UNIVERSITÀ
DEGLI STUDI
FIRENZE

PhD Program in Smart Computing
University of Florence, University of Pisa, University of Siena

Companion Fog Computing: Migrating the Fog Service to Support IoT Device Mobility

Carlo Puliafito

Advisors:

Prof. Enzo Mingozzi

Prof. Giuseppe Anastasi

Head of the PhD Program:

Prof. Paolo Frasconi

Evaluation Committee:
Prof. Marilia Curado, *University of Coimbra*
Prof. Fulvio Risso, *Polytechnic University of Turin*

XXXII cycle — October 2019

To Nonno Carlo, who is all I want to become

Acknowledgements

Academia

First and foremost, I thank **Prof. Enzo Mingozzi**. He has advised me since the very first day, teaching me how to be a researcher. He has introduced me to my current research topic, which is giving me great satisfaction. He has always encouraged me, both in good and difficult moments. I owe a lot to him.

Also **Prof. Giuseppe Anastasi** gave me a lot. He kindly welcomed me in Pisa and has always made helpful suggestions to improve my work. As director of the department, he has made it possible to work in a safe and stimulating place.

Then, I sincerely thank **Prof. Omer Rana** for the period that I spent at Cardiff University, United Kingdom, under his supervision. This was a very formative experience during which I had the possibility to produce valid works, start new collaborations with foreign research groups, and get to know new people and cultures.

I also thank **Prof. Andrea Passarella** and **Prof. Gianluca Dini** as members of the Supervisory Committee that evaluated my work at the end of each year. They both made interesting observations and comments, which significantly enhanced the quality of my research.

Prof. Marilia Curado and **Prof. Fulvio Risso** deserve a special thank as members of the Evaluation Committee that has evaluated this thesis. Their comments on this thesis as well as their suggestions for future work were much appreciated.

I also thank **Prof. Carlo Vallati**, as he has always been available to answer any of my questions and clear out all my doubts. His expertise and kindness have been very important to me.

It has been a pleasure for me to collaborate with **Giovanni Merlino** and **Prof. Francesco Longo** from the University of Messina. They have significantly helped me to improve my programming skills and my knowledge of Linux systems.

I also express my gratitude to the research group of the University of Campinas, Brazil, composed by **Prof. Luiz F. Bittencourt**, **Diogo M. Gonçalves**, **Márcio M. Lopes**, **Leonardo L. Martins**, and **Edmundo Madeira**. Working with them has been a pleasure for me.

All the professors, my colleagues, and the administrative staff from the Department of Engineering of the University of Pisa deserve my acknowledgements for their professionalism and kindness. For the same reason, I thank **Prof. Paolo Frasconi** as head of the regional PhD program in Smart Computing, all the members of the Doctoral Committee, the administrative staff of the PhD program in general and **Dr. Simona Altamura** in particular.

I also acknowledge the **Pegaso** grant from Regione Toscana for the scholarship that I have got.

Family and friends

My thoughts go first and foremost to my **family**. They have always provided me with anything a person could wish, supporting and encouraging me every single day. I express my gratitude to my parents, my brother, my grandparents, my aunts and uncles, and my cousins. I especially thank my father, who, in a difficult phase of my life, advised me to undertake a PhD program.

Abdullah, who is a PhD candidate in Smart Computing like me, deserves my thanks. He has proved to be a true friend and a brother to me. I wish him a golden future in terms of health, wealth, family, and career.

I am also grateful to Laura and the **Colajanni** family in general: Michele, Grazia, Elena, and Ilaria. They gave me a lot.

I also acknowledge my colleagues and friends from the University of Pisa. In particular, I express my gratitude to my friends of the "**Coffee League**" (Michele, Maurizio, Dario, Alessandro Bondielli, and Alessandro Renda) for the nice moments that we are spending together and for standing my invasive hyperactivity. I also thank **Marco** for sharing with me the three years of PhD program in Smart Computing. Besides, I thank **Francesca** and **Antonio** for being my office mates during the last years.

Next, I thank my friends from the **rowing class** for the nice moments that we have spent together. In particular, I thank Michele, Vito, Francesco, Luca, Alessandro, Mauro, Giulio, and Vasco.

