. In this context, predicting response times of possible network scenarios (The ML Framework) helps select the most suitable one concerning the expected response time for IIoT applications. Thus, the optimization execution in a later stage (MILP Allocation) becomes less costly and in less time, as the number of scenarios is reduced in the first stage, unlike the previous approach.

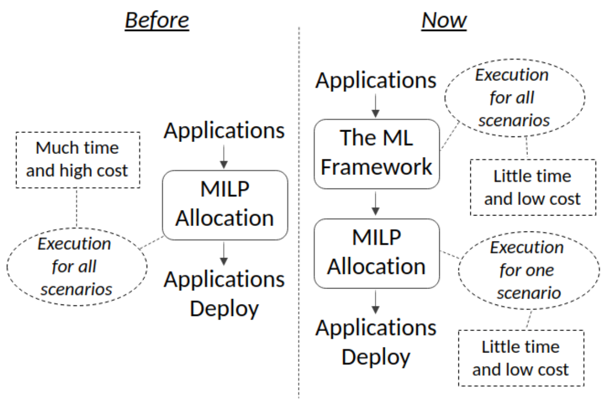


Fig. 1. The machine learning approach.

To ensure that the machine learning framework considers all network characteristics, it uses the data generated by Herrera *et al.* [11], [21] to train a pre-trained model that can predict with high accuracy the expected response time of a particular IIoT network setup, known as a scenario. This pre-trained model will be available as a default functionality of the framework. The framework evaluates how different ML approaches can predict continuous features in this context, such as linear regression, decision trees, and neural networks. In addition, the framework also aims to assess how the unsupervised machine learning technique based on dimensionality reduction impacts prediction.

IV. Experimental Design

In this section, we present the design and some details of the experiments.

A. *Fog IIoT Factory Scenario*

The Fog IIoT Factory Scenario is similar to the environment proposed by Herrera *et al.* [11]. It is based on fog computing to support the QoS requirements of IIoT devices and describes the details needed to predict applications' response time in this context. Fig. 2 illustrates this scenario. It consists of IIoT devices installed in robots and ten fog servers that provide computing services to the devices. Each fog server is directly connected to an SDN switch and has an 800 MHz CPU and 1 GB of RAM. Regarding SDN controllers, the environment includes at least one controller for each switch, being a classic SDN control model.

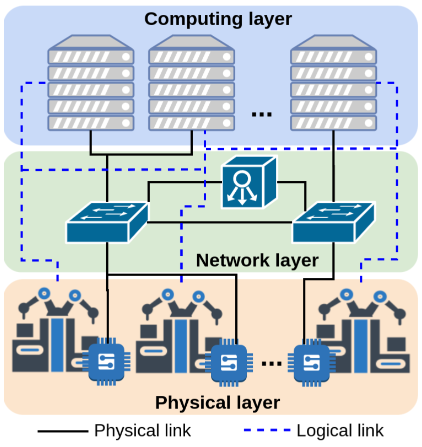


Fig. 2. Fog IIoT factory scenario.

An application that manages the information of robots was designed as a composition of a series of microservices. This application collects the robots' status through sensors connected to IIoT devices and processes the data in fog servers to provide commands accordingly. The commands processed by the microservices can be ordered separately or combined through workflows. Each workflow executes a specific functionality, by linking the deployed microservices depending on the desired behavior. Thus, each microservice has a computational complexity to perform the processing of a given activity. This processing is done by MCycles, being among 100, 500, or 1000.

The combination of the execution time of a microservice plus the communication latency of the network nodes results in the response time of specific functionality. Furthermore, this combination is directly influenced by the location of the microservices on different fog servers and by the SDN controllers available in the environment. All of these factors can cause response time delays. Thus, the reduction of latency between the various network nodes must be sought through the combination of DCDP and CPP to ensure that the response time of each specific functionality is consistent with the strict requirements of IIoT applications.

B. *The Data Set*

The IIoT factory data set consists of 17 features per scenario: computing nodes features such as ID, clock speed (MHz) and RAM (MB); switches IDs; links features such as the node ID that is the source of the link, ID of the node that is the destination of the link, latency (s) and capacity (MB); microservices features such as ID of the microservice, execution cycles (MCycles), RAM required (MB), input and output size of the microservice (MB); and workflows features such as ID, the IDs list of the microservices requested in the workflow, ID of the computing node requesting the workflow and whether the workflow should have a response (always true).

C. *Machine Learning Framework*

The modeling and execution of the SDN-Fog deployment problem in MILP can jointly solve DCDP and CPP to optimize the response time for IIoT applications. However, models' execution requires high computational power, in addition to consuming a relatively large amount of time for networks with more than 300 nodes [11]. Thus, to avoid these issues, the developed framework uses network environment features to predict the response time of IIoT scenarios. Fig. 3 presents the two main parts of the framework. The first part is responsible for processing the input, and output data of the DADO [11], [21], respectively, to compose the scenarios that are the primary inputs to the ML models. The second part is composed of the implementation and execution of supervised and unsupervised learning approaches to evaluate the response time prediction behavior of scenarios composed by the first part.

To compose the scenarios, the data processing step uses the Fog IIoT factory features along with a response time estimation for different workflows to compose the scenarios. Each scenario had features extended statistically according to their average, minimum, 25%, 50%, 75%, and maximum values, accounting for 113 features in total.

The framework composes Linear Regression, eXtreme Gradient Boosting (XGBoost) Regressor, and LSTM network on supervised ML models. The Linear Regression objective is to find the best estimates for its coefficients, which minimize errors in predicting target features. It fits a linear model to minimize the residual sum of squares between the observed

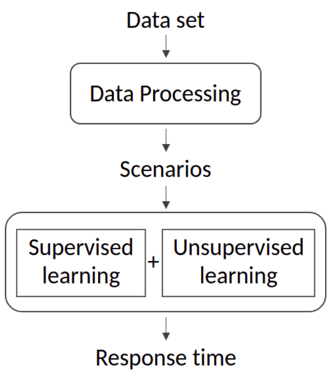


Fig. 3. Machine learning framework.

targets in the data set, in this context, response time, and the targets predicted by the linear approximation. XGBoost is an implementation of Gradient Boosted Decision Trees designed for speed and performance proposed by Chen [22]. This algorithm aims to minimize the loss function, choosing distinct optimal adjustment values for each region of the tree instead of a single one for the entire tree. Based on this concept, XGBoost calculates the fit of its sequential trees according to the weighted errors of the predecessor trees to predict the feature target of the model. Another model presented in the framework is the LSTM network, a type of Recurrent Neural Network (RNN) developed by Hochreiter and Schmidhuber [23] to avoid the long-term addiction problem or vanishing gradients problem, that causes stop learning from further training. LSTM network has a chain structure that, instead of having a single repetition module neural network layer, has four related. This approach allows data-dependent controls into the RNN cell approach to be implemented, ensuring that the gradient of the objective function does not vanish. An LSTM has three such gates to protect and control the state of the cell. In this way, long-term information that was lost during the execution of traditional models based on RNN is now maintained and used in a controlled manner in multi-layer networks [23], [24].

We apply Principal Component Analysis (PCA) to assess how unsupervised ML impacts response time prediction. PCA allows us to reduce the size of a data set while maintaining the features that have the most significant variance contribution [25], [26].

D. Experiments

We present an experimental comparison of ML models implemented using Scikit-learn and XGBoost for all scenarios composed in the data set processing step. The proposed framework and the Jupyter Notebooks used in the experiments are available on Github[1]. We observed three topology categories: *small* (10 IIoT and 10 fog nodes), *medium* (25 IIoT and 15 fog nodes) and *large* (50 IIoT and 25 fog nodes). The universe of data used for training the models contains 84 network scenarios, accounting 880 nodes for small, 1000 nodes for medium and 1125 nodes for large topology. To test the proposed models, the data universe includes 47 scenarios that total 880 nodes for small and 240 nodes for large topology with 30 fog nodes instead of 25. The difference in the amount of fog nodes is original from the source data set and was kept to also evaluate the models performance in distinct structures than those used for training.

[1]https://github.com/rodrigoney/ml-sdn-dado

We carried out a study on the main components using PCA, and we verified four components to compose with supervised learning models. Fig. 4 shows the number of components needed to explain variability of the data. The first component corresponds to 56.87%, the second to 21.44%, the third to 13.01% and the fourth to 4.90% of variability. Therefore, selecting a number of components above four to perform the dimensionality reduction will not drastically influence the model, since other components correspond to only 3.78% of variability. According to the calculated coefficient matrix, the features that most contributed to the first component formation are related to the computational power of IoT devices, such as processing and memory. For the second component, the features of network nodes number, IoT, Fog and Switch, and related to links, such as capacity and latency, were the ones that contributed the most. For the features related to the flow steps, they formed the third component. Finally, for the fourth and last component, the most important features are the workflows sizes and the number of devices per workflow.

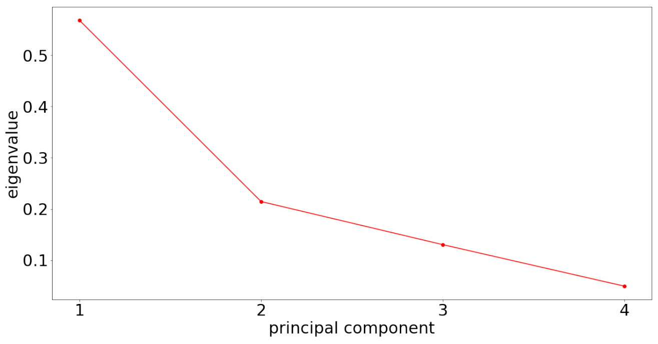


Fig. 4. The impact of the first four principal components.

Model performance validation was implemented using 5-fold cross-validation, i.e., five different measurements validate the implemented models, assessing their generalization ability of predictive and preventing overfitting. In addition, all data used in testing step were totally distinct from those used for training the models.

V. Results and Discussion

Fig. 5 and Fig. 7 illustrate a graphical representation of the Linear Regression and XGBoost performance on the training step, and Fig. 6 and Fig. 8 on the testing step. When comparing the training and testing steps, a factor that impacted the performance drop was the low number of test scenarios, one of the limitations found. Despite this, it is possible to verify by the approximation of the blue line, which represents the predicted response times, that the XGBoost model had the best performance when compared to the Linear Regression.

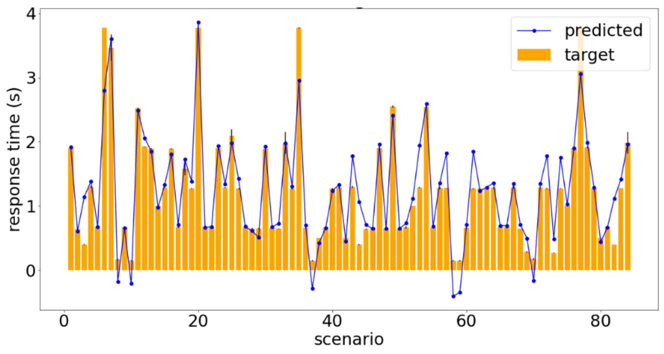


Fig. 5. Linear regression training performance.

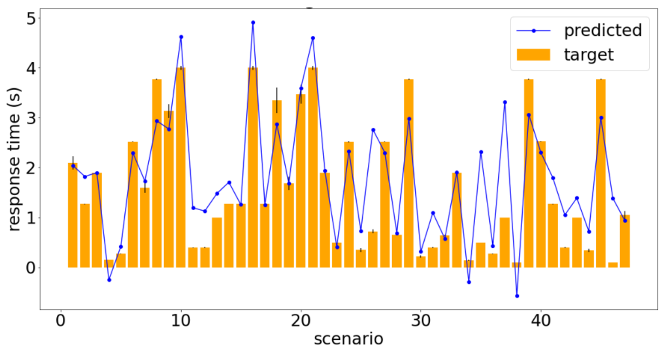


Fig. 6. Linear regression testing performance.

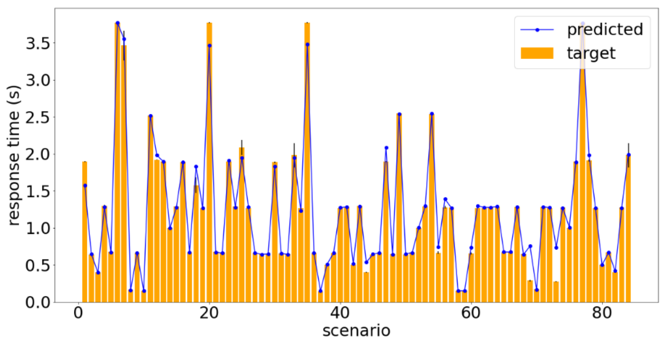


Fig. 7. XGBoost training performance.

Mean Squared Error (MSE) and R^2, or R-squared, score methods were used to evaluate the models accuracy besides the graphical representation. While R2 Score represents the proportion of the variance for a dependent variable that is explained by an independent variable or variables in a regression model, MSE measures the average squared difference between estimated values and the actual value. The resulting accuracy of the implemented models is described in Table II.

Although the four main components represent 96.22% of all data, it is possible to observe that the PCA strategy does not improve both models' performance, neither for the MSE

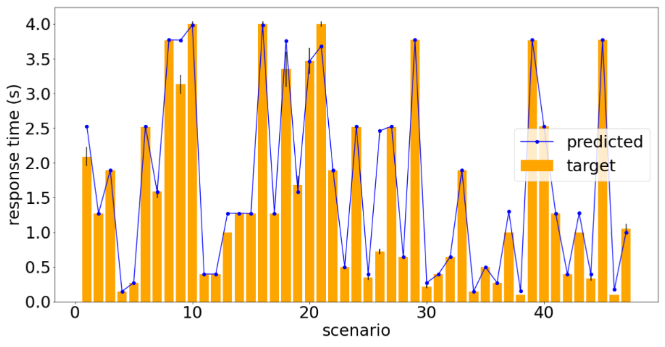


Fig. 8. XGBoost testing performance.

reduction nor the R2 Score increase. Also, when comparing the Linear Regression with PCA and XGBoost with PCA models, it is possible to see a greater negative impact on XGBoost. This phenomenon occurs due to the ineffective reduction of the error calculated through the low composition of features in successive gradient boosted trees. As XGBoost uses the various features to calculate the error and compose it in the learning function, reducing the features of the IIoT environment shown is not a good strategy to predict the network response time. Furthermore, the XGBoost model achieves the best performance at an average rate of 49.60 milliseconds for MSE and 96.63% for R2 Scored.

TABLE II
MODELS ACCURACY WITH RAW DATA

Model	**MSE**		**R2 Score**	
	Training	**Test**	**Training**	**Test**
Linear Regression	0.0822	0.4999	0.8892	0.7025
Linear Regression with PCA	0.1547	0.8215	0.7917	0.5111
XGBoost	0.0107	0.0885	0.9854	0.9473
XGBoost with PCA	0.1977	0.9754	0.7338	0.4196

In order to deepen the study of the impacts of data in relation to the implemented models, two approaches were carried out, normalization and z-score technique application. Table III shows that applying data normalization to the Linear Regression model improves its performance, unlike the XGBoost model, which is improved by applying the z-scored technique. However, both applied techniques do not improve the performance of the models, Table II, when run using raw data from the IIoT network.

TABLE III
MODELS ACCURACY WITH DATA NORMALIZED AND Z-SCORED

Model	**MSE**		**R2 Score**	
	Training	**Test**	**Training**	**Test**
Linear Regression Normalized	0.0857	0.8065	0.8845	0.7392
Linear Regression Z-Scored	0.0924	1.3510	0.8755	0.5632
XGBoost Normalized	0.1796	1.064	0.7582	0.6558
XGBoost Z-Scored	0.0108	0.8904	0.9854	0.7121

VI. Conclusion

This work presented an approach to predict IIoT network response time in an SDN-Fog context. The proposed framework consists of four approaches: Linear Regression, Linear Regression with PCA, XGBoost, and XGBoost with PCA. The experiments showed that the XGBoost is the most suitable for predicting response time in an SDN-Fog environment in terms of accuracy. Overall, the results showed that the prediction of the response time of scenarios was close to the times obtained by solving the problem given by Herrera *et al.* [11], in other words, when compared to the Herrera *et al.* results [21], the framework had an average of fewer than 50 milliseconds of variation according to the best model. Moreover, the approach based on ML models was executed in less than one second for Linear Regression and XGBoost models. Another factor to be considered is the composition of supervised and unsupervised learning models to predict the network response time. In this case, the unsupervised PCA technique did not prove to be an advantageous alternative for the IIoT context presented when composed with the supervised developed models. Therefore, response time prediction in IIoT networks based on ML models shows to be a viable and advantageous option for the SDN-Fog context in network setup planning time. As future work, we seek to evolve and adjust the framework to consider strict network response times and evaluate others machine learning approaches based on neural networks.

Acknowledgments

This work was developed with support of the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq). We thank the research agency for the financial proc. 130762/2021-0. The research is part of the INCT of the Future Internet for Smart Cities funded by CNPq proc. 465446/2014-0, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001, FAPESP proc. 14/50937-1 and proc. 15/24485-9. It is also part of the FAPESP proc. 18/22979-2, proc. 18/23098-0 and the project RTI2018-094591-B-I00 (MCI/AEI/FEDER,UE).

References

[1] M. d. V. D. da Silva, A. Rocha, R. L. Gomes, and M. Nogueira, “Lightweight Data Compression for Low Energy Consumption in Industrial Internet of Things,” in *Proceedings of the IEEE 18th Annual Consumer Communications Networking Conference (CCNC)*, 2021, pp. 1–2.

[2] D. M. Batista, A. Goldman, R. Hirata, F. Kon, F. M. Costa, and M. Endler, “InterSCity: Addressing Future Internet research challenges for Smart Cities,” in *Proceedings of the 7th International Conference on the Network of the Future (NOF)*, 2016, pp. 1–6.

[3] I. Alam, K. Sharif, F. Li, Z. Latif, M. M. Karim, S. Biswas, B. Nour, and Y. Wang, “A Survey of Network Virtualization Techniques for Internet of Things Using SDN and NFV,” *ACM Computing Surveys*, vol. 53, no. 2, pp. 1–40, 2020.

[4] Z. Chi, Y. Li, H. Sun, Y. Yao, and T. Zhu, “Concurrent Cross-Technology Communication among Heterogeneous IoT Devices,” *IEEE/ACM TON*, vol. 27, no. 3, pp. 932–947, 2019.

[5] 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.

[6] 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.

[7] 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.

[8] N. Wang, K. H. Ho, G. Pavlou, and M. Howarth, “An Overview of Routing Optimization for Internet Traffic Engineering,” *IEEE Communications Surveys & Tutorials*, vol. 10, no. 1, pp. 36–56, 2008.

[9] F. Y. Okay and S. Ozdemir, “Routing in Fog-Enabled IoT Platforms: A Survey and an SDN-Based Solution,” *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4871–4889, 2018.

[10] T. Das, V. Sridharan, and M. Gurusamy, “A Survey on Controller Placement in SDN,” *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 472–503, 2020.

[11] J. L. Herrera, J. Galán-Jiménez, J. Berrocal, and J. M. Murillo, “Optimizing the Response Time in SDN-Fog Environments for Time-Strict IoT Applications,” *IEEE Internet of Things Journal*, pp. 1–1, 2021.

[12] M. Pióro and D. Medhi, “CHAPTER 5 - General Optimization Methods for Network Design,” in *Routing, Flow, and Capacity Design in Communication and Computer Networks*, ser. The Morgan Kaufmann Series in Networking, M. Pióro and D. Medhi, Eds. Morgan Kaufmann, 2004, pp. 151–210.

[13] C. Hardegen, B. Pfülb, S. Rieger, and A. Gepperth, “Predicting Network Flow Characteristics Using Deep Learning and Real-World Network Traffic,” *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2662–2676, 2020.

[14] D. Mirkovic, G. Armitage, and P. Branch, “A Survey of Round Trip Time Prediction Systems,” *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 1758–1776, 2018.

[15] S. Bhulai, S. Sivasubramanian, R. van der Mei, and M. van Steen, “Modeling and Predicting End-to-End Response Times in Multi-tier Internet Applications,” in *Managing Traffic Performance in Converged Networks*, L. Mason, T. Drwiega, and J. Yan, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 519–532.

[16] T. H. H. Aldhyani, M. Alrasheedi, A. A. Alqarni, M. Y. Alzahrani, and A. M. Bamhdi, “Intelligent Hybrid Model to Enhance Time Series Models for Predicting Network Traffic,” *IEEE Access*, vol. 8, pp. 130 431–130 451, 2020.

[17] H. Nourikhah, M. K. Akbari, and M. Kalantari, “Modeling and Predicting Measured Response Time of Cloud-Based Web Services using Long-Memory Time Series,” *The Journal of Supercomputing*, vol. 71, pp. 673–696, 2015.

[18] G. White, A. Palade, C. Cabrera, and S. Clarke, “IoTPredict: Collaborative QoS Prediction in IoT,” in *Proceedings of the IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2018, pp. 1–10.

[19] Z. Zheng, Y. Zhang, and M. R. Lyu, “Investigating QoS of Real-World Web Services,” *IEEE Transactions on Services Computing*, vol. 7, no. 1, pp. 32–39, 2014.

[20] H. Yang and L. Zhang, “Response Time Prediction of IoT Service Based on Time Similarity,” *Journal of Computing Science and Engineering*, vol. 11, no. 3, pp. 100–108, 2017.

[21] J. L. Herrera, J. G. Jiménez, J. Berrocal, and J. M. Murillo, “Optimizing Response Time in SDN-Edge Environments for Time-Strict IoT Applications,” 2020, https://dx.doi.org/10.21227/m9s7-1q10. Accessed at October 29, 2021.

[22] T. Chen and C. Guestrin, “XGBoost: A Scalable Tree Boosting System,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, p. 785–794.

[23] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 11 1997.