I am grateful to all the people from the **dancing class**. In particular, I thank the teachers, Sandro and Francesca for their passion, kindness, and professionalism. I then express my gratitude to all my mates for their congeniality and joyfulness: Giulia, Daniele, Roberto, Marina, Urška, Annemarie, Nunzio, Antonella, Giulio, Vinicio, Letizia, Ambra, Filippo, Marta, Rumiana, Alex, Aurora, Roberto, Maria Teresa, Federica, and Claudio.

Staff members from **Residence Le Benedettine** have given me a lot during these years. They have always been kind and welcoming. I especially thank Antonella, Alessio, Marta, Rosario, Mirko, Choman, and Roberto.

I owe a lot also to my friends from Cardiff, where I spent six months of the PhD program. I thank **Lisa** and **Bobby** for their hospitality, their cheerfulness, and the enjoyable trips in Wales and England. I also acknowledge my colleagues and friends from **Cardiff University** (Lauren, Stefano, Adi, Roberto, Beryl, Masoud, Alessandro, and Daphne) for the amusing moments spent together. The experience in Cardiff is unforgettable also thanks to all of them.

I acknowledge my friends from **Catania** and in particular Irene, Manuel, Alessia, and Graziano as well as Paolo and Rita, Laura and Ettore, Stefania and Roberto, Salvo and Loredana, Maurizio and Loredana, Enzo and Delfina, Ottavio and Patrizia, Michele and Carla, Massimo and Carla, and the friends from Leonforte.

To conclude, I thank all the **artists and bands** that have helped me with their music. A beautiful song can often turn a bad day around. Especially I thank: AC/DC, Aerosmith, Aventura, Bastille, Black Eyed Peas, Bronski Beat, Cèline Dion, Coldplay, Corona, Cranberries, Credence, David Bowie, David Guetta, Depeche Mode, Dire Straits, Eagles, Ed Sheeran, Eiffel 65, Enrique Iglesias, Enya, Europe, Fools Garden, Franco Battiato, Gigi D'agostino, Grupo Extra, Guns N' Roses, Haddaway, Imagine Dragons, Linkin Park, Liquido, Lynyrd Skynyrd, Marc Anthony, Maroon 5, Metallica, Moby, Mozart, Pinto Picasso, Placebo, Police, Prince Royce, R.E.M., Romeo Santos, Shivaree, Sia, SNAP!, Strokes, Supertramp, Toploader, Toto, Twins, U2, Venditti, Vivaldi.

Abstract

Fog Computing (FC) extends the Cloud towards the network edge. It provides end devices with access to resources and services that are located in topological proximity to them. This proximity enables key benefits (e.g., low latencies, reduced bandwidth consumption) that are not achievable when relying on Cloud-only solutions. FC is a horizontal paradigm in the sense that it is generic enough to be leveraged in a number of different application domains. However, its characteristics make it particularly suitable for the Internet of Things (IoT). After providing a survey on FC for the IoT, this thesis focuses on a specific research problem introduced with FC: the issue raised by device mobility in a FC environment. When a (IoT) device moves, the topological distance to the current Fog node (i.e., the Fog resource hosting the Fog service) may increase. Therefore, device mobility may impair the FC benefits, which are a result of Fog proximity. The objective is to support device mobility, namely to provide the FC benefits even in the presence of mobile devices. The most popular approach in literature to achieve this purpose is by migrating the Fog service across the Fog infrastructure, thus to let it be always close enough to the served mobile device. We refer to this paradigm as Companion Fog Computing (CFC), an extension of standard FC where the Fog service behaves as a "companion" of the mobile application. In this thesis, we first analyse the different aspects that characterise CFC, by reporting the state of the art for each aspect and highlighting the open issues and research opportunities. Then, we propose our own solutions. Specifically, we first consider Fog services as application containers and set up a small-scale FC testbed to perform a quantitative evaluation and comparison of the existing container migration techniques. The objective of this work is twofold: (i) clarify whether there exists a technique that performs the best under any condition or, otherwise, understand which technique is the most appropriate under which condition; (ii) better understand the inner workings of container migration. The second contribution is the proposal and validation of a platform, which we call Companion Fog Platform (CFP), that provides the necessary mechanisms to support device mobility in the Fog. Our CFP implements Fog services as containers and migrates them between Fog nodes according to the state-of-the-art container migration techniques. The third work described in this thesis is MobFogSim, a simulator that extends iFogSim to model device mobility and service migration in FC. After providing its design and implementation details, we validate the simulator. This is done by reproducing in MobFogSim the same conditions under which we evaluated the container migration techniques and by comparing the simulation results against those over the real testbed.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The **Internet of Things** (IoT) conceives of a world where any object, from a "smart" one (e.g., a smartphone, a wearable device) to a "dumb" thing (e.g., a lamp post, a dumpster), can join the Internet (Atzori et al., 2010). Such objects may not only exchange data but may also store and process data, use sensors to collect data from the surrounding environment, and actively intervene on the latter through actuators. Moreover, people are also a part of this ecosystem, consuming and producing data through their smartphones and wearable devices. The number of objects connected to the Internet surpassed the world human population in 2010 (Al-Fuqaha et al., 2015) and is expected to reach between 50 and 100 billion by 2020 (Amiot, 2015). Furthermore, the McKinsey Global Institute forecasts a potential economic impact for IoT applications of as much as $11.1 trillion per year in 2025 (Manyika et al., 2015).