[24] A. Sherstinsky, “Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network,” *Physica D: Nonlinear Phenomena*, vol. 404, p. 132306, 2020.

[25] M. Alsharif, A. Hilary, K. Yahya, and S. Chaudhry, “Machine Learning Algorithms for Smart Data Analysis in Internet of Things Environment: Taxonomies and Research Trends,” *Symmetry*, vol. 12, p. 88, 01 2020.

[26] D. Rafique and L. Velasco, “Machine Learning for Network Automation: Overview, Architecture, and Applications [Invited Tutorial],” *J. Opt. Commun. Netw.*, vol. 10, no. 10, pp. D126–D143, Oct 2018.

Appendix M

On the Tradeoff Between Load Balancing and Energy-Efficiency in Hybrid IP/SDN Networks

ID : **NOF 2021**

Authors : Jaime Galán-Jiménez, Marco Polverini, Francesco Giacinto Lavacca, Juan Luis Herrera, Javier Berrocal

Publication type : Conference full-paper

Conference : International Conference on Network of the Future (NoF) 2021

DOI : `https://doi.org/10.1109/NoF52522.2021.9609876`

Conference rating : International conference

On the tradeoff between load balancing and energy-efficiency in hybrid IP/SDN networks

Jaime Galán-Jiménez*, Marco Polverini†, Francesco G. Lavacca†, Juan Luis Herrera* and Javier Berrocal*
*Dpt. of Computer Systems and Telematics Engineering, University of Extremadura, Spain. [jaime, jlherrerag, jberolm]@unex.es
†DIET Department, University of Rome ”Sapienza”, Rome, Italy. [marco.polverini, francescogiacinto.lavacca]@uniroma1.it

***Abstract*—The flexibility and programmability provided by the Software-Defined Networking (SDN) paradigm allow network operators to upgrade their legacy IP network infrastructures with the aim of improving their control over the network. However, the full migration from IP to SDN is not straightforward, at least, in the short term. Thus, transitory network infrastructures combining IP and SDN nodes (named hybrid IP/SDN networks) are required to coexist, while the correct coordination between paradigms is crucial to maintain the required Quality of Service. In this paper, two optimization problems that are normally solved separately due to their opposite nature: i) traffic load balancing and ii) the reduction of network power consumption, are jointly considered. In particular, an heuristic named Hybrid Spreading Load Algorithm (HSLA), is proposed to jointly minimize the Maximum Link Utilization (MLU) and the network power consumption during the transition from IP to SDN networks. Simulations over topologies of different size considering diverse selection methods for the replacement of the nodes reveal that HSLA outperforms other state-of-the-art approaches that specifically tackle only one objective, either the traffic load balancing or the reduction of the network power consumption.**

***Index Terms*—hybrid IP/SDN, load balancing, energy efficiency, traffic engineering, heuristic**

I. Introduction

THE way in which information is created, shared and consumed by the digital society is continuously under evolution. A common trend is that the volume of traffic data flowing through the Internet exponentially increases over time. Statistically, the prediction for next years is that there will be 5.3 billion total Internet users (66% of global population) by 2023, from 3.9 billion (51% of global population) in 2018 [1]. This increase in the number of users will result in a threefold rise in the Internet traffic by 2023, with about 2.6 Exabytes for a single day [2].

To deal up with the expected growth in the traffic demand, Internet Service Providers (ISPs) are extending their network infrastructures, by over-provisioning the existing resources or by providing network redundancy, both at node and at link levels. This extension is designed to accommodate peak traffic loads and to be able to react upon events (e.g., link failures [3], routers malfunctioning [4]) that affect network performance.

However, these network architectures derive in a low energy efficiency under normal network operation [5], i.e., all the elements in the network are powered on regardless the traffic load that is currently flowing through it. The irruption of the Software-Defined Networking (SDN) paradigm in the late 2000s, especially motivated by the global network knowledge acquired by the centralized controller and its flexibility and programmability, opened a niche to propose solutions able to find the minimal set of network resources (nodes and links) that can be powered on to steer the required traffic and fulfill the Quality of Service (QoS) requirements [6], [7], [8]. With such adaptive network configurations, an improvement in the network energy efficiency can be achieved [9].

Other aspects such as Traffic Engineering (TE) techniques were also re-thought within the scope of SDN networks. The achievement of traffic load balancing through the design of optimal flow forwarding and routing policies has led to a vast amount of works in the recent years [10], [11], [12].

However, the full deployment of an SDN network from a traditional IP one is not straightforward. A single flag day to migrate the whole network from IP to SDN is unfeasible, due to the economical issues derived from the replacement cost and the lack of provision of services already included in the Service Level Agreement (SLA) between the customers and the ISP. A common and cost-effective approach from the ISP point of view is to upgrade a subset of network elements and evaluate their practicality after a particular period of time [13]. This scenario represents the so-called hybrid IP/SDN networks, in which a coordination between legacy IP routers and upgraded SDN switches is required, as well as in the protocols used to perform the routing (OSPF and OpenFlow) [14], [15].

Although the energy consumption problem has also been studied in the scope of hybrid IP/SDN networks [16]–[19], existing works only focus on the network energy efficiency, not taking into account other key aspects such as link performance w.r.t. important QoS metrics [20]. Furthermore, load balancing represents a major TE method in this type of scenarios, whose purpose is to minimize the Maximum Link Utilization (MLU) by equally spreading the data traffic among links (ECMP protocol) [21]. The average link rate utilization for network links provided by load balancing techniques allow network operators to reduce OPEX and adopt extra users [22], [23].

Energy efficiency and load balancing techniques in hybrid IP/SDN networks have been studied separately. On the contrary, no prior work has addressed the trade-off between the optimization of energy efficient routing and load balancing performance with guaranteed QoS constraints, by also minimizing the MLU. In order to solve the joint problem of energy efficient improvement and load balancing, a multi-objective optimization problem has been defined and formalized. The two considered objectives to be minimized are: i) the network power consumption, and ii) the MLU. In addition, an heuristic algorithm, namely Hybrid Spreading Load Algorithm (HSLA) has been proposed to practically solve it in tractable time.

In the following, the main contributions of this work are listed:

- to consider the trade-off between the energy efficiency and load balancing during the transition from IP to SDN networks;
- to define the multi-objective optimization problem for fairly considering energy efficiency and load balancing in hybrid IP/SDN networks;
- to derive an heuristic algorithm, namely Hybrid Spreading Load Algorithm (HSLA), to practically solve the problem;
- to compare HSLA with other state-of-the-art solutions both in the energy efficiency and load balancing research areas on realistic scenarios;

The rest of the paper is organized as follows. Problem definition is described in Sec. II. The description of the proposed heuristic can be found in Sec. III. Experimental results are reported and analysed in Sec. IV. Finally, Sec. V draws some conclusions and future works.

II. Problem Definition

Let us consider a hybrid IP/SDN network modelled as a directed network graph $\mathcal{G} = (\mathcal{N}, \mathcal{L})$, where $\mathcal{N}$ is the set of nodes and $\mathcal{L}$ is the set of directed links. Considering the set of nodes $\mathcal{N}$, they can be SDN nodes, $\mathcal{N}_{SDN} \in \mathcal{N}$, or IP nodes, $\mathcal{N}_{IP} \in \mathcal{N}$, where $\mathcal{N}_{SDN} \cup \mathcal{N}_{IP} = \mathcal{N}$. Regarding the set of links, each link $l_{i,j} \in \mathcal{L}$ connecting node $i \in \mathcal{N}$ to node $j \in \mathcal{N}$ has a capacity of $C_{i,j}$ units to accommodate traffic flows Furthermore, the link i, j has a power consumption referred to as $P_{i,j}$.

Traffic is modelled by means of a traffic matrix $\mathcal{T}$, which is composed of a set of demands $\delta_{s,d} \in \mathcal{T}$ from each source node $s \in \mathcal{N}$ to each destination node $d \in \mathcal{N}$ in the network. Specifically, the traffic demand entering the network from node s and leaving it through the node d is referred to as $\delta_{s,d} \in \mathcal{T}$.

The objective is to find an optimal network configuration that jointly minimizes the network power consumption and the maximum link utilization.

Therefore, a set of variables are required to define the Mixed Integer Linear Programming (MILP) formulation aimed at solving the Multi-Objective and Multi-Commodity-Flow (MO-MCF) optimization problem at hand:

- $x_{i,j}$ represents a binary variable related to the operational mode of link $l_{i,j}$. Its value is $x_{i,j} = 1$ if the link is active. Otherwise, if $x_{i,j} = 0$, the link is put to sleep;
- $f_{i,j}^{s,d}$ is an integer variable representing the amount of traffic demand $\delta_{s,d} \in \mathcal{T}$ that is routed on the link $l_{i,j}$;
- $n_{i,j}^{d}$ is a binary variable indicating whether the node i uses node j as next hop in its forwarding table to reach the destination node d;
- p_i^d is an integer variable representing the cost of the path to go from node i to node d;
- $w_{i,j}$ is an integer variable indicating the Interior Gateway Protocol (IGP) weight of the link i, j, which is considered by the IP routers to determine the shortest path routing;
- θ is a real variable indicating the Maximum Link Utilization (MLU) in the network.

With the variables described above, the optimization problem to solve aims at jointly minimizing the power consumption of the network as well as its maximum link utilization. For this purpose, two objective functions are required to solve the multi-objective optimization problem. They are defined in eqs. (1-2):

$$F_1 = min \sum_{l_{i,j} \in \mathcal{L}} x_{i,j} \cdot P_{i,j} \tag{1}$$

$$F_2 = min \quad \theta \tag{2}$$

Eq. (1) represents the minimization of the global network power consumption, while the goal pursued by eq. (2) is to minimize the MLU. Thus, the joint optimization of both functions composes the MO-MCF problem as follows:

$$min\,[F_1; F_2] \tag{3}$$

subject to

$$\sum_{j \in \mathcal{N}_i^-} f_{i,j}^{s,d} - \sum_{j \in \mathcal{N}_i^+} f_{j,i}^{s,d} = \begin{cases} \delta_{s,d} & \text{if } i = s \\ -\delta_{s,d} & \text{if } i = d \\ 0 & \text{if } i \neq s, d \end{cases} \quad \forall i, s, d \in \mathcal{N}, \delta_{s,d} \in \mathcal{T} \tag{4}$$

$$\sum_{\delta_{s,d} \in \mathcal{T}} f_{i,j}^{s,d} \leq x_{i,j} \cdot C_{i,j} \qquad \forall i, j \in \mathcal{L} \tag{5}$$

$$\frac{1}{C_{i,j}} \sum_{\delta_{s,d} \in \mathcal{T}} f_{i,j}^{s,d} \leq \theta \qquad \forall i, j \in \mathcal{L} \tag{6}$$

$$f_{i,j}^{s,d} \leq n_{i,j}^{d} \cdot \delta_{s,d} \qquad \forall i, j \in \mathcal{L} : i \in \mathcal{N}_{\text{IP}}, \delta_{s,d} \in \mathcal{T} \tag{7}$$

$$\sum_{j \in \mathcal{N}_i^-} n_{i,j}^{d} = 1 \qquad \forall i, d \in \mathcal{N} : i \neq d \tag{8}$$

$$0 \leq p_j^d + w_{i,j} - p_i^d \leq (1 - n_{i,j}^{d}) \cdot M \qquad \forall i, j \in \mathcal{L}, d \in \mathcal{N} \tag{9}$$

$$1 - n_{i,j}^{d} \leq p_j^d + w_{i,j} - p_i^d \qquad \forall i, j \in \mathcal{L}, d \in \mathcal{N} \tag{10}$$

$$1 \leq w_{i,j} \leq w_{\max} \qquad \forall i, j \in \mathcal{L} \tag{11}$$

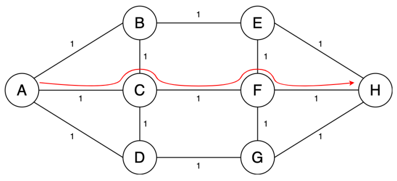


Fig. 1: Path from A to H without SDN nodes.

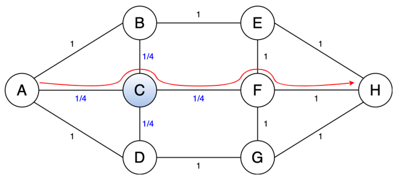


Fig. 2: Path from A to H with 1 SDN node (C).

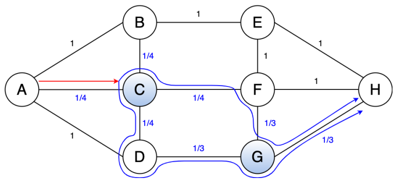


Fig. 3: Path from A to H with 2 SDN nodes (C, G).

Eq. 4 describes the classical flow conservation constraints. Eq. 5 represents the capacity constraint on links, where the total traffic routed on a link must be upper bounded by the available capacity. In particular, if the link is in on status ($x_{i,j} = 1$), all of its capacity can be exploited to transport the traffic, otherwise, its capacity is equal to 0, thus the link cannot be used. Eq. 6 forces the variable θ to be equal or greater than the MLU in the network. In the presented model, IP routers are forced to use a single and shortest path routing, and to forward traffic according to the destination. These aspects are included in the problem formulation through the set of constraints expressed in Eqs. 7-11. In particular, Eq. 7 models the destination based forwarding, by allowing the IP router i to send traffic destined to the node d over the link i, j only if the node j is the next hop for the destination ($n^d_{i,j} = 1$). The single path routing for IP routers is imposed by the Eq. 8, whose logic is to force the node i to choose only one next hop to reach the destination node d. Finally, the shortest path policy is enforced through the Eqs. 9-11. They impose that if $n^d_{i,j} = 1$, then the cost of the path between the nodes i and d must be equal to the weight of the arc connecting it with its neighbor j ($w_{i,j}$) plus the cost of the path between j and d.

Since the presented problem formulation is NP-hard [24], in the next an heuristic approach is presented to solve it in polynomial time.

III. Hybrid Spreading Load Algorithm (HSLA)

This section describes the proposed algorithm, namely Hybrid Spreading Load Algorithm (HSLA), which is proposed to solve the MILP problem defined in Sec. II. In particular, the algorithmic aspects and the architectural details are highlighted.

A. Exploiting SDN nodes for traffic splitting

The main idea behind HSLA is to exploit the use of the *generalized forwarding* of the SDN paradigm to split the traffic entering the set of SDN nodes in order to balance the traffic load. In this way, a flow reaching an SDN node is split among the rest of the outgoing ports so that the traffic is equally balanced toward the next hop (ECMP protocol [21]). In case of IP nodes, destination-based forwarding is used, such as in the case of traditional routing protocols (e.g., OSPF) [10].

Although the ECMP technique can be combined with OSPF to balance the traffic load on legacy IP nodes, the traffic splitting is based on flow hashing methods, thus allowing the balancing only in terms of number of flows (not with respect to the actual traffic volume) and leading to a low accuracy. On the other hand, in the SDN paradigm the traffic splitting can be performed in a more flexible way by, e.g., exploiting the routing mechanism of group tables or using specific flow rules to realize a more accurate split of the traffic.

Figs. 1-3 represent the idea behind the proposed routing scheme. Considering an 8-node hybrid IP/SDN network, let us focus on the path that is assessed to steer the flow traffic from node A to node H. Red line in Fig. 1 represents the resulting path for the case where all the nodes are legacy IP ones ($P_{A,H} = \{A - C - F - H\}$). Note that all the weights for links are set to 1 for the sake of simplicity. In this case, destination-based forwarding is applied, such as in traditional routing protocols, e.g., OSPF. In case a traditional IP node is replaced by a SDN one at a particular stage in the transition from IP to SDN, and the selection method for such replacement indicates that the upgraded node must be C, the resulting scenario is represented by Fig. 2. In this case, the set of weights for links associated to the SDN node are decreased from $w_{i,j} = 1$ (initial setting) to $w_{i,j} = \frac{w^-_{i,j}}{k_i} = \frac{1}{4}$, where $w^-_{i,j}$ represents the weight of link $l_{i,j}$ before the update and k_i is the degree of node i. The motivation behind the action of decreasing the weights of links related to SDN nodes is to force the flows to pass through them and hence to exploit the ECMP protocol [21] in order to balance the traffic among different links. However, in this simple example, although link weights are modified, the replacement of a single SDN node does not impact on the path followed by flow $A-H$ (red line) compared with the case of Fig. 1.

It is in the case of upgrading a second node to SDN, e.g., G, where the flow arriving at node C is equally splitted among the set of shortest paths from it toward the destination (see Fig. 3). In the example, there are two shortest paths with the same cost from C to H: $P^1_{C,H} = \{C - D - G - H\}$ and $P^2_{C,H} = \{C-F-G-H\}$, which are represented by blue lines. Therefore, the volume of traffic represented by the red line is divided by two and steered through the aforementioned (blue) paths. With this splitting of traffic, the proposed solution aims at balancing the traffic load and reducing the MLU through the use of the ECMP protocol at SDN nodes.

B. HSLA Description

In the next, the proposed solution is presented, which is composed of 2 algorithms: i) the first one aims at spreading the traffic through the subset of SDN nodes, thanks to the

Algorithm 1 Hybrid Spreading Load Algorithm (HSLA) pseudo code.

```
Require: A network graph: $\mathcal{G} = (\mathcal{N}, \mathcal{L})$, a traffic matrix: $\mathcal{T}$
 1: create $\mathcal{W} \leftarrow \emptyset$                               ▷ Set for link weights
 2: create $\mathcal{R} \leftarrow \emptyset$                               ▷ Routing matrix
 3: for all $l_{i,j} \in \mathcal{L}$ do                                    ▷ Update link weights
 4:     if $i \in \mathcal{N}_{\text{SDN}}$ then
 5:         $w_{i,j} \leftarrow \frac{1}{k_i}$
 6:     else
 7:         $w_{i,j} \leftarrow 1$
 8:     end if
 9: end for
10: for all $\delta_{s,d} \in \mathcal{T}$ do                               ▷ Find path $\forall \delta_{s,d} \in \mathcal{T}$
11:     create $p_{s,d} \leftarrow \emptyset$                               ▷ Path for demand $\delta_{s,d}$
12:     create $\lambda \leftarrow$ false                                    ▷ Boolean for path found
13:     $P \leftarrow$ k_shortest_paths$(\delta_{s,d}, \mathcal{G})$
14:     while $\lambda ==$ false do                                          ▷ Check each path $p \in P$
15:         if $p \in P$ is an IP path then                                  ▷ No SDN nodes
16:             $b \leftarrow$ check_link_constraints$(\delta_{s,d}, p, \mathcal{G})$
17:             if $b ==$ true then
18:                 $\mathcal{R} \leftarrow$ assign_flow_to_path$(\delta_{s,d}, p, \mathcal{G})$
19:                 $p_{s,d} \leftarrow p$
20:                 $\lambda \leftarrow true$
21:             end if
22:         else                                                             ▷ There are SDN nodes in the path
23:             for all $l_{i,j} \in p$ do
24:                 if $i \in \mathcal{N}_{\text{SDN}}$ then                 ▷ It is an SDN node
25:                     $P^* \leftarrow$ k_shortest_paths$(\delta_{i,d}, \mathcal{G})$
26:                     $u \leftarrow$ num_paths_same_cost$(P^*)$
27:                     for all $p' \in P^*$ with same cost do
28:                         $b \leftarrow$ check_link_constraints$(\frac{\delta_{s,d}}{u}, p', \mathcal{G})$
29:                         if $b ==$ true then
30:                             $\mathcal{R} \leftarrow$assign_flow_to_path$(\frac{\delta_{s,d}}{u}, p', \mathcal{G})$
31:                             $p_{s,d} \leftarrow p_{s,d} \cup p'$
32:                         else
33:                             clear incomplete flow assignments
34:                             check next path $p \in P$ (line 14)
35:                         end if
36:                     end for
37:                     $\lambda \leftarrow true$
38:                 else                                                     ▷ It is an IP node
39:                     $b \leftarrow$ check_link_constraints$(\delta_{s,d}, l_{i,j}, \mathcal{G})$
40:                     if $b ==$ true then
41:                         $\mathcal{R} \leftarrow$ assign_flow_to_link$(\delta_{s,d}, l_{i,j}, \mathcal{G})$
42:                         $p_{s,d} \leftarrow p_{s,d} \cup l_{i,j}$
43:                     else
44:                         clear incomplete flow assignments
45:                         check next path $p \in P$ (line 14)
46:                     end if
47:                 end if
48:             end for
49:         end if
50:     end while
51: end for
52: return feasible routing matrix $\mathcal{R}$ if it exists
```

use of the ECMP protocol (pseudo-code reported in Alg. 1); ii) once the traffic is handled, a greedy approach is used to save energy by iteratively removing the less loaded link in the network (see Alg. 2) and repeat i).

Starting with HSLA, 2 parameters are required as input: i) the network graph $\mathcal{G} = (\mathcal{N}, \mathcal{L})$; and ii) a traffic matrix $\mathcal{T}$.

Algorithm 2 Less Loaded Links Algorithm (L3A) pseudo code.

```
Require: A network graph: $\mathcal{G} = (\mathcal{N}, \mathcal{L})$, the routing matrix obtained
    by Alg. 1: $\mathcal{R}$, a traffic matrix: $\mathcal{T}$
 1: create $\mathcal{G}' \leftarrow \emptyset$                              ▷ Aux network graph
 2: create $\mathcal{L}^* \leftarrow \emptyset$, $\mathcal{L}' \leftarrow \mathcal{L}$       ▷ Aux links set
 3: create $\mathcal{R}' \leftarrow \emptyset$                              ▷ Output routing matrix
 4: $\mathcal{L}^* \leftarrow$ sort_links_by_load$(\mathcal{G}, \mathcal{R}$, 'ascend')
 5: for all $l_{i,j} \in \mathcal{L}^*$ do
 6:     $\mathcal{L}' \leftarrow \mathcal{L}' - l_{i,j}$                    ▷ Remove link from the set
 7:     $\mathcal{G}' = (\mathcal{N}, \mathcal{L}')$
 8:     $\mathcal{R}' \leftarrow$ HSLA$(\mathcal{G}', \mathcal{T})$
 9:     if $\mathcal{R}'$ is not feasible then
10:         $\mathcal{L}' \leftarrow \mathcal{L}' \cup l_{i,j}$
11:     end if
12: end for
13: return feasible routing matrix $\mathcal{R}'$ if it exists
```

At first, an initialization phase for setting the link weights is performed in lines $(1-9)$. As introduced in Sec. III-A, the weight for links connecting two IP nodes is set to $w_{i,j} = 1$, while they are decreased to $w_{i,j} = \frac{1}{k_i}$ in case they connect an SDN node i, with k_i as the node degree.

Once the initial weight setting is carried out, the goal of HSLA is to accommodate all the traffic demands of the traffic matrix passed as input, i.e., to find a feasible routing matrix $\mathcal{R}$ for $\mathcal{T}$ in which the MLU is minimal. For this purpose, each traffic demand (s, d) is iteratively handled. At first, k shortest paths are obtained from the source node s to the destination d according to destination based forwarding (line 13).

Then, each path in the reported set is evaluated to check if it is a *non-controllable path* (all the nodes are IP) or a *controllable* one (at least one node is SDN). In the former case (lines $15-23$), a classical destination based forwarding is adopted to steer the traffic, checking that link constraints are satisfied. If otherwise SDN nodes are found in the evaluated path (it is a *controllable path*), the algorithm checks each node in the path from s to d to discriminate whether they are SDN or IP nodes (lines $24-49$). If the evaluated node i is SDN, the k shortest path procedure is executed again (line 27) to obtain the subset of u paths with same cost from the SDN node i to the destination d. Then, the traffic demand (s, d) arriving at i is equally splitted among the subset of u paths with equal cost to d, with a volume of $\frac{(s,d)}{u}$, checking that no loops are present (lines $29-37$). In case the evaluated node i is IP, the traffic demand is fully accommodated over the outgoing link, without performing the traffic splitting (lines $39-47$). This process is repeated until all the nodes in the path from s to d are evaluated and the traffic is successfully steered, satisfying link constraints throughout the path and the avoidance of loops. Finally, a feasible routing matrix $\mathcal{R}$ is reported as output.

Once a feasible routing is found by Alg. 1, in which the traffic load is balanced, a second algorithm named Less Loaded Links Algorithm (L3A) is executed to find a feasible network configuration that minimizes the required power consumption.

L3A pseudo code is reported in Alg. 2. It takes as input the routing matrix outputted by HSLA, $\mathcal{R}$, as well as the network graph and the traffic matrix to be handled. The first step is to sort the set of links according to their load in ascending order (line 4). Iteratively, each link is removed (put to sleep) from the network topology and HSLA is executed to find a feasible routing (if possible). If no feasible routing is found, the link is put back to the topology (switched on) and the next link is considered. This process is repeated until all the links in the network are evaluated. Finally, a feasible routing matrix $\mathcal{R}'$ is reported, which i) balances the link load, and ii) reduces the network power consumption.

IV. Experimental Results

In this section, an experimental evaluation of the proposed solution is provided. At first, the simulation environment is described. Next, a performance analysis is provided, in which the benefits of applying HSLA over topologies of different sizes and traffic loads are discussed. Finally, a comparison with two representative state-of-the-art solutions which tackle separately the load balancing and energy consumption problem in hybrid IP/SDN networks is carried out.

A. Simulation Set-up

Three network topologies of different size are considered in the simulations: Nobel (17 nodes, 52 links), Geant (22 nodes, 72 links), and Germany (50 nodes, 176 links). In order to evaluate the variability of the daily traffic patterns in each network, a set of traffic matrices are retrieved from [25]. In particular, Fig. 4 shows the daily traffic load of Nobel, for which 5 representative TMs are extracted (TM1-TM5). As it can be seen, the peak TM (TM5) is scaled down with a factor of $0.4, 0.5, 0.7, 0.9$ to form TM1-TM4.

Link capacities are set as follows. First, we select the peak TM (TM5) and we route it over a set of shortest paths derived after the application of the Dijkstra algorithm on the network graph. After this step, each link $l_{i,j}$ is carrying an amount of traffic $t_{i,j}$. Then, we assume that the capacity of each link can be upgraded by installing a set of line cards. A line card has a capacity of Δ_C equal to $0.5 \max\limits_{l_{i,j}}(t_{i,j})$, i.e., the half of the traffic carried by the link with highest link utilization. Finally, we consider to install the minimum number of line cards needed by each link to make their utilization not greater than 100%.

We assume that all the links in the network have the same power consumption when they are active ($P_{i,j} = 1$). It is worth to remark that, although this assumption is unrealistic (there can be different types of links with different values of power consumption [26]), it allows us to perform a much fair comparison with the benchmark algorithms, since their objective is to minimize the number of active links in the network.

The transition from IP to SDN networks is performed step-by-step, i.e., by upgrading one IP legacy node to SDN at a time. At each stage, one node is selected as most suitable to be upgraded to SDN. For the selection process, three methods

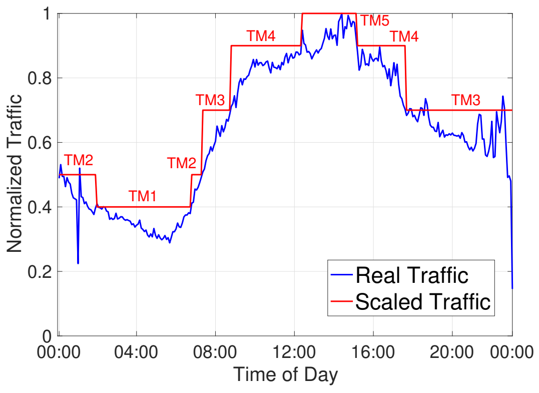


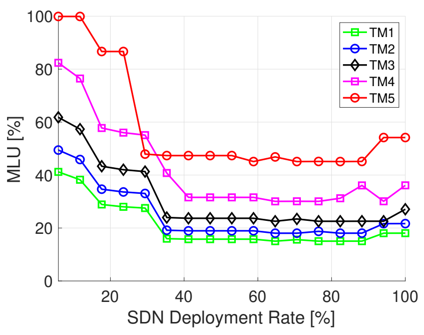
Fig. 4: Daily traffic for Nobel topology.

are considered: i) Highest Degree First (HDF), which sorts the set of potential nodes (remaining IP nodes) according to their degree; i.e., the number of incoming/outgoing links; ii) Highest Closeness Centrality (HCC), to select the node having the shortest distance to the rest of the nodes; and iii) Highest Betweennes Centrality (HBC), with the aim of obtaining the node that is involved in most of the shortest paths among the nodes in the network.

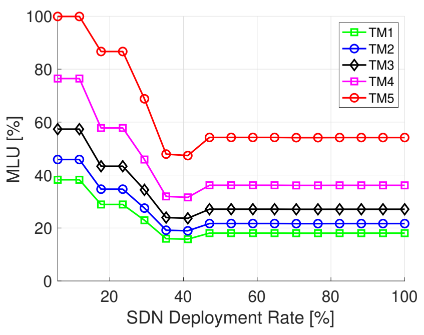
The motivation of the choice of the three aforementioned selection methods is explained next. If the upgrade of IP nodes to SDN is performed according to the HDF method, it means that, after each stage, the controller will have the biggest possible amount of information under control, compared with the case in which other potential node would be selected. In this way, the node with the highest number of links is chosen, so that the controller will be able to control the highest number of ports through, e.g., the use of the *OFPMP_PORT_STATS* messages. If HCC is the method for SDN nodes selection, the focus is put on the distance of a node to the rest of the nodes. This situation is interesting to detect nodes that are able to spread information very efficiently throughout the network. If a node is selected as SDN using HCC, it means that the destination node (as well as the source node) is close, thus the path is shorter w.r.t. other potential nodes, and improvements on the energy efficiency can be experienced. Finally, with HBC the importance of a node is evaluated. The node with HBC is involved in most of the shortest paths for each source-destination pair. Since HSLA is based on the computation of shortest paths both from the source node (line 13 of Alg. 1) and from each SDN node (if any) to the destination (line 27 of Alg. 1), HBC seems an interesting option to be exploited in the SDN nodes selection process.

B. HSLA Performance Evaluation

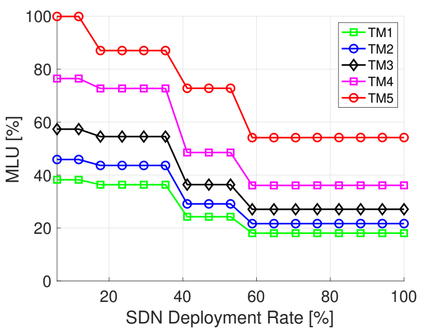
The first analysis that is proposed aims at evaluating the effectiveness of the proposed solution considering the joint optimization of energy efficiency and load balancing during the transition from IP to SDN nodes. Fig. 5 shows the MLU obtained by HSLA as a function of the SDN deployment rate for Nobel topology. In particular, results for each of the 5 TMs



(a) HDF

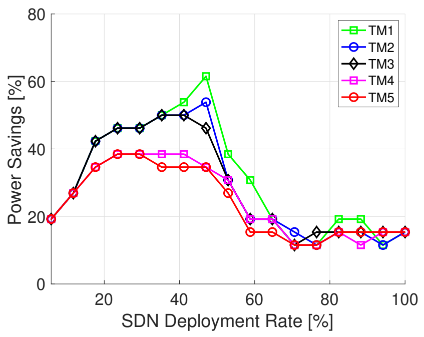


(b) HCC

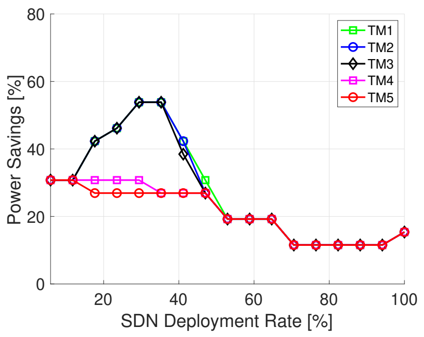


(c) HBC

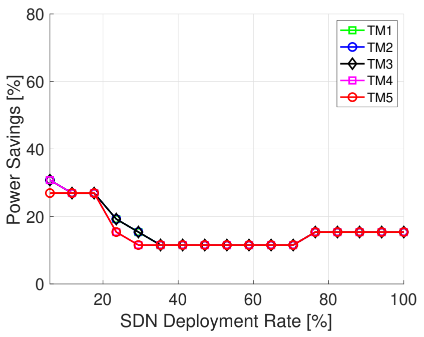
Fig. 5: Maximum Link Utilization as a function of the SDN deployment rate for Nobel.



(a) HDF



(b) HCC



(c) HBC

Fig. 6: Power Saving as a function of the SDN deployment rate for Nobel.

are reported for the three SDN node selection methods (HDF, HCC and HBC).

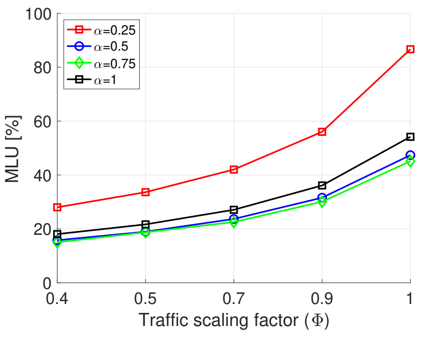
A general trend can be extracted by looking at Fig. 5, since the MLU decreases with the SDN deployment rate regardless the selection method. Clearly, there exists a direct proportional relationship between the overall traffic load that is injected to the network and the obtained MLU. TM5 represents the worst-case scenario and TM1 the case for non-peak traffic, in which more energy savings opportunities can be found.

A significant decrease in the obtained MLU is experienced every time an IP node is replaced by an SDN one when the SDN deployment rate is below 40%, both for HDF (Fig. 5a) and HCC (Fig. 5b). After such percentage, the achieved MLU is approximately constant up to reaching the full SDN scenario (deployment rate of 100%). In other words, the highest impact during the transition from IP to SDN in terms of MLU is found in the range $(0, 40)\%$ of deployment rate. It is in Fig. 5c where the MLU behaviour is step-wise. In this case, the benefits of adding a new SDN node is not as high as in the case of HDF, or HCC, regardless the transition stage.

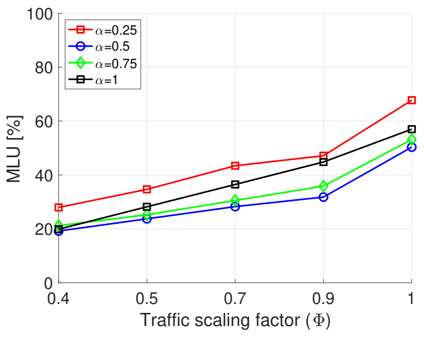
Once the evaluation of the obtained MLU has been carried out, we move our attention to the power savings achieved by HSLA throughout the IP/SDN transition process. In Fig. 6, the power savings are reported for the 5 TMs and the three selection methods, considering different percentages of SDN deployment rate. By looking at the three sub-figures, it is clear that HDF is the method that outperforms the rest, reaching a power saving peak about 50% when the half of the network nodes are upgraded to SDN. In this case, the experienced trend is different from the one experienced when evaluating the MLU in Fig. 5: the power savings are increased with the SDN deployment rate up to reaching a threshold that slightly exceeds the half of the nodes considered as SDN. After such value, the power savings sharply decrease and no significant differences can be found among traffic matrices and with the increase of the deployment rate. Although HCC method slightly improves the power savings outcomes w.r.t. HDF for a deployment rate of 30% and low traffic scenarios (see Fig. 6b), it suffers when the traffic load is increased. Finally, HBC has been found as the worst selection method for the power savings objective. As a summary, the SDN selection method that presents the best results both for the minimization of the MLU and the minimization of the power consumption is HDF. An explanation to this outcome is derived by the fact that such method aims at selecting the node with the highest degree, hence involving the maximum number of links and controllable paths.

C. *Performance analysis over bigger networks*

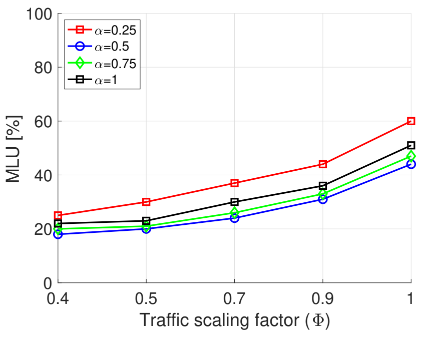
The goal of the next analysis is to evaluate the outcomes of the proposed solution over topologies with different size. In particular, Fig. 7 reports the MLU as a function of the traffic



(a) Nobel

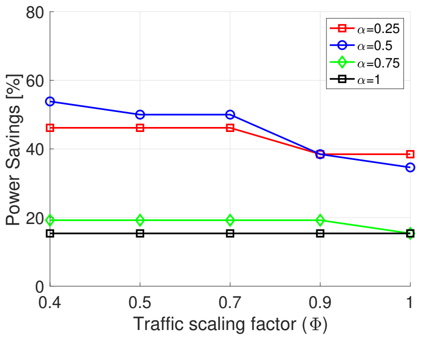


(b) Geant

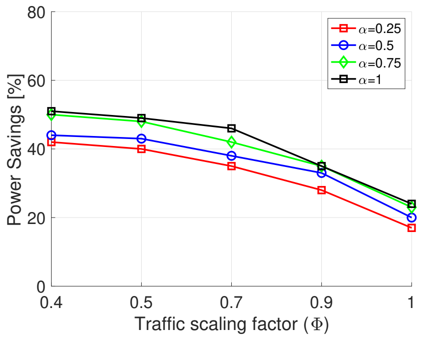


(c) Germany50

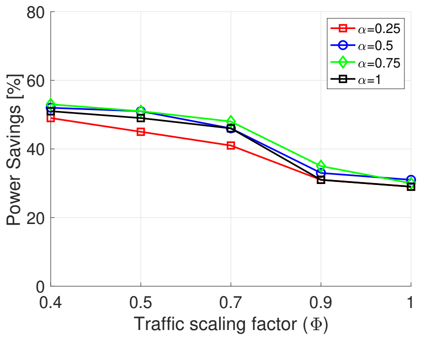
Fig. 7: Maximum Link Utilization as a function of the traffic load for Nobel, Geant and Germany50, with HDF.



(a) Nobel



(b) Geant



(c) Germany50

Fig. 8: Power Saving as a function of the traffic load for Nobel, Geant and Germany50, with HDF.

scaling factor for the three considered topologies: Nobel, Geant and Germany50. For each topology, 4 deployment rate percentages (α) have been considered, approximating to 25%, 50%, 75% and 100% of the nodes in the network. HDF is adopted as the nodes selection method. From the figures, it can be extracted that the MLU increases with traffic, as previously discussed when analyzing Fig. 5, for the three topologies. Surprisingly, the best outcomes for the medium and big topologies are not obtained in the full SDN scenario ($\alpha = 100\%$), but when the half of the nodes are replaced (see Figs. 7b and 7c). In case of Nobel, in turn, the minimal MLU is experienced when $\alpha = 75\%$. In any case, the gap with the half hybrid IP/SDN network is negligible.

Power savings outcomes are reported in Fig. 8 as a function of the traffic load for the three considered topologies and different values of α. By inspecting Fig. 8, there are differences between the results obtained for small and large topologies. In the latter case (Geant and Germany50), a decreasing trend in the power savings can be extracted with the increase in the traffic scaling factor. Moreover, a higher value of α is generally associated with power saving gains, especially for Geant. It is in the case of Nobel (Fig. 8a) where the behaviour is the opposite. The highest power savings are achieved for values of $\alpha \leq 50\%$, as previously discussed when analyzing Fig. 6a. Once this threshold is surpassed, the power savings are significantly decreased, regardless the network traffic load.

The explanation to the obtained power saving results is as follows. The proposed algorithm aims at either balancing the traffic load and to reduce the network power consumption. In the first stage, HSLA, the link weights are adjusted so that to create a large portion of the controllable traffic. As a consequence, the main role of the SDN nodes is to balance the traffic, and not to reduce the power consumption (goal of the second stage, L3A). At the same time, the weights selection logic becomes less effective when the deployment ratio increases (approaching to a full SDN network), since the amount of controllable traffic is already high. Therefore, the algorithm reaches its performance peak when the network is half IP-half SDN.

D. *Comparison with other solutions*

In this subsection, two benchmark solutions are first introduced to compare their results with the ones obtained by the proposed heuristic. Then, a performance analysis is carried out to show the outcomes of the proposed solution.

The first algorithm we focus to highlight the benefit allowed by the use of HLSA is named *SDN/OSPF Traffic Engineering* (SOTE) and is proposed in [27]. The main goal of SOTE is to minimize the MLU of a hybrid SDN network, which is one of the metrics tackled in the present work. However, energy efficiency aspects are not taken into account. To solve the problem at hand, SOTE jointly optimizes the OSPF weight setting of the whole network to balance outgoing flows at IP nodes, as well as the traffic splitting ratio of flows that aggregate at SDN nodes.

The second algorithm that is considered is named *Hybrid Energy-Aware Traffic Engineering* (HEATE) and is proposed in [18]. This solution minimizes the power consumption of a hybrid IP/SDN network by finding the optimal setting of OSPF link weight and splitting ratio on SDN nodes. In particular, IP routers perform shortest path routing by optimizing OSPF link weights and SDN nodes fulfil multi-path routing with traffic flow splitting through the action of the controller.

These two algorithms have been selected to be compared with HSLA since they are both representative when tackling the load balancing problem [27] and the energy consumption problem [18] in hybrid networks. Although works such as [2] and [28] consider the joint problem of load balancing and energy efficiency in these scenarios, both of them are survey papers that review a set of works tackling each type of problem ([2] considering Machine Learning aspects), not the combined one. Moreover, no proposals to compare with are provided.

Results for the first analysis are reported in Fig. 9, in which the Cumulative Distribution Function (CDF) curves of MLU are drawn for HSLA, SOTE and OSPF. The deployment ratio is set to $\alpha = 50\%$, i.e., a half hybrid IP/SDN network, and the non-peak TM (TM1) is considered. As shown in Fig. 9, HSLA achieves the lowest MLU compared with the rest of solutions. SOTE, which is specifically designed to minimize the MLU, presents slightly worse results, with an average of 1% of difference in the MLU w.r.t. HSLA in about 80% of the links. The difference is that HSLA allows for a better load balancing in the less congested links with respect to SOTE. Furthermore, the performance of OSPF is decreased due to the fact that its goal is not to minimize the MLU, but to find the shortest path among each source-destination pair.

In order to compare the power saving gains between HSLA and HEATE, Fig. 10 shows the obtained results as a function of the SDN deployment rate for Geant topology. Two lines are reported for each solution, for the peak and non-peak TMs, respectively (TM5 and TM1). By looking at the figure, it can be seen that HSLA outperforms HEATE in all the cases, with an average power saving gain of 3.4% for the non-peak TM, and of 1.8% for the peak TM. As a summary, the proposed solution, which jointly minimizes the MLU and the power consumption, presents better results than the ones obtained by ad-hoc solutions that specifically tackle a single objective.

V. Conclusion

This paper analyzes the tradeoff between load balancing and energy-efficiency in hybrid IP/SDN networks. Starting from the opposite nature of the two optimization problems, which are normally tackled separately, an heuristic is proposed to jointly minimize the MLU and the network power consumption during the transition from IP to SDN networks. Simulations on realistic network topologies show that HSLA outperforms other state-of-the-art solutions such as SOTE,

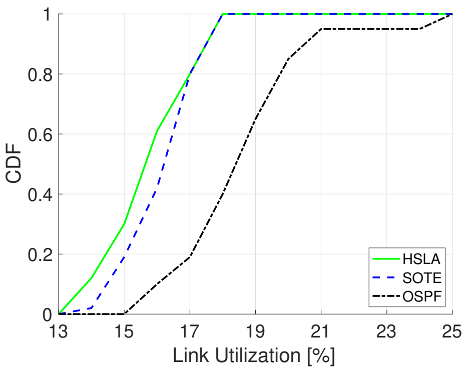


Fig. 9: CDF curves of link utilization for HSLA, SOTE and OSPF over Nobel topology considering TM1.

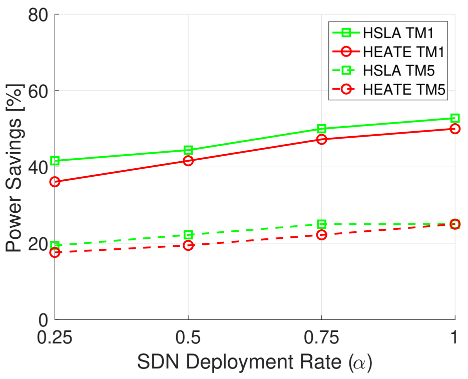


Fig. 10: Comparison of power saving achieved by HSLA and HEATE for TM1 and TM5 over Geant topology.

which is intended only to minimize the MLU, and HEATE, aimed at solely minimizing the power consumption. As future steps, HSLA is planned to be installed in a real SDN controller to evaluate its impact on the network performance in a working test-bed.

VI. Acknowledgements

This work has been partially funded by the project RTI2018-094591-B-I00 (MCI/AEI/FEDER,UE), 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 and Infrastructure of the Government of Extremadura (GR18112, IB18030), and by the European Regional Development Fund.

References

[1] Cisco, “Cisco Annual Internet Report (2018–2023),” March 2020, https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.pdf.

[2] R. Etengu, S. C. Tan, L. C. Kwang, F. M. Abbou, and T. C. Chuah, “Ai-assisted framework for green-routing and load balancing in hybrid software-defined networking: Proposal, challenges and future perspective,” *IEEE Access*, vol. 8, pp. 166 384–166 441, 2020.

[3] J. L. Herrera, M. Polverini, and J. Galán-Jiménez, “A machine learning-based framework to estimate the lifetime of network line cards,” in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, 2020, pp. 1–5.

[4] P. E. Heegaard and K. S. Trivedi, "Network survivability modeling," *Comput. Netw.*, vol. 53, no. 8, p. 1215–1234, Jun. 2009. [Online]. Available: https://doi.org/10.1016/j.comnet.2009.02.014

[5] R. Bolla, R. Bruschi, F. Davoli, and F. Cucchietti, "Energy efficiency in the future internet: A survey of existing approaches and trends in energy-aware fixed network infrastructures," *IEEE Communications Surveys Tutorials*, vol. 13, no. 2, pp. 223–244, 2011.

[6] L. Chiaraviglio, M. Mellia, and F. Neri, "Reducing power consumption in backbone networks," in *2009 IEEE International Conference on Communications*, 2009, pp. 1–6.

[7] C. Phillips, "Pro-active energy management for wide area networks," *IET Conference Proceedings*, pp. 317–322(5), January 2011.

[8] J. Galán-Jiménez, M. Polverini, and A. Cianfrani, "Reducing the reconfiguration cost of flow tables in energy-efficient software-defined networks," *Computer Communications*, vol. 128, pp. 95–105, 2018.

[9] 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.

[10] S. Agarwal, M. Kodialam, and T. V. Lakshman, "Traffic engineering in software defined networks," in *2013 Proceedings IEEE INFOCOM*, 2013, pp. 2211–2219.

[11] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in sdn-openflow networks," *Computer Networks*, vol. 71, pp. 1–30, 2014.

[12] A. Mendiola, J. Astorga, E. Jacob, and M. Higuero, "A survey on the contributions of software-defined networking to traffic engineering," *IEEE Communications Surveys Tutorials*, vol. 19, no. 2, pp. 918–953, 2017.

[13] D. K. Hong, Y. Ma, S. Banerjee, and Z. M. Mao, "Incremental deployment of sdn in hybrid enterprise and isp networks," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '16. New York, NY, USA: Association for Computing Machinery, 2016.

[14] X. Huang, S. Cheng, K. Cao, P. Cong, T. Wei, and S. Hu, "A survey of deployment solutions and optimization strategies for hybrid sdn networks," *IEEE Communications Surveys Tutorials*, vol. 21, no. 2, pp. 1483–1507, 2019.

[15] R. Amin, M. Reisslein, and N. Shah, "Hybrid sdn networks: A survey of existing approaches," *IEEE Communications Surveys Tutorials*, vol. 20, no. 4, pp. 3259–3306, 2018.

[16] J. Galán-Jiménez, "Legacy ip-upgraded sdn nodes tradeoff in energy-efficient hybrid ip/sdn networks," *Computer Communications*, vol. 114, pp. 106–123, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S014036641630514X

[17] J. Galán-Jiménez, "Minimization of energy consumption in ip/sdn hybrid networks using genetic algorithms," in *2017 Sustainable Internet and ICT for Sustainability (SustainIT)*, 2017, pp. 1–5.

[18] Y. Wei, X. Zhang, L. Xie, and S. Leng, "Energy-aware traffic engineering in hybrid sdn/ip backbone networks," *Journal of Communications and Networks*, vol. 18, no. 4, pp. 559–566, 2016.

[19] N. Huin, M. Rifai, F. Giroire, D. Lopez Pacheco, G. Urvoy-Keller, and J. Moulierac, "Bringing energy aware routing closer to reality with sdn hybrid networks," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, 2017, pp. 1–7.

[20] N. Saha, S. BERA, and S. Misra, "Sway: Traffic-aware qos routing in software-defined iot," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 1, pp. 390–401, 2021.

[21] C. Hopps, "Analysis of an equal-cost multi-path algorithm," *RFC*, vol. 2992, pp. 1–8, 2000.

[22] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Communications Surveys Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

[23] T. Lin, C. Kuo, H. Chang, W. Chang, and Y. Lin, "A parameterized wildcard method based on sdn for server load balancing," in *2016 International Conference on Networking and Network Applications (NaNA)*, 2016, pp. 383–386.

[24] A. Cianfrani, V. Eramo, M. Listanti, M. Polverini, and A. V. Vasilakos, "An ospf-integrated routing strategy for qos-aware energy saving in ip backbone networks," *IEEE Transactions on Network and Service Management*, vol. 9, no. 3, pp. 254–267, 2012.

[25] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly, "SNDlib 1.0–Survivable Network Design Library," in *Proceedings of the 3rd International Network Optimization Conference (INOC 2007), Spa, Belgium*, April 2007, http://sndlib.zib.de, extended version accepted in Networks, 2009.

[26] J. Galan-Jimenez and A. Gazo-Cervero, "Elee: Energy levels-energy efficiency tradeoff in wired communication networks," *IEEE Communications Letters*, vol. 17, no. 1, pp. 166–168, 2013.

[27] Y. Guo, Z. Wang, Z. Liu, X. Yin, X. Shi, J. Wu, Y. Xu, and H. J. Chao, "Sote: Traffic engineering in hybrid software defined networks," *Computer Networks*, vol. 154, pp. 60–72, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128618302603

[28] R. Etengu, S. Tan, F. Abbou, C. Lee, Z. Yusoff, and M. Shahbe, "Hybrid software-defined networking traffic scheduling: Energy-aware load balancing perspective," in *Conference of Open Innovations Association, FRUCT*, no. 25. FRUCT Oy, 2019, pp. 452–457.

Appendix N

Quality of Service-Adaptive Industrial Internet of Things Leveraging Edge Computing and Deep Reinforcement Learning: The Deep QoS-Adaptive Learning Environment (DeQALE) Architecture

ID : **CISTI 2022**

Authors : Juan Luis Herrera, Javier Berrocal, Jaime Galán-Jiménez, José García-Alonso, Juan Manuel Murillo

Publication type : Conference full-paper

Conference : IEEE Iberian Conference on Information Systems and Technologies (CISTI) 2022

DOI : `https://doi.org/10.23919/CISTI54924.2022.9820397`

Conference rating : International conference

Quality of Service-Adaptive Industrial Internet of Things leveraging Edge Computing and Deep Reinforcement Learning

The Deep QoS-Adaptive Learning Environment (DeQALE) Architecture

Juan Luis Herrera, Javier Berrocal, Jaime Galán-Jiménez, José García-Alonso and Juan Manuel Murillo
Department of Computer Systems and Telematics Engineering
University of Extremadura
Cáceres, Spain
[jlherrerag, jberolm, jaime, jgaralo, juanmamu]@unex.es

Abstract — **The advent of the Internet of Things (IoT) paradigm allows for real-world tasks to be monitorized and managed using computing applications. The application of IoT to the industrial environment leads to the Industrial Internet of Things (IIoT), in which industrial processes are managed through IoT, thus allowing industry workers to better control their facilities and processes. However, IIoT applications have very strict Quality of Service (QoS) requirements, such as short response times, that require for the deployment of their services in edge nodes, close to the facilities. In IIoT scenarios, deploying each of the services so that the QoS requirements are met is not an easy task. Moreover, the dynamicity of the environment requires for a fast, adaptive solution. In this position paper, we propose DeQALE, an approach to train a deep reinforcement learning agent to solve this problem in short cycles.**

Keywords – Quality of Service; Industry 4.0; Internet of Things, Deep Reinforcement Learning; Edge computing

I. Introduction

In recent years, the number of Internet-connected devices has grown exponentially, as it is expected to reach over three times the global population by 2023 [1]. This growth has been especially motivated by the Internet of Things (IoT) paradigm, which allows computer applications to interact with the real world. Through the use of IoT, it is possible to bring the benefits of computing and telecommunications to real-world processes, sending, processing and receiving data making use of the Internet. Such potential is especially interesting in the industrial domain, in which the monitoring and management of information related to critical processes can be crucial [2]. Hence, the term Industrial Internet of Things (IIoT) refers to the application of the IoT paradigm to industry. IIoT is considered one of the key enabling technologies for Industry 4.0, the next step in industrialization, bringing computerization and automation to industrial processes [3].

Nonetheless, IIoT devices rarely have the power to run a complete IIoT application themselves, and instead, they rely on their connections to offload the computational workload to other machines, while they serve as a gateway to the real world [4]. Traditionally, the applications were deployed to the cloud, a set of vastly powerful computers in the core of the network. However, IIoT applications usually require a high Quality of Service (QoS), such as very short cycle times [2]. Hence, the latency to the cloud may be unsuitable for IIoT applications. Instead, these applications can be deployed to the edge computing environment, in which the computation is offloaded to multiple devices within the close vicinity to the IIoT devices themselves [5].

Moreover, IIoT applications are generally have a Service-Oriented Architecture (SOA) [6]. SOAs are composed of multiple, loosely-coupled services that perform different tasks, enabling for high modularity and evolvability [6]. However, this modular nature, together with the multiple offloading points provided by edge computing, complicate the management of the services themselves, as their placement affects the application's QoS [4]. Thus, while it is desirable to place the services in a manner that optimizes QoS, the assessment of this placement is not trivial in an edge computing environment. Furthermore, Industry 4.0's focus on computerization motivates the development of solutions that adapt automatically, rather than requiring workers to manually adapt the IIoT systems to the environment [3]. This is especially important in a dynamic environment such as the industrial one, in which adapting the service placement to the current situation should be performed in a fast, and preferably automatic, manner [2]. As such, the development of software solutions that automatically assess the placement of services can help operators meet the QoS requirements of IIoT applications.

Due to this fast requirement, techniques with a high computational complexity, such as backtracking or mathematical programming, are unfit for this task. Thus, Deep Reinforcement Learning (DRL) is a promising technique to apply to service placement assessment [7]. The objective of DRL is to train an *agent* to perform a given set of *actions* in an *environment* through the training of a deep neural network [8]. This training is performed by assessing the optimality of the actions taken by the agent and giving it an according *reward*, which is used to tune the weights in the neural network [9]. However, while the inference process of a DRL agent is fast enough for dynamic adaptation, its training process can take a large amount of time [10]. Moreover, a digital environment needs to exist for the agent's training process, as pushing an untrained agent to production may result in malfunctions.

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 (GR21133, IB18030), and by the European Regional Development Fund.

In this position paper, we propose the Deep QoS-Adaptive Learning Environment (DeQALE), a conceptual architecture to enable the use of DRL agents to assess QoS-adaptive service placements, providing industrial facilities an intelligent technology and enabling for the use of IIoT. Specifically, the contributions of this position paper are the proposal of a conceptual architecture for the training and use of DRL agents in IIoT service placement, and the proposal of a model for the environment of DRL-based, QoS-adaptive service placement agents.

The remainder of this paper is structured as follows: section II details DeQALE's conceptual architecture, section III proposes the model for DeQALE's DRL system, and section IV concludes the paper.

II. The DeQALE Architecture

The DeQALE conceptual architecture addresses some of the problems that arise due to the time-consuming task of training DRL agents for service placement. These problems appear at two key points of an agent's lifecycle: its inception, and its use in production environments. In the inception phase, there is no prior DRL agent in place, and thus, one is needed. However, the use of an untrained agent at this phase, allowing it to control the real industrial environment to learn, can lead to catastrophic results, especially considering the amount of time required to properly train a DRL agent. On the other hand, the agent requires an environment to train, which needs to be realistic for it to learn correctly. Moreover, once the DRL is in production, it would be desirable for the agent to continuously learn at production, further enhancing its performance. However, an agent trained with more data may not necessarily lead to a better result [8]. DeQALE provides a conceptual architecture, depicted in Fig. 1, that addresses these concerns.

A. *Training the DRL Agent: the Simulated Environment*

First, to solve the problem at the inception phase, DeQALE provides a simulated environment for the DRL agent to train. This simulated environment can be constructed using estimations about the IIoT application, its services, the application's QoS requirements, the IIoT and edge computing infrastructures, its hardware resources, and their performance metrics. Since they are estimations, and not necessarily real information, these metrics can be obtained at the late design phase of the application's development. Then, using simulators such as iFogSim [11], or even emulators such as Netkit [12], it is possible to construct a realistic simulation based on these estimations. DeQALE allows the DRL agent to control the simulation by migrating the services between nodes, effectively creating a training environment for the DRL agent. Due to the fact that this environment can be created at early stages of the IIoT application's development, the time required at the training phase is not an issue, and the DRL agent can be integrated once the application enters the deployment phase.

B. *Continuously Enhancing the DRL Agent*

However, as the initial DRL agent has been trained with estimations, while its initial performance may be good, it would be desirable to further train and enhance its performance with real data. Thus, in the same manner monitoring is leveraged to detect any events that may trigger the DRL agent, DeQALE uses the information obtained through monitoring to feed the simulated environment with real data, adjusting the simulation to be as realistic as possible. Nonetheless, the agent that is used at production is not directly trained. Instead, the agent is cloned, and a clone of the production agent is the one trained at the enhanced simulation. Furthermore, an evaluator module is on charge of assessing the performance of both, the trained agent in production and the training agent in the simulation, using the Key Performance Indicators (KPIs) selected by DeQALE's operator (e.g., response time, energy consumption). Whenever the KPIs of the training agent indicate it is outperforming the agent in production (i.e., it is enhanced due to the additional training), the training agent is cloned, and this clone substitutes the agent in production. Thus, the agent can improve constantly, and each of these improvements are continuously reflected in production.

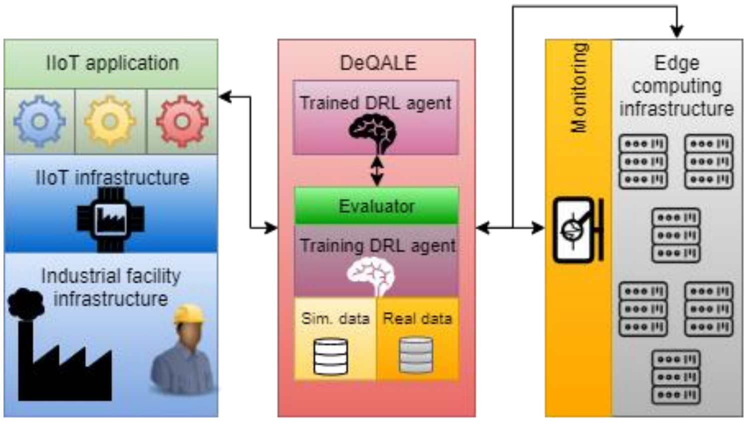


Figure 1. DeQALE's architecture and role in the IIoT environment.

C. *DeQALE's Role in the IIoT Environment*

Within the IIoT environment, DeQALE acts as an intermediary between the IIoT application and the edge computing infrastructure. Making use of the information provided through the monitoring of the edge infrastructure, DeQALE places the services from the IIoT application into its edge nodes, while performing its continuous dual training-inference approach to enhance the DRL agent.

III. The DeQALE Environment Reference Model

In order to enable the use of DeQALE, there is a need to create a DRL environment in which the agents can train. In this section, we present DeQALE's DRL environment reference model, which can be used for this purpose. Nonetheless, as a conceptual architecture, DeQALE is not necessarily bound to a concrete DRL environment for QoS-adaptive service placement, and environments that do not follow the reference model can be used as well. In DRL, the environment is usually modeled as a Markov Decision Process (MDP) [7], and comprises three main elements: the state space, the action space, and the reward system [8]. The agent observes the current state, one of the nodes of the MDP, which belongs to the state space, and takes a suitable action from the action space, i.e., an edge from the MDP. This action may alter the state of the system, as the edge of the MDP may lead to another node. As a result of taking that action, the agent is given a reward, as its objective is to maximize it through their actions. This process is repeated until the agent reaches the terminal state of the MDP. Furthermore, as the agent also needs to control real service placement, we also introduce the integration with this sort of control as a fourth element. The complete reference model, including all its elements and interactions, can be seen in Fig. 2.

elements in the vector take the value 0 are thus considered terminal. For the observation space, along with the full X_S vector, the DRL agent should be provided with additional information about both common limitations (e.g., hardware consumption of services and hardware resources available at each node), and KPI-specific information (e.g., latency to each node, energy consumption of each service).

B. *Action Space*

A DeQALE agent is expected to place a single service on each action, i.e., each episode will always take S actions to complete. Therefore, the action space of the DeQALE agent is E, as it simply has to choose an edge node per action. Once the action is taken, the value it selects will be the edge node in which the service s is placed, where the agent is performing its s_{th} action in the episode.

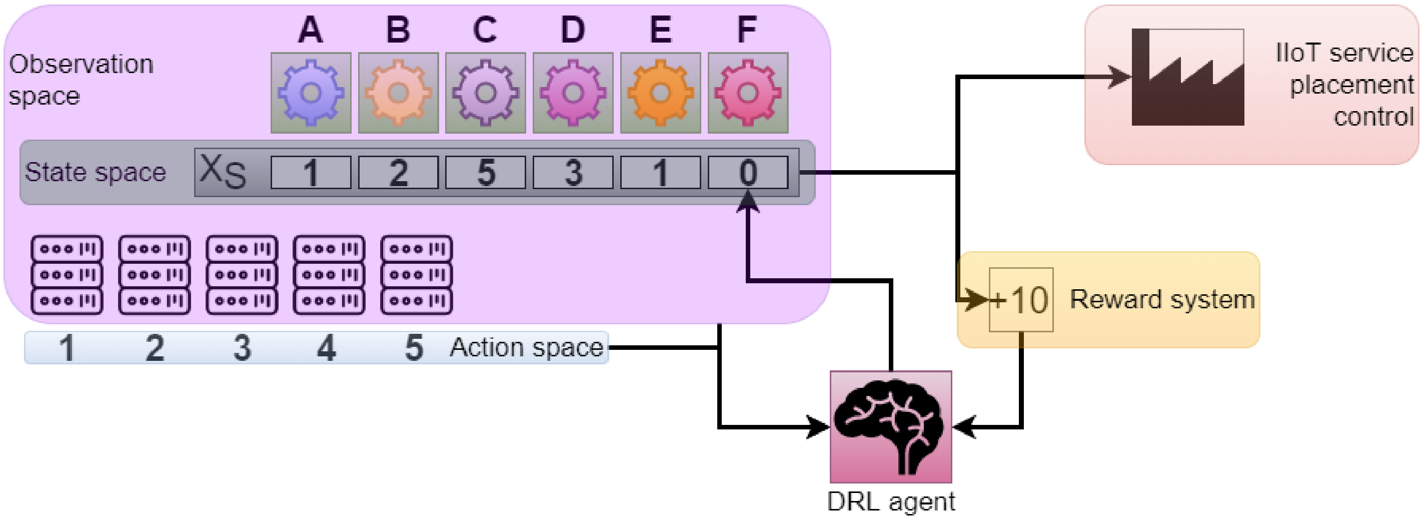


Figure 2. DeQALE’s environment reference model.

A. *State Space*

In the DeQALE reference model, we consider the state space to be bound to another entity, the *observation space*. The state space represents the possible states in which the environment can be, while the observation space represents all the information that the DRL agent can *sense* of a given state [8]. This allows for the state space to have a compact representation, which can be expanded in the observation space to give additional information to the agent. For the state space, in an environment with S services and E edge nodes, each of edge nodes can host zero or more services, while each service can be deployed to one edge node. Thus, a state in DeQALE is an integer vector X_S, where x_s takes the value e, and e is the edge node the service was deployed to (e.g., edge node 2), and 0 if it is yet to be deployed. Therefore, the complete state space can be represented as X_S, i.e., all the possible combinations of vectors in which each element takes values between 0 and E. All the states in which none of the

C. *Reward System*

The objective of an agent trained by DeQALE is twofold. On the one hand, it must place the services in a valid manner, as edge nodes have certain limitations (e.g., RAM, number of CPU cores, storage) that cannot be surpassed by the consumption of the services placed in them. On the other hand, it must place the services as optimally as possible w.r.t. the KPIs defined for the IIoT application. To perform both of these actions, we propose to punish (i.e., provide a negative reward) any invalid solutions provided by the agent, so that it learns to place services in a valid manner [8]. Once the deployment is valid, the reward should be calculated according to the KPIs that are defined for the application. Namely, those KPIs that should be minimized (e.g., energy consumption) should provide either an inversely proportional or negative reward, while KPIs to maximize (e.g., reliability) should provide directly proportional and positive reward [8]. Furthermore, if multiple KPIs are optimized at the same time, all of them should to be normalized to maintain a similar r

D. Service Placement Control

Finally, the integration of DeQALE with the simulated and real infrastructure and application must be done in both ways, from DeQALE to the infrastructure and vice-versa. On the one hand, the decisions taken by DeQALE's DRL agents in the environment must also be performed in the infrastructure. On the other hand, the DeQALE agent needs to receive updated information from monitoring to adapt the deployment to the changes that may appear. To perform this communication, we propose the monitoring of both, the simulated and the real infrastructure, to be performed in a *push* scheme by calling the API of the environment. Thus, the monitoring system can directly provide the agent with updated information, request for placement adaptations, and retrieve the results after an episode. Furthermore, DeQALE's environment can directly push the updated information to the agent by simply updating the observation space and the reward that the agent is awarded.

IV. Conclusions and Future work

The introduction of Industry 4.0 is turning industry to a cyber-physical system model, in which technology becomes the wrapper for automation and management of industrial assets. IIoT, allowing for the computerized management of industrial processes, is one of the enabling technologies of this future. However, the high QoS requirements of IIoT applications motivate the use of edge computing, which in turn complicate the placement of services throughout the infrastructure. Moreover, the high dynamicity of the industrial environment requires for fast and intelligent service placement solutions. In this position paper, we have introduced DeQALE as a solution to make use of DRL agents for QoS-adaptive service placement in IIoT environments.

As future work, we expect to evaluate the performance of DeQALE in a simulated or emulated industrial testbed, as well as comparing its performance to other approaches to this adaptation problem.

References

[1] Cisco, 'Cisco Annual Internet Report (2018–2023)', *Cisco*, pp. 1–41, 2020.

[2] 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. Institute of Electrical and Electronics Engineers Inc., pp. 78238–78259, 2018, doi: 10.1109/ACCESS.2018.2884906.

[3] P. K. R. Maddikunta *et al.*, 'Industry 5.0: A survey on enabling technologies and potential applications', *J. Ind. Inf. Integr.*, p. 100257, Aug. 2021, doi: 10.1016/J.JII.2021.100257.

[4] A. Brogi, S. Forti, C. Guerrero, and I. Lera, 'How to place your apps in the fog: State of the art and open challenges', *Softw. Pract. Exp.*, vol. 50, no. 5, pp. 719–740, May 2020, doi: 10.1002/spe.2766.

[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 Mob. Comput.*, vol. 52, pp. 71–99, 2019, doi: https://doi.org/10.1016/j.pmcj.2018.12.007.

[6] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, 'Service-oriented computing: State of the art and research challenges', *Computer (Long. Beach. Calif).*, vol. 40, no. 11, pp. 38–45, Nov. 2007, doi: 10.1109/MC.2007.400.

[7] Y. Hao, M. Chen, H. Gharavi, Y. Zhang, and K. Hwang, 'Deep Reinforcement Learning for Edge Service Placement in Softwarized Industrial Cyber-Physical System', *IEEE Trans. Ind. Informatics*, vol. 17, no. 8, pp. 5552–5561, Aug. 2021, doi: 10.1109/TII.2020.3041713.

[8] C. Lei, 'Deep Reinforcement Learning', pp. 217–243, 2021, doi: 10.1007/978-981-16-2233-5_10.

[9] V. Francois-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, 'An Introduction to Deep Reinforcement Learning', *Found. Trends Mach. Learn.*, vol. 11, no. 3–4, pp. 219–354, Nov. 2018, doi: 10.1561/2200000071.

[10] G. Dulac-Arnold *et al.*, 'Deep Reinforcement Learning in Large Discrete Action Spaces', Dec. 2015.

[11] R. Mahmud *et al.*, 'IFogSim2: An Extended iFogSim Simulator for Mobility, Clustering, and Microservice Management in Edge and Fog Computing Environments', Sep. 2021.

[12] M. Pizzonia and M. Rimondini, 'Netkit: Easy Emulation of Complex Networks on Inexpensive Hardware', 2008.

Appendix O

Despliegue Óptimo de Aplicaciones IoT Distribuidas

ID	: **JISBD 2021**
Authors	: Juan Luis Herrera, Jaime Galán-Jiménez, José García-Alonso, Javier Berrocal, Juan Manuel Murillo
Publication type	: Conference full-paper
Conference	: Jornadas de Ingeniería del Software y Bases de Datos (JISBD) 20/21
Handle	: `http://hdl.handle.net/11705/JISBD/2021/024`
Conference rating	: National conference

Despliegue óptimo multidimensional de aplicaciones IoT distribuidas[*]

Juan Luis Herrera, Jaime Galán-Jiménez, José García-Alonso, Javier Berrocal, and Juan M. Murillo

Departamento de Ingeniería de Sistemas Informáticos y Telemáticos, Universidad de Extremadura, España `[jlherrerag, jaime, jgaralo, jberolm, juanmamu]@unex.es`

Resumen La aparición del Internet de las cosas (IoT) ha atraído el interés de industria y academia para su aplicación en dominios intensivos, como la salud. Esta clase de aplicaciones tienen requisitos estrictos de calidad de servicio (QoS), lo que motiva el uso de paradigmas como *edge* o *fog* computing. Las redes definidas por sofware, junto a las arquitecturas de microservicios, permiten el uso de dichos paradigmas proveyendo virtualización, flexibilidad y programabilidad a las aplicaciones IoT distribuidas. Sin embargo, para cumplir los estrictos requisitos de estas aplicaciones, la QoS debe optimizarse considerando la interacción de tres dimensiones: computación, red y aplicación. En este trabajo presentamos el framework Despliegue Óptimo de Aplicaciones Distribuidas, que optimiza la localización de microservicios y recursos de red en términos de tiempo de respuesta y coste del despliegue.

Keywords: Internet of Things, Software-Defined Networks, Edge Computing, Fog Computing, Quality of Service

1. Introducción

El incremento en el número de dispositivos conectados a Internet en los últimos años, provocado en especial por la llegada del paradigma del Internet de las cosas (IoT), ha generado un crecimiento exponencial de la cantidad de tráfico que fluye por la red. En particular, se espera que el número de dispositivos conectados triplique la población mundial en 2023, siendo la mitad de estas conexiones de máquina a máquina (M2M) [6].

Las tareas de IoT, así como las tareas de procesamiento de datos relacionadas, pueden llegar a ser computacionalmente pesadas y, por ende, los limitados recursos de los dispositivos IoT ordinarios (memoria, batería, etc.) suelen no ser suficientes para su ejecución [3]. Para superar esta dificultad, estas tareas suelen

[*] Este trabajo ha sido parcialmente financiado por el proyecto RTI2018-094591-B-I00 (MCI/AEI/FEDER,UE), el proyecto 4IE+ (0499-4IE-PLUS-4-E) financiado por el programa Interreg V-A España-Portugal (POCTEP) 2014-2020, por el Departamento de Economía e Infraestructura de la Junta de Extremadura (GR18112, IB18030), por la Fundación Valhondo Calaff, y por el Fondo Europeo de Desarrollo Regional.

realizarse en el cloud, donde se pueden ejecutar sin comprometer estos recursos. Sin embargo, el uso del paradigma cloud impone un retraso en el tiempo de respuesta debido a la latencia entre los dispositivos finales y los servidores cloud.

Mientras que esta técnica se puede utilizar fácilmente con aplicaciones que no tienen requisitos estrictos de calidad de servicio (QoS), existen aplicaciones emergentes muy sensibles al retardo (realidad aumentada y mixta, conducción autónoma, etc.). Estas requieren de grandes anchos de banda y muy bajas latencias, que no son factibles si se usa un paradigma cloud puro. Por ello, los paradigmas *edge* y *fog computing* suponen una solución adecuada cuando existen estos requisitos [3]. Al mover los servicios del cloud al borde de la red, se obtiene una reducción de la latencia y una menor carga de tareas en los dispositivos finales, que solicitan dichas tareas a los servidores edge [3].

En la actualidad, se está tratando de integrar computación y red para mejorar los tiempos de respuesta y el uso de recursos, formando el llamado *continuo edge-cloud* [8]. Para realizar esta integración, se pueden acoplar recursos de computación (servidores) a equipos de red (switches, routers, etc.), a medio camino entre el cloud y el edge, de modo que se reduzca el tiempo de respuesta cuando sea necesario. Diversos paradigmas, como las redes definidas por software (SDN) o las arquitecturas de microservicios, facilitan esta tarea, ya que proporcionan características como virtualización o programabilidad a la red. Gracias a esto, por ejemplo, es posible realizar tareas como descubrimiento o composición de servicios directamente en la red, de manera transparente [2]. Estas tareas son interesantes en las aplicaciones sensibles al retardo actuales, puesto que permiten desacoplar infraestructura y aplicación, encargando las tareas correspondientes a la red [15]. Sin embargo, la QoS que experimentan las aplicaciones en arquitecturas de este tipo depende del rendimiento de los recursos de computación y red. A su vez, este rendimiento depende de la colocación de microservicios y controladores SDN en la infraestructura [3,7]. Por tanto, para aprovechar los beneficios del continuo edge-cloud, se deben abordar con especial atención los problemas de colocar tanto los microservicios de la aplicación como los controladores SDN.

En este artículo, presentamos Despliegue Óptimo de Aplicaciones Distribuidas (DADO), un framework para optimizar la colocación de microservicios y recursos de red en el continuo edge-cloud. Concretamente, la solución propuesta obtiene: i) la localización óptima de los microservicios que se ejecutarán en los servidores de la infraestructura, y ii) la localización del controlador o controladores SDN que maximiza la QoS de aquellas tareas asignadas a la red (descubrimiento de servicios, composición, etc.). Este framework se evalúa sobre un escenario de aplicaciones IoT sensibles al retardo considerando dos métricas: i) la minimización del tiempo de respuesta medio, y ii) la minimización de los costes derivados del despliegue de la infraestructura. Por último, exponemos los beneficios de considerar conjuntamente las dimensiones de computación y red para gestionar las aplicaciones IoT actuales y sus estrictos requisitos de QoS.

El resto de este artículo está organizado de la siguiente manera. La sección 2 describe la arquitectura considerada y los paradigmas que la habilitan. La sección 3 presenta el framework propuesto para optimizar el despliegue de aplicaciones. La sección 4 presenta la evaluación del rendimiento del framework en un escenario relacionado con la salud. La sección 5 presenta los trabajos relacionados con DADO. Finalmente, la sección 6 concluye el artículo.

2. Arquitectura de despliegue de aplicaciones IoT jerárquica

El desarrollo de aplicaciones IoT con requisitos estrictos de QoS requiere de la coordinación de tres dimensiones altamente relacionadas: la dimensión de aplicación, que indica cómo esta modularizada y coordinada la aplicación; la dimensión de computación, que define los servidores disponibles y cómo se despliegan los módulos de aplicación en ellos; y la dimensión de red, que comprende aquellos elementos de la infraestructura que permiten la comunicación entre módulos de aplicación. Todas estas dimensiones deben visualizarse de manera holística, puesto que están altamente relacionadas, ya que la configuración de cada dimensión afecta al resto y, por tanto, a la QoS obtenida.

2.1. Dimensión de aplicación

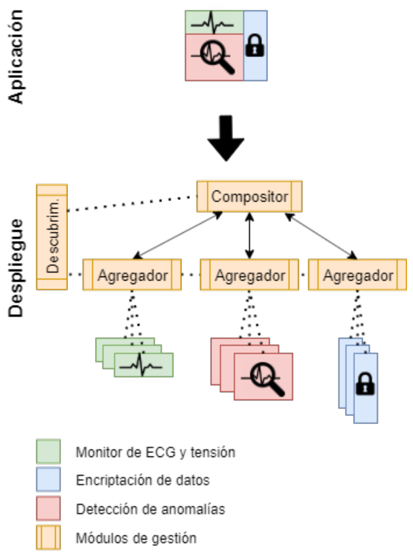


Figura 1: Dimensión de aplicación.

Las aplicaciones IoT suelen estar basadas en el paradigma de computación orientada a servicios (SOC), ya que deben ser interoperables, altamente evolucionables y masivamente distribuidas [17]. El patrón de arquitecturas orientadas a microservicios (MSA) permite dividir las aplicaciones en módulos poco acoplados que colaboran entre sí. Estos módulos, llamados habitualmente contextos acotados o servicios, contienen uno o más microservicios altamente acoplados, que normalmente se despliegan de forma conjunta [14].

Para ejemplificar esta arquitectura, nos basamos en una aplicación del Internet de las cosas médico (IoMT) que permite monitorizar la tensión de un paciente y detectar anomalías en su electrocardiograma (ECG) mediante los datos obtenidos por sensores. Este ejemplo se basa en la arquitectura propuesta en [11] y las características de entradas, salidas y procesamiento obtenidas por [16]. En él, cada usuario está equipado con un dispositivo IoT que obtiene muestras de los sensores durante 15 segundos, tras lo cual se envía para ser procesado. Las funcionalidades que se utilizan en este caso de estudio se dividen en tres servicios: monitorización de ECG y tensión (verde), encriptación de datos (azul) y detección de anomalías (rojo). La Figura 1 muestra un diseño arquitectónico a alto nivel de esta aplicación.

Estos servicios pueden desplegarse de manera independiente, de modo que pueden desplegarse en la misma máquina, en diferentes máquinas o replicarse para equilibrar la carga y mejorar la QoS. Pueden usarse diversas técnicas de virtualización, como Kubernetes, para reducir el esfuerzo del despliegue. Además, SOC hace uso de técnicas clave para la integración de los servicios desplegados como el descubrimiento de servicios, que permite conocer la localización de cada servicio; o la composición, que permite coordinar la ejecución de funcionalidades complejas orquestando diversos servicios [17,13].

2.2. Dimensión de computación

Paradigmas como fog, edge o mist computing permiten que las aplicaciones IoT intensivas con requisitos de QoS estrictos se puedan desarrollar utilizando una arquitectura jerárquica [3]. Los servicios de aplicaciones basadas en MSA pueden desplegarse cerca de los usuarios finales, reduciendo la carga de red y el tiempo de respuesta [3]. Los servicios que más recursos requieren siguen desplegándose en nodos potentes (como servidores cloud o fog), mientras que aquellos que requieren menos recursos pueden desplegarse en nodos de red con capacidades de computación (edge) o incluso en los propios dispositivos (mist) [3]. Ergo, al desplegar estas aplicaciones, se debe evaluar una gran variedad de posibilidades, y los nodos elegidos para desplegar son clave para cumplir sus estrictos requisitos de QoS. Del mismo modo, deben coordinarse estos servicios, haciendo uso de descubrimiento de servicios, agregadores y módulos de composición, para que las funcionalidades y workflows definidos puedan ofrecerse. La localización de dichos elementos es también crucial para obtener la QoS requerida.

La Figura 2 muestra cómo se despliegan en las diferentes capas cada uno de los servicios definidos en nuestro ejemplo, despliegue que depende de la QoS que requiere cada servicio y los recursos disponibles. Por ejemplo, los monitores de ECG y tensión (verde) se despliegan en dispositivos IoT, ya que requieren menos recursos. Además, diferentes agregadores se despliegan en la capa superior (edge) para procesar las respuestas de los dispositivos IoT y reducir el tráfico. De

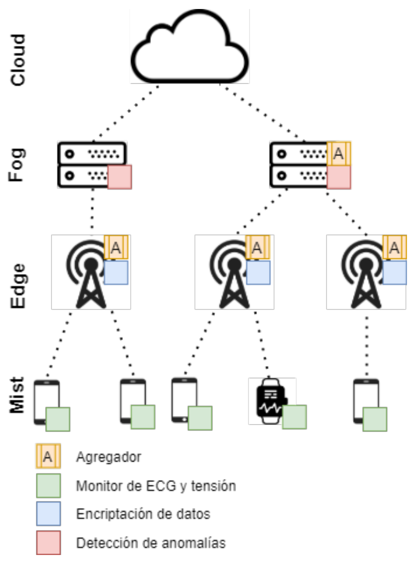


Figura 2: Dimensión de computación.

forma similar, la encriptación (azul) y detección de anomalías (rojo) requieren de recursos adicionales, por lo que se despliegan en nodos más potentes (edge y fog, respectivamente). Definir una distribución de la computación óptima depende en gran medida de los servicios definidos y los recursos disponibles.

2.3. Dimensión de red

Desplegar una aplicación IoT teniendo en cuenta exclusivamente la dimensión de computación no siempre garantiza una QoS óptima. En particular, la configuración de red, incluyendo el enrutamiento entre los servicios desplegados o el despliegue de infraestructuras específicas, tiene un gran impacto en la QoS, ya que afecta a la latencia de red. Las redes SDN permiten el despliegue de controladores SDN, que utilizan la virtualización para que la red sea programable, basándose en los principios de SOC [8]. Estas funciones de red virtuales (VNFs) pueden utilizarse para realizar ingeniería de tráfico, mejorando así el rendimiento de la red [2]. Para hacer uso de ellas, la red SDN centraliza su lógica en el *controlador SDN*, mientras que los switches pasan a ser elementos puros del plano de datos: tan solo reciben y envían mensajes de acuerdo a las normas que tienen instaladas. En caso de que no sepan cómo manejar un mensaje, lo envían al controlador SDN, que les enviará de vuelta una norma para instalar. Dado que el controlador SDN es programable, el administrador de red puede desplegar diversas VNFs en la red instalando las normas apropiadas. No obstante, el reenvío de paquetes queda detenido desde que el switch SDN envía el paquete al controlador hasta que recibe la norma correspondiente. De este modo

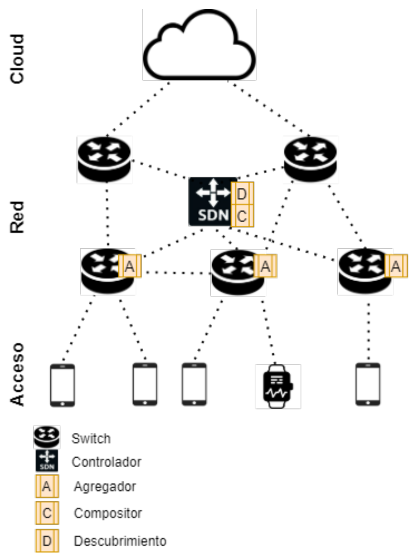


Figura 3: Dimensión de red.

el problema de colocación del controlador SDN (*Controller Placement Problem* o CPP) identifica la localización óptima para desplegar controladores SDN, de modo que se reduzca la latencia entre switches y controladores y el tiempo de respuesta [7]. En la Figura 3, se muestra el controlador SDN colocado cerca de switches específicos, entre las capas edge y fog, para mejorar la latencia.

Sin embargo, la infraestructura de red tiene también otras capacidades. La aplicación de la virtualización y los principios de SOC a la red ha dado lugar a una *cloudificación* de los controladores y switches [8]. El paradigma *edge computing* incorpora servidores en la infraestructura de red, con lo que los switches pueden desplegar funciones de computación virtuales además de VNFs [8]. Esto incluye tanto servicios IoT como otros módulos de gestión (descubrimiento de servicios, agregación, composición, etc.). Por ejemplo, ETSI propone el módulo MANO(*MANagement and Orchestration*) [10] para gestionar y orquestar los diferentes servicios y VNFs, así como encadenarlos en workflows. Por tanto, todos estos elementos de red también deben ser parte de la dimensión de computación, ya que se puedan desplegar servicios en ellos. La Figura 3 muestra que la capa edge está compuesta por tres switches SDN con capacidades de computación. Además, el descubrimiento y la composición de servicios se han desplegado en el controlador SDN, mejorando la QoS de la aplicación considerada.

Por estos motivos, una arquitectura de despliegue de aplicaciones IoT jerárquica facilita el cumplimiento de requisitos de QoS estrictos en aplicaciones IoT distribuidas. Sin embargo, para obtener un despliegue y configuración óptima, deben evaluarse tres dimensiones de manera conjunta: aplicación, computación y red. En este artículo, se toma como base una aplicación IoT que hace uso de los principios de SOC y MSA [14], y se propone un framework que coloca de manera óptima los servicios IoT y los controladores SDN en arquitecturas IoT jerárquicas.

3. Despliegue Óptimo de Aplicaciones Distribuidas

Obtener una QoS óptima requiere un despliegue que evalúe conjuntamente las dimensiones de aplicación, computación y red, colocando de forma óptima servicios IoT y controladores SDN. DADO es un framework que tiene en cuenta las tres dimensiones para encontrar despliegues factibles que cumplen los requisitos de QoS de dichas aplicaciones IoT. Aunque DADO es extensible y soporta diferentes clases de QoS, este artículo se enfoca en dos parámetros: tiempo de respuesta y coste. Estos parámetros suelen ser importantes en aplicaciones IoT y están estrechamente relacionados (una infraestructura que obtiene menores tiempos de respuesta suele tener un mayor coste), por lo que realizar un trade-off entre ambos es complejo [1]. DADO toma como entrada las características de la aplicación, la topología de red, los recursos de computación y los objetivos de QoS a satisfacer. Procesando esta información, DADO optimiza la QoS, considerando las tres dimensiones. Finalmente, se obtiene un plan de despliegue, óptimo según los objetivos seleccionados, como salida.

3.1. Entradas

Para conocer el despliegue óptimo de una aplicación, DADO requiere de diversa información de las tres dimensiones consideradas. Estas entradas se dividen en dos tipos para facilitar la reutilización y extensibilidad de DADO: información *básica*, e información *específica* de un objetivo de QoS.

La información *básica* es siempre necesaria, independientemente de los objetivos de QoS a optimizar, ya que se utiliza para conocer la factibilidad de un determinado despliegue, comprobando que no se rebasen los recursos disponibles. De la dimensión de aplicación, DADO necesita conocer la cantidad de memoria RAM consumida por cada servicio, el tamaño de sus entradas y salidas, y las peticiones procesadas por cada servicio, incluyendo qué dispositivo realiza cada petición o qué servicios se solicitan. De la dimensión de computación, es necesario indicar los recursos de computación disponibles (incluyendo su cantidad de memoria RAM disponible) y su localización en la red. Por último, de la dimensión de red, se requiere un grafo que represente la topología de red, así como la capacidad de los enlaces. Si el framework se utiliza en la fase de diseño, muchos de estos datos serán estimaciones obtenidas por los desarrolladores. Por otro lado, en tiempo de ejecución, deberán utilizarse técnicas de monitorización para obtener tanto valores reales como predicciones basadas en ellos.

La información *específica* de la QoS comprende aquellas entradas necesarias para identificar la optimalidad de un despliegue para un determinado objetivo de QoS. En su estado actual, DADO soporta como objetivos de QoS tanto tiempo de respuesta como coste del despliegue, pudiendo combinarse para buscar el trade-off óptimo entre ambos. El tiempo de respuesta se calcula como el tiempo medio necesitado para ejecutar cada petición, lo que incluye tiempo de ejecución de los servicios y tiempo de comunicación entre los dispositivos involucrados. Para evaluar este objetivo, se necesita: i) de la dimensión de aplicación, el número de ciclos que tarda en ejecutarse cada servicio; ii) de la dimensión de computación, la velocidad de reloj en ciclos de cada recurso de computación; iii) de la dimensión de red, la latencia de cada enlace. Por otra parte, el objetivo de coste se calcula como la suma de los gastos capitales (CAPEX) de cada uno de los elementos de la infraestructura y los gastos operacionales (OPEX) derivados de su uso. La información requerida de la dimensión de aplicación también son sus ciclos de ejecución, mientras que de la de computación, se necesita el OPEX por ciclo (así como el CAPEX) de cada recurso de computación. En la dimensión de red, se pide el CAPEX de cada elemento de red, así como su OPEX por segundo.

3.2. Framework DADO

DADO procesa todas las entradas definidas y obtiene un plan de despliegue óptimo según los objetivos de QoS seleccionados. Para ello, DADO tiene en cuenta como se ejecutan los servicios en los recursos de cómputo, como generan tráfico si se ejecutan en ciertos recursos, y como gestiona la red este tráfico. Siguiendo con el ejemplo de la Sección 2: dada una petición de detección de anomalías en un ECG, si ambos servicios están desplegados en la misma máquina, el tráfico se enviará del dispositivo IoMT a dicha máquina y volverá. No obstante, si cada servicio se despliega en máquinas diferentes, se generará un flujo de tráfico que comunique ambas máquinas. DADO utiliza la QoS que proporciona la red dependiendo de la localización del controlador SDN, así como la QoS que proporciona cada máquina, como criterio para seleccionar entre diversas opciones, eligiendo aquel que mejor QoS proporcione considerando las tres dimensiones. Sin embargo, un plan de despliegue óptimo puede dejar de serlo si el entorno cambia. Es por esto que el plan de despliegue debe ser evaluado y modificado continuamente, de modo que se garantice el cumplimiento de los requisitos de QoS definidos. Existen dos fases clave para encontrar el despliegue óptimo: diseño y ejecución. Por limitaciones de espacio, este artículo se centra en la fase de diseño. Durante esta fase no suele haber grandes cambios en las estimaciones de la aplicación y la infraestructura, y no existe un límite de tiempo estricto para obtener el plan de despliegue.

Para identificar el plan de despliegue óptimo en la fase de diseño, DADO formula el problema de optimización mediante programación lineal entera mixta (MILP). MILP garantiza que la solución obtenida cumple todas las restricciones definidas y es óptima considerando la función objetivo seleccionada. Para dar soporte a los tres objetivos definidos (tiempo de respuesta, coste, y ambos), se definen tres funciones objetivo. Esta formulación busca un despliegue óptimo que coloca servicios y controladores SDN, mientras que su factibilidad se evalúa con las siguientes restricciones: i) la memoria RAM disponible en los dispositivos no se sobrepasa; ii) cada flujo de tráfico, ya sea de datos o de control, tiene un único origen y un único destino, que dependen del despliegue de servicios y controladores; y iii) las capacidades de los enlaces no se sobrepasan. Aplicando la función objetivo, ya sea tiempo de respuesta, coste, o el mejor compromiso entre ambos, DADO encuentra el despliegue óptimo.

3.3. Salidas

La salida de DADO es un conjunto de colocaciones, es decir, un plan de despliegue para servicios IoT y controladores SDN. Este despliegue sigue ciertos patrones, tales como que cada servicio se desplegará en la máquina (o máquinas) que lo ejecutan, que los controladores SDN deben estar co-localizados con switches SDN [7] o que cada switch está bajo el control de un solo controlador SDN. Este plan, en la fase de diseño, es una guía para el ingeniero de operaciones, el administrador de sistemas y el administrador de red, que deberán seguirlo para localizar servicios y controladores. No obstante, en la fase de ejecución, es posible automatizar los cambios en el despliegue con herramientas de orquestación como Kubernetes. De este modo, evaluando conjuntamente las tres dimensiones, DADO obtiene el plan de despliegue óptimo para el objetivo QoS seleccionado. Además, dado su soporte para las fases de diseño y ejecución del ciclo de vida, se puede adaptar el despliegue en el tiempo para mantener una QoS óptima.

4. Evaluación de rendimiento

En esta sección, se evalúan los posibles beneficios derivados del uso del framework, así como las posibles contras a su aplicación. Primero, se describe el entorno de evaluación. Más tarde, se proponen tres conjuntos de experimentos: el primero, evalúa la escalabilidad en cuatro topologías de diferente tamaño, evaluando el segundo la complejidad computacional y el último el porcentaje de servicios desplegados en las capas fog/edge dependiendo del proveedor cloud elegido.

4.1. Entorno de evaluación

Para evaluar el rendimiento de DADO, se han utilizado cuatro escenarios basados en el ejemplo de la Sección 2. En cada uno de estos escenarios se varía el número de nodos SDN, usuarios que realizan peticiones, nodos fog capaces de desplegar servicios y nodos *gateway* que conectan la red local al cloud. Todos ellos han sido realizados mediante redes generadas con el modelo Erdös-Rényi [9], derivando en los escenarios detallados en la Tabla 1. El modelo específico de cada elemento en el escenario, así como su CAPEX y OPEX, se desglosa en [12]. Respecto a la conexión inalámbrica utilizada por los dispositivos IoMT para conectarse a la red SDN, los dispositivos Arduino utilizan Wi-Fi y Bluetooth, mientras aquellos de Texas Instruments utilizan 6LoWPAN y ZigBee. Por último, el número de servicios desplegados depende del número de peticiones y usuarios, siendo 18 en el caso con menor número de ellos y 125 en el mayor.

Cuadro 1: Parámetros de los escenarios.

Escenario	**Switches SDN**	**Usuarios**	**Nodos fog**	**Gateways**
1	7	5	1	1
2	20	15	3	2
3	50	40	10	5
4	150	50	20	10

4.2. Trade-off tiempo de respuesta-coste de despliegue

El primer análisis propuesto evalúa el rendimiento de DADO en los objetivos definidos: tiempo de respuesta y coste del despliegue. Inicialmente, cada objetivo se evalúa por separado. Más tarde, se comparan ambas métricas para encontrar un trade-off que represente el mejor compromiso entre tiempo de respuesta y coste, trade-off que DADO realiza automáticamente. Utilizando esta comparación, se puede seleccionar si se prefiere un despliegue con tiempo de respuesta mínimo, con coste mínimo, o un despliegue equilibrado que priorice ambos objetivos.

La Figura 4 muestra el resultado de este análisis, que mide el impacto de las métricas consideradas para cada uno de los cuatro escenarios. La Figura 4 tiene dos ejes y: el coste del despliegue se muestra en el eje y derecho (rojo), mientras que el izquierdo (azul) está relacionado con el tiempo de respuesta . Estos valores se presentan para tres grupos de simulaciones, con cada uno de los tres objetivos: minimizar tiempo de respuesta medio, minimizar coste, o minimizar ambos (buscando el mejor compromiso). Inspeccionando la Figura 4, queda claro que el tamaño de la red tiene un gran impacto en el coste del despliegue. Sin embargo, no hay grandes diferencias en el coste cuando se cambia el objetivo de QoS de la optimización. Por otra parte, surgen diversas consideraciones al analizar el tiempo de respuesta. Inicialmente, el tamaño de la red impacta negativamente en el tiempo de respuesta si se elige el coste como objetivo de QoS, llegando a ser hasta 16.28 veces superior en los escenarios más grandes. Sin embargo, si se selecciona el tiempo de respuesta como objetivo, las topologías grandes obtienen menores tiempos de respuesta que las pequeñas. Esto se debe a que los escenarios más pequeños disponen de menos recursos, por lo que se necesita desplegar más servicios en el cloud. Por último, si se considera el objetivo conjunto, la diferencia en el tiempo de respuesta obtenido es despreciable.

4.3. Análisis computacional

En el siguiente análisis, se utilizan datos del tiempo que DADO ha necesitado para obtener una solución. DADO está implementado utilizando Gurobi, y se ejecutó en una máquina Intel con cuatro núcleos a 2 GHz y 16 GB de RAM. La Figura 5 muestra los resultados de tiempo requerido para obtener el plan de despliegue en función del tamaño de la topología y el objetivo de QoS.

El primer aspecto a tener en cuenta es el incremento exponencial (nótese que el eje y es logarítmico) del tiempo de ejecución con relación al tamaño de la

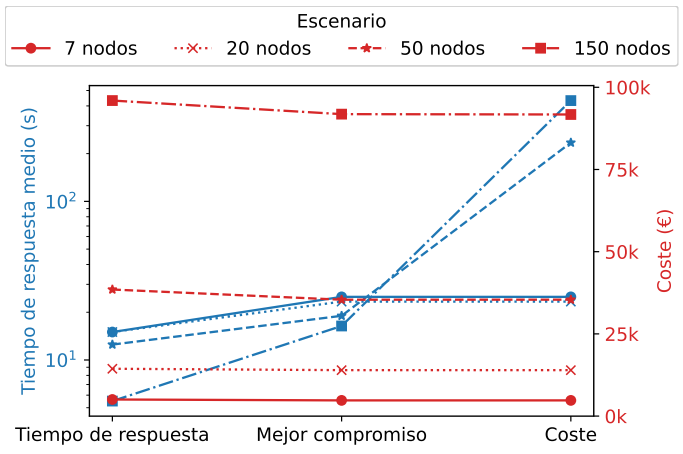


Figura 4: Trade-off de tiempo de respuesta vs. coste del despliegue.

topología. Como se esperaba, la optimización conjunta de tiempo de respuesta y coste requiere más tiempo para resolverse, entre 2 y 3,5 veces más que si se selecciona un único objetivo de QoS. Por último, la métrica que menos tarda en optimizarse es el coste del despliegue, que, de media, tarda 502,55 segundos menos que el tiempo de respuesta en obtener solución. Este análisis muestra claramente que, aunque la versión MILP de DADO es apta para tiempo de diseño, se debe utilizar una heurística en tiempo de ejecución.

4.4. Análisis de despliegue de servicios

En este último análisis, nuestro objetivo es evaluar la distribución de los servicios desplegados por DADO en las capas fog/edge (y, por extensión, cloud). La Figura 6 muestra los resultados de este análisis, en el que se analizan los porcentajes de servicios desplegados en la capa fog (siendo el porcentaje complementario aquellos en el cloud) en función del proveedor cloud y el tamaño de la topología. Se han considerado dos proveedores: Amazon Web Services (AWS) y Google Cloud Platform (GCP). Inicialmente, el porcentaje de servicios desplegados en las capas fog/edge aumenta con el tamaño de la topología: las pequeñas, con menos recursos, llevan más servicios al cloud, ya que es económico y la pérdida en rendimiento es baja. Sin embargo, el OPEX del cloud es superior al de la capa fog [12]. Por ende, al aumentar el número de servicios en las topologías mayores, el uso de fog/edge reduce tanto tiempos de respuesta como costes. Si se comparan los proveedores cloud, puede verse que DADO despliega más recursos en el cloud cuando se utiliza GCP. Esto se debe a que el ciclo de ejecución es $1{,}39 \cdot 10^{-15}$€ más barato en GCP. Aunque el ahorro por ciclo es despreciable, al extrapolarlo a la gran cantidad de ciclos de cada servicio, se vuelve significativo. En conclusión, el paradigma de fog computing es interesante tanto a nivel de tiempo de respuesta como a nivel de coste.

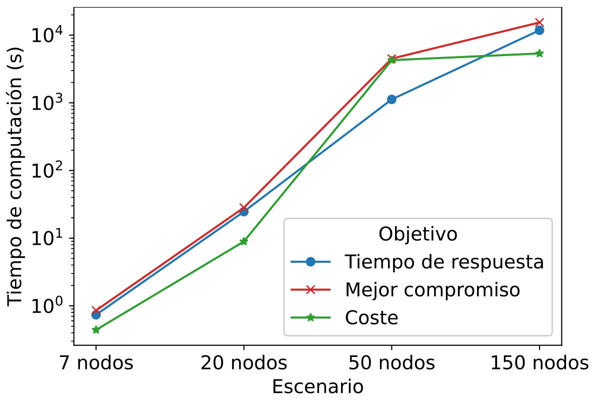


Figura 5: Tiempo de computación necesario para obtener un plan de despliegue.

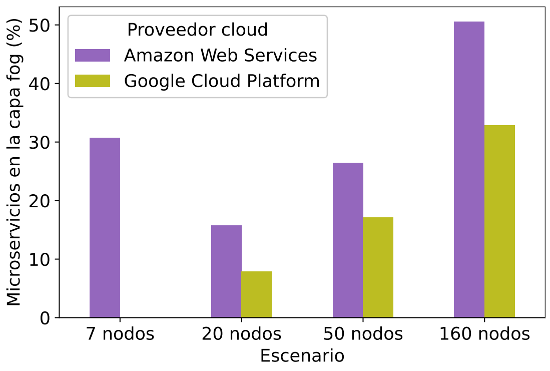


Figura 6: Porcentaje de servicios desplegados en la capa fog.

5. Trabajos relacionados

En la actualidad, y con el objetivo de acortar los tiempos de respuesta en aplicaciones de diversas clases (tales como realidad aumentada o IoT intensivo), existe un esfuerzo de investigación en paradigmas de edge y fog computing. Estos esfuerzos tratan tanto de aplicar estos paradigmas a dichas aplicaciones, como de estandarizarlos. En [3], Bellavista *et al.* hacen un repaso por diferentes paradigmas tipo edge y fog, clasificándolos en base a su utilidad en aplicaciones IoT intensivas y en la clase de soporte que proporcionan a servicios clave en dichas aplicaciones, como seguridad, virtualización o comunicación. No obstante, ninguna de estas propuestas optimiza la colocación de microservicios y controladores SDN, sino que proveen una plataforma para el despliegue de aplicaciones en estos paradigmas.

Por otro lado, la optimización del despliegue de microservicios en una arquitectura de fog computing, enfocado completamente en la dimensión de computación, es también un problema con líneas de investigación activas. No obstante, no existe un consenso respecto al nombre de este problema, denominándose problema de la distribución de la computación descentralizada, problema de la distribución de la computación, o problema de la colocación de aplicaciones en el fog, según el autor. Se han propuesto diversas soluciones a este problema: Carrega *et al.* [4] proponen una solución para aplicaciones tipo MSA que las despliega teniendo en cuenta las dependencias entre microservicios. Sun *et al.* [18] proponen hacerlo con una heurística de doble subasta, en la que los dispositivos IoT utilizan los recursos del fog en función de las necesidades de la aplicación. Choudhury *et al.* [5] lo enfocan como un problema en el que algunos servicios del cloud se traen a demanda a la red local para mejorar el rendimiento. No obstante, todas estas propuestas se diferencian de DADO en su visión: en lugar de la visión holística en tres dimensiones de DADO, estas propuestas son unidimensionales, centrándose solo en la computación o aplicación.

Por otro lado, en la dimensión de red, el CPP es un problema conocido, dado que la colocación del controlador SDN es clave para la QoS de la red y, por extensión, de todas las comunicaciones que se realizan a través de ella. Existen diversas soluciones para el CPP: tanto con uno como con varios objetivos, varios tipos de heurística, aplicando teoría de juegos, etc., siendo revisados en [7]. No obstante, de nuevo, la visión holística de DADO es clave. El CPP considera que existe un tráfico en la red, pero no considera que ese tráfico provenga de aplicaciones y, más concretamente, de su ejecución. Por tanto, DADO es de nuevo innovador, al supeditar la solución del CPP a la conveniencia para la QoS de las aplicaciones que utilizan la red, en lugar de considerar tan solo la red misma.

Aunque tanto la colocación de microservicios como la de controladores son problemas tratados en la literatura, no conocemos ningún otro trabajo que los integre en una visión holística del despliegue, que una aplicación, computación y red, además de considerar sus influencias mutuas. Por tanto, DADO aporta al integrar la optimización de las tres dimensiones, minimizando tiempo de respuesta y coste en aplicaciones IoT críticas.

6. Conclusión y trabajos futuros

El número de dispositivos IoT conectados a Internet crece cada año, y con él la demanda de aplicaciones IoT con requisitos de QoS estrictos, que difícilmente admiten un despliegue puramente cloud. Las arquitecturas IoT jerárquicas facilitan el despliegue de estas aplicaciones, pero optimizar su QoS requiere la consideración conjunta entre las dimensiones de aplicación, computación y red. En este trabajo, hemos presentado DADO, un framework capaz de desplegar de forma óptima aplicaciones IoT distribuidas en arquitecturas jerárquicas considerando las tres dimensiones conjuntamente. En el futuro, esperamos presentar el comportamiento de DADO durante la fase de ejecución, así como evaluarlo sobre infraestructuras reales o emuladas.

Referencias

1. Badawy, M.M., Ali, Z.H., Ali, H.A.: Qos provisioning framework for service-oriented internet of things (iot). Cluster Computing pp. 1–17 (2019)
2. Baktir, A.C., Ozgovde, A., Ersoy, C.: How can edge computing benefit from software-defined networking: A survey, use cases, and future directions. IEEE Communications Surveys Tutorials **19**(4), 2359–2391 (2017)
3. Bellavista, P., Berrocal, J., Corradi, A., Das, S.K., Foschini, L., Zanni, A.: A survey on fog computing for the internet of things. Pervasive and Mobile Computing **52**, 71 – 99 (2019)
4. Carrega, A., Repetto, M., Gouvas, P., Zafeiropoulos, A.: A middleware for mobile edge computing. IEEE Cloud Computing **4**(4), 26–37 (2017)
5. Choudhury, B., Choudhury, S., Dutta, A.: A Proactive Context-Aware Service Replication Scheme for Adhoc IoT Scenarios. IEEE Transactions on Network and Service Management **16**(4), 1797–1811 (2019)
6. Cisco: Cisco Annual Internet Report (2018–2023) (March 2020), https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.pdf, (visited on Jan. 29, 2021)
7. Das, T., Sridharan, V., Gurusamy, M.: A survey on controller placement in sdn. IEEE Communications Surveys & Tutorials pp. 472–503 (2019)
8. Duan, Q., Wang, S., Ansari, N.: Convergence of networking and cloud/edge computing: Status, challenges, and opportunities. IEEE Network **34**(6), 148–155 (2020)
9. Erdös, P., Rényi, A.: On random graphs i. Publ. math. debrecen **6**(290-297), 18 (1959)
10. ETSI: Open Source NFV Management and Orchestration (MANO), https://www.etsi.org/technologies/open-source-mano, (visited on Jan. 14, 2021)
11. Gia, T.N., Jiang, M., Rahmani, A.M., Westerlund, T., Liljeberg, P., Tenhunen, H.: Fog computing in healthcare internet of things: A case study on ecg feature extraction. In: 2015 IEEE CIT/IUCC/DASC/PICOM. pp. 356–363. IEEE (2015)
12. Herrera, J., Galán-Jiménez, J., Berrocal, J., Murillo, J.: Optimal deployment of distributed iot applications exploiting the coin paradigm (2021), https://tinyurl.com/dado-capex-opex, (visited on Jan. 29, 2021)
13. Huang, M., Liu, W., Wang, T., Song, H., Li, X., Liu, A.: A queuing delay utilization scheme for on-path service aggregation in services-oriented computing networks. IEEE Access **7**, 23816–23833 (2019)
14. Indrasiri, K.: Microservices in practice - key architectural concepts of an MSA, https://wso2.com/whitepapers/microservices-in-practice-key-architectural-concepts-of-an-msa/, (visited on Jan. 14, 2021)
15. Kutscher, D., Karkkainen, T., Ott, J.: Directions for Computing in the Network. Tech. rep., Internet Engineering Task Force (Jul 2020), https://datatracker.ietf.org/doc/html/draft-kutscher-coinrg-dir-02, (visited on Jan. 29, 2021)
16. Limaye, A., Adegbija, T.: A workload characterization for the internet of medical things (iomt). In: 2017 IEEE ISVLSI. pp. 302–307. IEEE (2017)
17. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-oriented computing: State of the art and research challenges. Computer **40**(11), 38–45 (2007)
18. Sun, W., Liu, J., Yue, Y., Zhang, H.: Double Auction-Based Resource Allocation for Mobile Edge Computing in Industrial Internet of Things. IEEE Transactions on Industrial Informatics **14**(10), 4692–4701 (2018)

Appendix P

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

ID	: **JCIS 2021**
Authors	: Juan Luis Herrera, Jaime Galán-Jiménez, Javier Berrocal, Juan Manuel Murillo
Publication type	: Short conference paper
Conference	: Jornadas de Ciencia e Ingeniería de Servicios (JCIS) 20/21
Handle	: `http://hdl.handle.net/11705/JCIS/2021/013`
Conference rating	: National conference

Optimizing the Response Time in SDN-Fog Environments for Time-Strict IoT Applications (Summary)[⋆]

Juan Luis Herrera, Jaime Galán-Jiménez, Javier Berrocal, and Juan M. Murillo

University of Extremadura, Spain `[jlherrerag,jaime,jberolm,juanmamu]@unex.es`

Summary of the contribution

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.

Acknowledgement

This work was supported in part by MCI/AEI/FEDER,UE under grant RTI2018-094591-B-I00; in part by the 4IE+ Project under grant 0499-4IE-PLUS-4-E funded by the Interreg V-A España-Portugal (POCTEP) 2014-2020 Program; in part by the Department of Economy, Science and Digital Agenda of the Government of Extremadura under Grant GR18112 and Grant IB18030; in part by the European Regional Developent Fund; and in part by the Valhondo Calaff institution.

[⋆] This work has been published in IEEE Internet of Things Journal, Vol 8, Issue 11, 2021. Impact Factor: 9.936. DOI: https://doi.org/10.1109/JIOT.2021.3077992

Appendix Q

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

ID	: **JITEL 2021**
Authors	: Juan Luis Herrera, Jaime Galán-Jiménez, Javier Berrocal, Juan Manuel Murillo
Publication type	: Short conference paper
Conference	: Jornadas de Ingeniería Telemática (JITEL) 2021
Conference rating	: National conference

XV Jornadas de Ingeniería Telemática.
JITEL 2021.
Universidad de A Coruña.

Actas de las XV Jornadas de Ingeniería Telemática (JITEL 2021), A Coruña (España), 27-29 de octubre de 2021.

ISBN

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

Juan Luis Herrera, Jaime Galán-Jiménez, Javier Berrocal, Juan M. Murillo
Departamento de Ingeniería de Sistenas Informáticos y Telemáticos
Universidad de Extremadura
Avda. de la Universidad, S/N, Cáceres, España
{jlherrerag, jaime, jberolm, juanmamu}@unex.es

The Internet of Things (IoT) paradigm has brought to applications the potential of automating real-world processes. Applying IoT to intensive domains comes with strict Quality of Service (QoS) requirements. To achieve such goals, a first option is to distribute the computational workload throughout the infrastructure (edge, fog, cloud). In addition, its integration with Software-Defined Networks (SDN) can even further improve the QoS experienced, thanks to the global network view of the SDN controller. Therefore, the best placement for both the computation elements and the SDN controllers must be identified to optimize QoS. To obtain a truly optimal result, it is crucial to solve the problem in a single effort. In this work, a framework to identify the optimal deployment for distributed applications, DADO, is proposed, implemented and evaluated over an IIoT case study. DADO achieves response times up to 15.42% shorter than state-of-the-art benchmarks.

***Palabras Clave*—Fog computing, Internet of Things, Software-Defined Network**

I. Summary

We live surrounded by everyday *things* that are connected to the Internet and run IoT applications – programs that interact with the real world. This interaction has generated interest in intensive domains such as industry or healthcare. However, integrating IoT applications into these domains is complex, as they have very strict QoS requirements. For this reason, cloud computing is very complicated to leverage, because it imposes a large latency penalty. New paradigms, such as fog computing, propose bringing some of the computing resources closer to the network edge, hence enhancing the QoS. Nonetheless, to properly make use of fog paradigms, it is important to optimally distribute the application's microservices among the available resources. This distribution problem is known as the Decentralized Computation Distribution Problem (DCDP). It is also important to note that the QoS of the application is determined not only by the computing QoS, but also by the networking QoS. Within the networking dimension, SDNs are also key enablers for fog paradigms because of the flexibility, scalability, and network programmability provided by the SDN paradigm, that allow for common tasks in IoT applications, such as service discovery, to be performed transparently. The QoS of SDNs, however, heavily depends on the QoS between the network's switches and the SDN controller they are assigned to. The problem in which controllers are placed and assigned optimally to SDN switches is the SDN Controller Placement Problem (CPP).

Both problems are deeply related, as the decisions on microservice placement are related to the networking QoS, and the CPP is heavily affected by the steering of application traffic flows. Thus, both problems require to be solved jointly in order to obtain optimal QoS. In this paper, we present Distributed Application Deployment Optimization (DADO), a framework based on mixed integer linear programming that jointly solves the DCDP and the CPP to minimize the response time in SDN-Fog environments. An evaluation of DADO over an industrial case study shows that this framework provides scalable solutions with response times up to 37.89% lower than alternative solutions, and up to 15.42% shorter than state-of-the-art benchmarks.

In the future, we expect to extend DADO, developing heuristics that will allow DADO to be applied to infrastructures larger than 300 nodes and while finding near-optimal solutions, as well as to add mobility considerations to DADO. Finally, we intend to expand DADO to consider multiple QoS features, such as reliability.

This work has been published in IEEE Internet of Things Journal, Vol 8, Issue 11, 2021. Impact Factor: 9.936. DOI: https://doi.org/10.1109/JIOT.2021.3077992. This work has been partially funded by the project RTI2018-094591-B-I00 (MCI/AEI/FEDER,UE), 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.

Appendix R

Joint Optimization of Response Time and Deployment Cost in Next-Gen IoT Applications (Summary)

ID	: **JCIS 2022**
Authors	: Juan Luis Herrera, Jaime Galán-Jiménez, José García-Alonso, Javier Berrocal, Juan Manuel Murillo
Publication type	: Short conference paper
Conference	: Jornadas de Ciencia e Ingeniería de Servicios (JCIS) 2022
Handle	: `http://hdl.handle.net/11705/JCIS/2022/008`
Conference rating	: National conference

Joint Optimization of Response Time and Deployment Cost in Next-Gen IoT Applications (Summary)*

Juan Luis Herrera, Jaime Galán-Jiménez, José García-Alonso, Javier Berrocal, and Juan M. Murillo

University of Extremadura, Spain
`[jlherrerag,jaime,jgaralo,jberolm,juanmamu]@unex.es`

Summary of the contribution

The irruption of the Internet of Things (IoT) has attracted the interest of both the industry and academia for their application in intensive domains, such as healthcare. The strict Quality of Service (QoS) requirements of the next generation of intensive IoT applications requires the QoS to be optimized considering the interplay of three key dimensions: computing, networking and application. This optimization requirement motivates the use of paradigms that provide virtualization, flexibility and programmability to IoT applications. In the computing dimension, paradigms such as edge or fog computing, Software-Defined Networks in the networking dimension, along with micro-services architectures for the application dimension, are suitable for QoS-strict IoT scenarios. In this work, we present a framework, named Next-gen IoT Optimization (NIoTO), that considers these three dimensions and their interplay to place micro-services and networking resources over an infrastructure, optimizing the deployment in terms of average response time and deployment cost. The evaluation of NIoTO in a healthcare case study reveals a response time speed-up of up to 5.11 and a reduction in cost of up to 9% with respect to other state-of-the-art techniques.

Keywords: Fog computing · Internet of Things (IoT) · Quality of Service (QoS)

Acknowledgement

This work has been partially funded by the project RTI2018-094591-B-I00 (MCI/AEI/FEDER,UE), 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 (GR21133, IB18030), by the Valhondo Calaff institution, and by the European Regional Development Fund.

* This work has been published in IEEE Internet of Things Journal, Special issue on Empowering the Future Generation Systems: Opportunities by the Convergence of Cloud, Edge, AI, and IoT, 2022. Impact Factor: 9.471. DOI: https://doi.org/10.1109/JIOT.2022.3165646

Appendix S

Deploying Next Generation IoT Applications Through SDN-Enabled Fog Infrastructures

ID	: **PERCOM 2022**
Authors	: Juan Luis Herrera, Javier Berrocal, Juan Manuel Murillo
Publication type	: PhD forum contribution
Conference	: IEEE International Conference on Pervasive Computing and Communications (PERCOM)
DOI	: `hhttps://doi.org/10.1109/PerComWorkshops53856.2022.9767448`
Conference rating	: Workshop associated with an international conference, GGS Class 1 (A+)

Deploying Next Generation IoT Applications Through SDN-Enabled Fog Infrastructures

Juan Luis Herrera*
Supervised by Javier Berrocal and Juan M. Murillo
*Department of Computer Systems and Telematics Engineering, University of Extremadura, Spain.
jlherrerag@unex.es

***Abstract*—The next generation of Internet of Things (IoT) applications automates critical, real-world processes from domains such as industry or healthcare. These applications have very strict Quality of Service (QoS) requirements. To meet these requirements, in recent years, the deployment of the application services can be performed not only in the cloud, but also through the fog to enhance the QoS, as well as for the use of programmable, Software-Defined Networks to optimize the QoS of their communications. However, the application and the network dimensions have been addressed separately, which leads to sub-optimal QoS in these critical applications. In this paper, we present the proposal of a framework to optimize the deployment of next-gen IoT applications through the fog that optimizes both, application and network, in a single effort.**

***Index Terms*—Fog computing, Internet of Things (IoT), Software-Defined Networking (SDN)**

I. Introduction

The advent of the Internet of Things (IoT) paradigm allows computer applications to automate and computerize real world processes. The next generation of IoT applications will apply this computerization to intensive domains, such as industry or healthcare [1]. Nonetheless, these applications require a high Quality of Service (QoS) [1], reflecting the criticality of the real-world processes they automate. These QoS requirements can span over different, or even multiple, QoS attributes, such as reliability or performance.

Traditionally, the most popular paradigm for IoT application deployment is cloud computing [2]. However, the large distance between IoT devices and cloud data centers is often reflected on its QoS (e.g., as a high latency), and hence it may be difficult to meet the QoS requirements of next-gen IoT applications in pure cloud deployments [2]. Therefore, paradigms such as fog computing, in which the application is deployed to servers closer to IoT devices, are more suitable for these applications [2]. Moreover, next-gen IoT applications are often implemented as a set of loosely-coupled *microservices* that collaborate to perform the application's functionalities [3]. Each of these microservices can be deployed independently or along with others, and it can also be replicated, allowing for the IoT application to be deployed through the complete cloud-to-thing infrastructure. Nonetheless, the placement of each microservice within the infrastructure affects the QoS of the application. The problem of placing these microservices to optimize the application's QoS is known as the Decentralized Computation Distribution Problem (DCDP) [3].

One of the main causes of the DCDP is the network fabric's QoS (e.g., latency), which directly impacts the communications between devices. Therefore, it is also desirable to optimize the network's QoS with techniques such as routing optimization [3]. Software-Defined Networking (SDN) is a paradigm that allows networks to be programmed by centralizing the control plane in the figure of SDN controllers. SDN enables, among other techniques, for the optimization of the communications' QoS through specific routing optimization. However, SDN switches must communicate with the SDN controller to retrieve their expected behavior in the form of rules [4]. These communications affect the QoS of the network fabric, but are themselves affected by the network fabric's QoS. Therefore, it is key to place SDN controllers in a manner that optimizes the network's QoS, i.e., to solve the Controller Placement Problem (CPP) [4].

Both the DCDP and the CPP are inherently related. On the one hand, the DCDP is partially caused by the QoS of the communications between microservices, which depends on where SDN controllers are placed, and thus, different controller placements may lead to different microservice distributions [3]. On the other hand, the CPP heavily depends on the sources and targets and of the traffic demands that are routed through the network [4], demands that are generated by the communications between microservices, and that may vary depending on the microservice distribution. Therefore, we conclude that both the DCDP and the CPP must be solved as part of a single optimization effort able to control all four, microservice replication, microservice placement, routing optimization and SDN controller placement. The objective of this project is the proposal of the Distributed Application Deployment Optimization (DADO) framework, able to solve both the DCDP and the CPP, optimizing the QoS of the next generation of IoT applications.

II. DADO framework

The DADO framework allows developers to optimize the QoS of their IoT applications in two key stages of their lifecycle, optimally deploying the application at design-time

This work has been partially funded by the project RTI2018-094591-B-I00 (MCI/AEI/FEDER,UE), 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 and Infrastructure of the Government of Extremadura (GR18112, IB18030), by the Valhondo Calaff institution, and by the European Regional Development Fund.

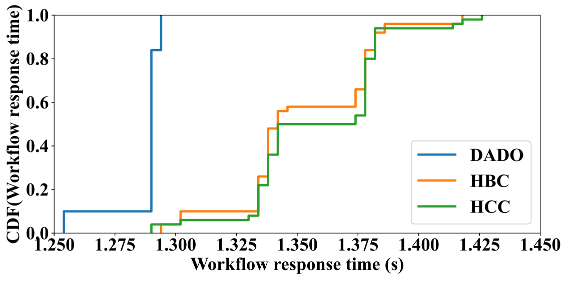


Figure 1: Empirical CDF of workflow response times.

and adapting the deployment to environmental changes at execution-time. In DADO, QoS is the *set of characteristics of a service that bear on its ability to satisfy the needs of its user* [5]. This definition includes technical and non-technical QoS, e.g., performance, availability or deployment cost. Therefore, DADO is envisioned as framework that provides an extensible model for QoS optimization. DADO provides multiple built-in QoS attributes to be optimized, and the possibility for users to define their own attributes.

DADO requires three inputs: the application architecture, the computing devices from the infrastructure and the network fabric that connects them. These inputs are plans and estimations if DADO is leveraged at design time, or their monitored versions at execution time. Along with these inputs, the developer must also specify the QoS attributes to be optimized. The output received by the developer consists on a deployment plan that optimizes the QoS.

Applications in DADO are modeled as sets of independent modules or *microservices*, that define their technical characteristics. It is important to note that each microservice should be defined only once, as DADO will automatically replicate the microservice if needed. These microservices are often not requested independently, e.g., after data has been processed by a microservice, it will probably be stored by another microservice. Therefore, the application is also defined by the interactions among microservices or *workflows*: pipelines of multiple microservices in which the output of a microservice is the input of the next one. The workflow definition includes the valid workflows for the application as well as the estimated or real number of requests per workflow, allowing DADO to adapt the application's deployment to its load.

Each of the microservices requested by these workflows and instanced by DADO needs to be deployed at a computing device (e.g., fog node, cloud server). DADO models all the computation-capable devices as a set of technical characteristics, normally analogous to those of microservices (e.g., microservices consume RAM, and devices have a total RAM). DADO does not directly model whether a device is a cloud, fog, edge or mist device, thus allowing the developer to define an infrastructure with arbitrary layers.

All these devices are connected to each other using a network fabric, modeled as a graph. Each of the vertices of the graph is either a computing device, or an SDN switch (and thus, a potential placement for an SDN controller [4]), while each edge represents a link between devices or switches. Furthermore, each of the switches and links also define their own technical characteristics (e.g., latency, capacity).

Using these inputs, DADO generates an optimal deployment plan for the application. This deployment plan details how many replicas of each microservice need to be deployed, where to deploy each of them, how many SDN controllers need to be set up, where to place each controller, which controller should each switch communicate with, and the paths followed by application and SDN control traffic. All these decisions are taken by DADO to optimize the QoS, considering the effects of each decision in the rest (e.g., deploying two microservices that communicate in different machines creates a traffic flow, which alters the optimal controller placement).

Currently, a version of DADO based on Mixed-Integer Linear Programming (MILP) is under development [3]. Fig. 1 depicts an empirical CDF of the preliminary results from this prototype version, comparing DADO with the usage of graph metrics such as Highest Closeness Centrality (HCC) or Highest Betweenness Centrality (HBC) to take the deployment decisions. As seen in Fig. 1, the slowest workflows deployed using DADO have similar response times to the fastest workflows deployed using HBC and HCC. However, MILP does not scale well on large infrastructures, taking over 10 hours to optimize the deployment plan [3].

III. Conclusions and Future Work

The next generation of the Internet of Things will computerize and automate critical real-world processes, and their criticality will be reflected as high QoS requirements for next-gen IoT applications. Meeting the QoS requirements will require the applications to be optimally deployed in a fog infrastructure, communicated by a SDN network fabric. The objective of this PhD thesis is to build DADO, a framework to optimize the application's deployment, and a possible enabler for the next generation of IoT.

DADO is currently a work in progress, and thus, there are limitations to what this paper currently presents. The current MILP-based prototype takes a long amount of time to optimize the deployment of large scenarios [3], and therefore is only useful at design time. Accordingly, we expect to develop heuristic solutions that allow for the optimization to be performed at execution time, allowing DADO to adapt the deployment to environmental changes.

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] 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.
[3] J. L. Herrera, J. Galán-Jiménez, J. Berrocal, and J. M. Murillo, "Optimizing the response time in sdn-fog environments for time-strict iot applications," *IEEE Internet of Things Journal*, 2021.
[4] T. Das, V. Sridharan, and M. Gurusamy, "A Survey on Controller Placement in SDN," *IEEE Communications Surveys & Tutorials*, pp. 472–503, 2019.
[5] ITU-T, *Definitions of terms related to quality of service*, International Telecommunication Union Telecommunication Standarization Sector Std. E.800, Rev. 09/2008.

Appendix T

DADO and NIoTO implementation

The details on the implementation of DADO and NIoTO can be found on their source code repository, including documentation, code, and other information: `https://bitbucket.org/spilab/dado-nioto`

Appendix U

FNPP implementation

The details on the implementation of the FNPP MILP solver and the FNPP heuristic solver can be found on their source code repository, including documentation, code, and other information: `https://bitbucket.org/spilab/fnpp-heuristic` and `https://bitbucket.org/spilab/fnpp-milp`

Appendix V

ConDADO implementation

The details on the implementation of ConDADO can be found on its source code repository, including documentation, code, and other information: `https://bitbucket.org/spilab/continuousdado`

Appendix W

Faustum implementation

The details on the implementation of Faustum can be found on its source code repository, including documentation, code, and other information: `https://bitbucket.org/spilab/faustum`

Appendix X

Pascal implementation

The details on the implementation of Pascal can be found on its source code repository, including documentation, code, and other information: `https://bitbucket.org/spilab/pascal`

Appendix Y

S-DADO implementation

The details on the implementation of S-DADO can be found on its source code repository, including documentation, code, and other information: `https://bitbucket.org/spilab/s-dado`

Appendix Z

Umizatou-MO-SFO implementation

The details on the implementation of Umizatou and MO-SFO can be found on their source code repository, including documentation, code, and other information: `https://bitbucket.org/spilab/umizatou-mo-sfo`

Appendix AA

μDADO implementation

The details on the implementation of μDADO can be found on its source code repository, including documentation, code, and other information: `https://bitbucket.org/spilab/udado`

Appendix AB

Vernier implementation

The details on the implementation of Vernier can be found on its source code repository, including documentation, code, and other information: `https://bitbucket.org/spilab/vernier`

Appendix AC

Glossary

AC.1 Glossary of acronyms

- **CAPEX**: Capital Expenditures
- **CD**: Continuous Deployment
- **CI**: Continuous Integration
- **CPP**: Controller Placement Problem
- **DCDP**: Decentralized Computation Distribution Problem
- **Dev**: Software developers
- **FNPP**: Fog Node Placement Problem
- **FN**: Fog Node
- **GGS**: GII-GRIN-SCIE conference rating
- **HTTP**: HyperText Transfer Protocol
- **IF**: Impact Factor
- **IIoT**: Industrial Internet of Things

- **IP**: Internet Protocol
- **IaaS**: Infrastructure as a Service
- **IoMT**: Internet of Medical Things
- **IoT**: Internet of Things
- **JCR**: Journal Citation Reports
- **JSON**: JavaScript Object Notation
- **KPI**: Key Performance Indicator
- **MILP**: Mixed-Integer Linear Programming
- **MSA**: Microservices Architecture
- **MS**: Microservice
- **NIST**: National Institute of Standards and Technology
- **OPEX**: Operational Expenditures
- **Op**: Application operators
- **PFE**: Private Function Evaluation
- **PaaS**: Platform as a Service
- **QoS**: Quality of Service
- **REST**: REpresentational State Transfer
- **RQ**: Research Question
- **SDN**: Software-Defined Networking
- **SOA**: Service-Oriented Architecture
- **SOC**: Service-Oriented Computing

- **SaaS**: Software as a Service
- **XML**: eXtensible Markup Language

AC.2 Glossary of venues

- **CCGRID**: IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing
- **CISTI**: IEEE Iberian Conference on Information Systems and Technologies
- **GLOBECOM**: IEEE Global Communications Conference
- **ICC**: IEEE International Communications Conference
- **ISCC**: IEEE Symposium on Computers and Communications
- **IoTJ**: IEEE Internet of Things Journal
- **JCIS**: Jornadas de Ciencia e Ingeniería de Servicios
- **JISBD**: Jornadas de Ingeniería del Software y Bases de Datos
- **JITEL**: Jornadas de Ingeniería Telemática
- **LATINCOM**: IEEE Latin-American Conference on Communications
- **NoF**: International Conference on Network of the Future
- **PERCOM**: IEEE International Conference on Pervasive Computing and Communications
- **PMC**: Pervasive and Mobile Computing
- **SMARTCOMP**: IEEE International Conference on Smart Computing

AC.3 Glossary of artifacts

- **CA**: Continuous Adaptation
- **CRE**: Continuous Reasoning Engine
- **ConDADO**: Continuous Distributed Application Deployment Optimization
- **DADO**: Distributed Application Deployment Optimization
- **DeQALE**: Deep QoS-Adaptive Learning Environment
- **MO-SFO**: Multi-Objective SDN Fog Optimization
- **MinMaxLat**: Minimization of Maximum Latency
- **MinMeanLat**: Minimization of Mean Latency
- **NIoTO**: Next-generation IoT Optimization
- **PADEC**: Privacy-Aware DEvice Communication
- **PODS**: Pervasive Opportunistic Delegation of Services
- **S-DADO**: Stochastic Distributed Application Deployment Optimization

AC.4 Glossary of terms

- **Application dimension**: Dimension of deployment that represents software and its interaction with the other two dimensions. In next-gen IoT, the application dimension is generally architected as an MSA, using the SOC paradigm.
- **Capital Expenditures**: Cost of acquiring new assets, which are generally one-time costs.
- **Cloud computing**: Computational paradigm where application components, usually microservices, are executed by cloud servers that are in remote data centers.

- **Cloud servers**: Computational device, real or virtual, that resides in a cloud data center. They are usually more powerful than fog nodes.
- **Cloud-to-Thing Continuum**: Umbrella term for the computational paradigms that propose bringing computing devices with the key traits of cloud computing closer to users, creating a continuum of computing devices from the cloud to the IoT device (thing).
- **Computing dimension**: Dimension of deployment that represents computing hardware and its interaction with the other two dimensions. in next-gen IoT, the computing dimension generally uses the Cloud-to-Thing continuum paradigm.
- **Constraint (mathematical programming)**: Element of a mathematical programming model that takes the form of an equality or inequality and describes the behavior of the model. Constraints must always hold true for a solution to be considered valid. If they are contradictiory, either due to their definition or to the values of the parameters, the model is deemed infeasible. They can express how different decision variables interact, as well as how they interact with the parameters. From the point of view of computer science, it represents the behavioral rules of a system based on mathematical programming.
- **Continuous reasoning**: Technique to speed up continuous evaluation processes based on reasoning over which elements in a scenario were affected by a change, and re-evaluating only said affected elements.
- **Controller Placement Problem**: Problem of determining the placement of a set of SDN controllers, the assignment of each controller to one or more SDN switches, and the routing from each controller to its assigned switches, to optimize the QoS of a given scenario. It is the main deployment optimization problem of the networking dimension.
- **Count to infinity**: Problem of the distance-vector routing algorithm where, if a link is disconnected, the devices will wrongly report their distances to

another device, entering an infinite loop of constant updates where the distance is increased slowly until reaching infinity.

- **Decentralized Computation Distribution Problem**: Problem of determining the number of replicas of each microservice and the placement of each replica to optimize the QoS in a given scenario. It is the main deployment optimization problem of the application dimension.

- **Decision variable**: Element of a mathematical programming model whose optimal value is not known *a priori* and must be determined by solving the problem. From the point of view of computer science, it represents the outputs of a system based on mathematical programming. From a more general point of view, it represents the decisions that the user of a mathematical programming-based system desires to optimize.

- **DevOps**: Software development methodology that focuses on accelerating the delivery of features to final users by continuously following a process that involves both devs and ops.

- **Edge computing**: Computational paradigm where application components, usually microservices, are executed by edge nodes that are one network hop away from the final IoT or mobile devices.

- **Edge node**: Computational device that is one network hop away from the final IoT or mobile devices, or a subset of them. They are usually more powerful than IoT or mobile devices.

- **Execution time**: Time that a given computing device takes to perform a given task.

- **FLOW-INSTALL**: Message sent by an SDN controller to an SDN switch to install a rule on how to manage traffic. Finalizes the flow setup process.

- **Flow setup**: Process in SDN networks where an SDN controller installs a rule in an SDN switch to allow it to handle a given type of messages. Flow setups are requested by SDN switches when they find a message that does not match any of their installed rules.

- **Fog Node Placement Problem**: Problem of determining the placement of a set of FNs, the assignment of each FN to one or more IoT devices, and the routing from each IoT device to its assigned FN, to optimize the QoS of a given scenario. It is the main deployment optimization problem of the computing dimension.

- **Fog computing**: Computational paradigm where application components, usually microservices, are executed by fog nodes that are at different points in the network, closer to final devices than the cloud, but normally further away than the edge. It is possible to have multiple layers of fog nodes in a fog computing scenario.

- **Fog node**: Computational device that is closer to the final IoT or mobile devices, or a subset of them, than the cloud data center. Generally, they are more than one network hop away from the final devices. They are usually more powerful than edge nodes. If it is referred to as capital (Fog Node) or FN, it refers to the model of fog node used in the FNPP, which integrates the computing device with an SDN switch and is allowed to be within one hop of the final devices.

- **Industrial Internet of Things**: Term that refers to the application of the IoT paradigm to the industrial field to automate and manage industrial and manufacturing processes.

- **Infrastructure as a Service**: Cloud computing service model where the cloud provider offers the consumer virtual machines, provisioned from the resources of the general pool, and allows the consumer to control the software executed by these machines.

- **Infrastructure**: Hardware equipment of a given scenario. In terms of deployment, it is the union of the computing and networking dimensions.

- **Internet Protocol**: Networking paradigm where each network device can only forward messages, exclusively using the destination fields. Each network device in this paradigm embodies both the data and control plane.

- **Internet of Medical Things**: Term that refers to the application of the IoT paradigm to the healthcare field, generally to automatically and remotely monitor, diagnose, or treat patients.

- **Internet of Things**: Computer science paradigm in which the inputs of an application can come from the real world, and the outputs can also affect the real world directly, making use of devices that are able to communicate the real-world data over the Internet.

- **IoT device**: ”Computing device, usually low-power, low-capacity, and low-cost, that has sensors and actuators and can connect to the Internet, normally using wireless technologies. IoT devices are generally integrated into ””things””, such as clothes, wearable accessories or home appliances.”

- **Latency**: Time from the source of a communication sending a message to the destination receiving it.

- **Mathematical programming**: Mathematical model that expresses an optimization model as a set of parameters, decision variables, constraints, and an objective function.

- **Microservices Architecture**: Software architecture where an application is designed as a set of microservices that collaborate to perform application functionalities. It generally is a SOA where services are very cohesive, very small, and very numerous, accentuating the characeristics of SOC.

- **Microservice**: Very small service that generally performs a single task. Thus, a microservice is very loosely-coupled, very cohesive service that reinforces the characteristics of SOC.
- **Mist computing**: Computational paradigm where application components, usually microservices, are executed by the IoT or mobile devices themselves.
- **Mixed-Integer Linear Programming**: Subset of mathematical programming models where the decision variables can be integer or binary, and the constraints and objective function are linear. If a problem can be expressed using Mixed-Integer Linear Programming, it can be solved using appropriate software.
- **Monolithic application**: Application paradigm that considers that applications are made of a single and indivisible software module that contains all its logic.
- **Networking dimension.Dimension of deployment that represents networking hardware equipment and its interaction with the other two dimensions. in next-gen IoT, the networking dimension generally uses the SDN paradigm.**:
- **OF-STAT**: Message sent by an SDN controller to monitor the network.
- **Objective function**: Element of a mathematical programming model that takes the form of a function of the decision variables, and whose value must be either maximized or minimized. From a more general point of view, it represents the metrics that a mathematical programming-based system must optimize.
- **Operational Expenditures**: Ongoing cost of maintenance of assets, generally expressed as the cost per unit of time, or the cost per usage over time.
- **Opportunistic network**: Ad-hoc wireless network where mobile devices communicate with those within their reach, forwarding messages as users move. Two devices in an opportunistic network can communicate without the need for an end-to-end communication path by exploiting the mobility of other devices.

- **PACKET-IN**: Message sent by an SDN switch to its SDN controller to request a flow setup.
- **PACKET-OUT**: Message sent by an SDN controller to an SDN switch as a response to a PACKET-IN, which indicates how to handle the message.
- **Parameter (mathematical programming)**: Element of a mathematical programming model that allows the model to be generalized. Although they are technically variable, once the model is created, they remain constant. From the point of view of computer science, it represents the inputs of a system based on mathematical programming.
- **Private Function Evaluation**: System that allows two parties to compute the result of a function without any of the parties revealing their inputs to the other party.
- **Quality of Service**: Term related to all the non-functional quality metrics provided by a given service, such as response time or monetary cost. When used in a given scenario, it normally refers to the metrics that are relevant for a certain party, generally the user or the provider, in that scenario.
- **Response time**: Time from a device requesting the execution of a functionality to the same device receiving the result of said functionality. Response time is the summation of the execution times of the microservices of the functionality and the network latencies of the messages between the devices where they were deployed.
- **SDN controller**: Hardware element in an SDN network that embodies the control plane by running SDN controller software. There can be multiple SDN controllers in a network.
- **SDN switch**: Network equipment in SDN that embodies the data plane. It follows the rules installed by the controller, and is able to request for flow setups.

- **Service discovery**: Process in which a device which does not initially know where a given microservice is deployed finds the microservice. It can be performed through service discovery applications, or can be managed at the network level.
- **Service-Oriented Architecture**: Software architecture where an application is designed as a set of services that collaborate to perform application functionalities.
- **Service-Oriented Computing**: Application paradigm that considers that applications are made of multiple autonomous, loosely-coupled, cohesive, reusable, and platform-independent components.
- **Service**: Autonomous, loosely-coupled, cohesive, reusable, and platform-independent component that is part of a SOA.
- **Software-Defined Networking**: Networking paradigm where the behavior of the network, including the operations performed over messages, how they are forwarded, or which fields of the headers should be considered to decide on how to behave with a given message, can all be programmed. In this paradigm, the control and data plane are separated.

Bibliography

[1] 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.

[2] J. L. Herrera, J. Galán-Jiménez, J. Berrocal, and J. M. Murillo, “Optimizing the response time in sdn-fog environments for time-strict iot applications,” *IEEE Internet of Things Journal*, vol. 8, no. 23, pp. 17 172–17 185, 2021.

[3] J. L. Herrera, J. Galán-Jiménez, L. Foschini, P. Bellavista, J. Berrocal, and J. M. Murillo, “Qos-aware fog node placement for intensive iot applications in sdn-fog scenarios,” *IEEE Internet of Things Journal*, 2022.

[4] J. L. Herrera, J. Berrocal, and J. M. Murillo, “Deploying next generation iot applications through sdn-enabled fog infrastructures,” in *2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. IEEE, 2022, pp. 130–131.

[5] J. L. Herrera, J. Galán-Jiménez, J. García-Alonso, J. Berrocal, and J. M. Murillo, “Joint optimization of response time and deployment cost in next-gen iot applications,” *IEEE Internet of Things Journal*, 2022.

[6] M. Weiser, “The computer for the 21 st century,” *Scientific american*, vol. 265, no. 3, pp. 94–105, 1991.

[7] Cisco. (2020, March) Cisco Annual Internet Report (2018–2023). (visited on Jan. 29, 2021). [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual\\-internet-report/white-paper-c11-741490.pdf

[8] N. Dimitriou and K. Kontovasilis, "Mobile iot networks: Standards, use cases and requirements," in *2021 IEEE Conference on Standards for Communications and Networking (CSCN)*. IEEE, 2021, pp. 46–51.

[9] 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.

[10] 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, 7 2020.

[11] Meticulous Research, "Industrial IoT Market by Component (Hardware, Platform, Services and Connectivity), Industry Verticals (Agriculture, Manufacturing, Energy & Utility, Oil & Gas, Aerospace, Retail and Others) and Geography - Global Forecasts to 2029," Tech. Rep., 2022. [Online]. Available: https://www.meticulousresearch.com/product/industrial-iot-market-5102

[12] ITU-T, *Definitions of terms related to quality of service*, International Telecommunication Union Telecommunication Standarization Sector Std. E.800, Rev. 09/2008.

[13] 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.

[14] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: State of the art and research challenges," *Computer*, vol. 40, no. 11, pp. 38–45, 2007.

[15] G. Mazlami, J. Cito, and P. Leitner, “Extraction of microservices from monolithic software architectures,” in *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 2017, pp. 524–531.

[16] K. Indrasiri. Microservices in practice - key architectural concepts of an MSA. (visited on Jan. 14, 2021). [Online]. Available: https://wso2.com/whitepapers/microservices-in-practice-key-architectural-concepts-of-an-msa/

[17] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, “Devops,” *IEEE Software*, vol. 33, no. 3, pp. 94–100, 2016.

[18] H. Haas, D. Orchard, F. McCabe, C. Ferris, E. Newcomer, D. Booth, and M. Champion, “Web services architecture,” W3C, W3C Note, Feb. 2004, https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.

[19] R. T. Fielding, *Architectural styles and the design of network-based software architectures*. University of California, Irvine, 2000.

[20] 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, 2016.

[21] P. Mell and T. Grance, “The NIST definition of cloud computing,” *Cloud Computing and Government: Background, Benefits, Risks*, pp. 171–173, sep 2011. [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-145/final

[22] Amazon Web Services, “AWS Global Infrastructure,” 2022. [Online]. Available: https://aws.amazon.com/about-aws/global-infrastructure/?nc1=h{_}ls

[23] 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. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1383762118306349

[24] J. F. Kurose and K. W. Ross, "Computer networking: A top-down approach edition," *Addision Wesley*, 2007.

[25] S. Forti and A. Brogi, "Continuous reasoning for managing next-gen distributed applications," in *ICLP 2020 Tech. Comm.s*, ser. EPTCS, vol. 325, 2020, pp. 164–177.

[26] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 27–51, 2014.

[27] T. Das, V. Sridharan, and M. Gurusamy, "A survey on controller placement in sdn," *IEEE Communications Surveys Tutorials*, pp. 1–1, 2019.

[28] A. C. Baktir, A. Ozgovde, and C. Ersoy, "How Can Edge Computing Benefit from Software-Defined Networking: A Survey, Use Cases, and Future Directions," pp. 2359–2391, oct 2017.

[29] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "Onos: towards an open, distributed sdn os," in *Proceedings of the third workshop on Hot topics in software defined networking*, 2014, pp. 1–6.

[30] S. Kaur, J. Singh, and N. S. Ghumman, "Network programmability using pox controller," in *ICCCS International conference on communication, computing & systems, IEEE*, vol. 138. sn, 2014, p. 70.

[31] A. Brogi and S. Forti, "Qos-aware deployment of iot applications through the fog," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1185–1192, 2017.

[32] 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, dec 2019.

[33] 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.

[34] T. N. Gia, M. Jiang, A.-M. Rahmani, T. Westerlund, P. Liljeberg, and H. Tenhunen, "Fog computing in healthcare internet of things: A case study on ecg feature extraction," in *2015 IEEE international conference on computer and information technology; ubiquitous computing and communications; dependable, autonomic and secure computing; pervasive intelligence and computing*. IEEE, 2015, pp. 356–363.

[35] I. Bedhief, L. Foschini, P. Bellavista, M. Kassar, and T. Aguili, "Toward self-adaptive software defined fog networking architecture for IIoT and industry 4.0," in *Proceedings of IEEE CAMAD 2019*, 2019.

[36] J. L. Herrera, L. Foschini, J. Galán-Jiménez, and J. Berrocal, "The service node placement problem in software-defined fog networks," in *2020 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2020, pp. 1–6.

[37] J. L. Herrera, P. Bellavista, L. Foschini, J. Galán-Jiménez, J. M. Murillo, and J. Berrocal, "Meeting stringent qos requirements in iiot-based scenarios," in *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE, 2020, pp. 1–6.

[38] 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, pp. 1–26, 2020.

[39] 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.

[40] P. Johannesson and E. Perjons, *An Introduction to Design Science*. Springer International Publishing, 2014.

[41] V. Vaishnavi and W. Kuechler, *Design Science Research Methods and Patterns: Innovating Information and Communication Technology*, 2015.

[42] J. L. Herrera, J. Galán-Jiménez, P. Bellavista, L. Foschini, J. Garcia-Alonso, J. M. Murillo, and J. Berrocal, “Optimal deployment of fog nodes, microservices and sdn controllers in time-sensitive iot scenarios,” in *2021 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2021, pp. 1–6.

[43] A. R. Hevner, “A Three Cycle View of Design Science Research,” Tech. Rep. 2, 2007.

[44] J. L. Herrera, P. Bellavista, L. Foschini, J. García-Alonso, J. Galán-Jiménez, and J. Berrocal, “Fog node placement in iot scenarios with stringent qos requirements: Experimental evaluation,” in *ICC 2021-IEEE International Conference on Communications*. IEEE, 2021, pp. 1–6.

[45] J. L. Herrera, J. Galán-Jiménez, J. García-Alonso, J. Berrocal, and J. M. M. Rodríguez, “Despliegue Óptimo de Aplicaciones IoT Distribuidas,” in *JISBD2021*, S. Abrahão Gonzales, Ed. SISTEDES, 2021. [Online]. Available: http://hdl.handle.net/11705/JISBD/2021/024

[46] M. J. Neely, “Stochastic network optimization with application to communication and queueing systems,” *Synthesis Lectures on Communication Networks*, vol. 3, no. 1, pp. 1–211, 2010.

[47] 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)*. IEEE, 2020, pp. 33–37.

[48] J. L. Herrera, J. Berrocal, J. M. Murillo, H.-Y. Chen, and C. Julien, "A privacy-aware architecture to share device-to-device contextual information," in *2020 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 2020, pp. 188–195.

[49] G. W. de Oliveira, R. T. Ney, J. L. Herrera, D. M. Batista, R. Hirata, J. Galán-Jiménez, J. Berrocal, J. M. Murillo, A. L. Dos Santos, and M. Nogueira, "Predicting response time in sdn-fog environments for iiot applications," in *2021 IEEE Latin-American Conference on Communications (LATINCOM)*. IEEE, 2021, pp. 1–6.

[50] J. L. Herrera, J. Galán-Jiménez, J. Berrocal, and J. M. M. Rodríguez, "Optimizing the Response Time in SDN-Fog Environments for Time-Strict IoT Applications (Summary)," in *JCIS2021*, P. Fernández, Ed. SISTEDES, 2021. [Online]. Available: http://hdl.handle.net/11705/JCIS/2021/013

[51] J. L. Herrera, J. Galán-Jiménez, J. Berrocal, and J. M. Murillo, "Optimizing the Response Time in SDN-Fog Environments for Time-Strict IoT Applications," in *JITEL 2021*, V. Carneiro, Ed. JITEL, 2021. [Online]. Available: https://jitel21.udc.es/files/actasJITEL21.pdf

[52] J. Galàn-Jimènez, M. Polverini, F. G. Lavacca, J. L. Herrera, and J. Berrocal, "On the tradeoff between load balancing and energy-efficiency in hybrid ip/sdn networks," in *2021 12th International Conference on Network of the Future (NoF)*. IEEE, 2021, pp. 1–9.

[53] J. L. Herrera, H.-Y. Chen, J. Berrocal, J. M. Murillo, and C. Julien, "Privacy-aware and context-sensitive access control for opportunistic data sharing," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2021, pp. 762–769.

[54] J. L. Herrera, J. Galán-Jiménez, J. García-Alonso, J. Berrocal, and J. M. M. Rodríguez, "Joint Optimization of Response Time and Deployment Cost in Next-Gen IoT Applications (Summary)," in *JCIS2022*, E. Navarro, Ed. SISTEDES, 2022. [Online]. Available: http://hdl.handle.net/11705/JCIS/2022/008

[55] J. L. Herrera, J. Berrocal, J. Galán-Jiménez, J. García-Alonso, and J. M. Murillo, "Quality of service-adaptive industrial internet of things leveraging edge computing and deep reinforcement learning: The deep qos-adaptive learning environment (deqale) architecture," in *2022 17th Iberian Conference on Information Systems and Technologies (CISTI)*. IEEE, 2022, pp. 1–4.

[56] M. Jiménez-Lázaro, J. L. Herrera, J. Berrocal, and J. Galán-Jiménez, "Improving the energy efficiency of software-defined networks through the prediction of network configurations," *Electronics*, vol. 11, no. 17, p. 2739, 2022.

[57] J. Galán-Jiménez, M. Polverini, F. G. Lavacca, J. L. Herrera, and J. Berrocal, "Joint energy efficiency and load balancing optimization in hybrid ip/sdn networks," *Annals of Telecommunications*, pp. 1–19, 2022.

[58] J. L. Herrera, H.-Y. Chen, J. Berrocal, J. M. Murillo, and C. Julien, "Context-aware privacy-preserving access control for mobile computing," *Pervasive and Mobile Computing*, p. 101725, nov 2022. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1574119222001389

[59] G. Guastaroba, M. Savelsbergh, and M. G. Speranza, "Adaptive kernel search: A heuristic for solving mixed integer linear programs," *European Journal of Operational Research*, vol. 263, no. 3, pp. 789–804, 2017.

[60] P. W. O'Hearn, "Continuous reasoning: Scaling the impact of formal methods," in *Proceedings of ACM/IEEE Symposium on Logic in Computer Science*, 2018, pp. 13–25. [Online]. Available: https://doi.org/10.1145/3209108

[61] G. Bonofiglio, V. Iovinella, G. Lospoto, and G. Di Battista, “Kathará: A container-based framework for implementing network function virtualization and software defined networks,” in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2018, pp. 1–9.

[62] V. F. Pacheco, “Chained Microservice Design Pattern,” in *Microservice Patterns and Best Practices*, O’Reilly, Ed., 2018, ch. 8.

[63] S. Wang, M. Zafer, and K. K. Leung, “Online placement of multi-component applications in edge computing environments,” *IEEE Access*, vol. 5, pp. 2514–2533, 2017.

[64] Google Inc., “Kubernetes,” 2014. [Online]. Available: https://kubernetes.io/

[65] Edge, “The leading Edge computing platform — Edge,” 2022. [Online]. Available: https://edge.network/en/

[66] 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.

[67] D. Maguire, *The business benefits of GIS: an ROI approach*, 1st ed. Redlands: ESRI press, 2008.

[68] Z. Á. Mann, A. Metzger, J. Prade, and R. Seidl, “Optimized application deployment in the fog,” in *International Conference on Service-Oriented Computing*. Springer, 2019, pp. 283–298.

[69] S. Forti, G. Bisicchia, and A. Brogi, “Declarative continuous reasoning in the cloud-iot continuum,” *Journal of Logic and Computation*, vol. 32, no. 2, pp. 206–232, 2022.