The IoT is necessary for the implementation of many innovative services, but it is not sufficient in most cases to host such services directly. The great amount of heterogeneous data (i.e., the Big Data (Chen et al., 2014)) collected by IoT devices needs to be stored and processed, and the obtained insights need to be retrieved for visualisation or actuation. However, all these tasks can rarely be performed on the IoT devices themselves, as such devices typically have limited compute, storage, and networking resources and can be battery-powered (Delicato et al., 2017). Therefore, the IoT needs support from more powerful resources – the most common being the use of **Cloud Computing** (CC) resources. CC is defined by the National Institute of Standards and Technology (NIST) as "*a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources* (*e.g., networks, servers, storage, applications, and services*) *that can be rapidly provisioned and released with minimal management effort or service provider interaction*" (Mell and Grance, 2011). In other words, CC allows to transparently use almost limitless, virtualised computing resources whenever needed and to release them when not needed anymore. IoT devices may offload the collected data and complex computation onto the Cloud and retrieve the obtained insights from it, thus exploiting its many advantages (Botta

et al., 2016). Clouds may be public, private, or an hybrid combination of both (Suciu et al., 2012). The distinctive feature of public Clouds is that services and resources are made available by a third-party provider to anyone who requires them. Such resources are off-premises and rented according to a pay-per-use pricing model[1]. On the other hand, private Clouds are such that services and resources are accessible only by specific users (e.g., the members of an organisation). Even though also private Clouds can be off-premises and managed by third-party providers under payment, they typically are on-premises, and their resources are released for free, as in that case users and providers coincide.

Although CC can support resource-limited devices with a virtually unlimited amount of resources, it presents a non-negligible shortcoming. CC resources are concentrated in few Data Centres (DCs) which are considerably far away from the vast majority of data producers and consumers. This is especially true for public Clouds rather than private ones. Such a considerable distance from end (user) devices leads to some drawbacks (first and foremost, high round-trip latencies) that are not acceptable for several emerging applications and services. The **Fog Computing** (FC) paradigm was therefore proposed (Bonomi et al., 2012) as a means to extend Cloud-based capabilities towards the network edge, distributing resources and services of computing, storage, and networking along the Cloud-to-Things continuum and in closer topological proximity[2] to IoT devices. This proximity permits a set of advantages with respect to the exclusive dependence on the distant Cloud, such as low latencies and reduced bandwidth consumption, to name a few. Using FC, the key characteristics of CC should be still preserved, including resource virtualisation, transparency, and elasticity (Consortium, 2017). Furthermore, as for the Cloud resources and services, also the Fog ones may be provided either for free or under payment. For instance, a municipality can exploit part of its own Fog resources for free and grant upon payment the rest to third-party developers. Since its inception in 2012, FC has been drawing increasing interest. In a report commissioned by the OpenFog Consortium (OFC), which is now incorporated by the Industrial Internet Consortium (IIC[3]), the global Fog market opportunity is forecast to be worth $3.7 billion by 2019 and to reach $18.2 billion by 2022 (Hedge, 2017) – with significant (and growing) academic and industry literature in this area.

Several research challenges still need to be faced to fulfil the full potential of FC. The focus of this thesis is on the challenge raised by **device mobility** in a FC environment. Mobility may indeed compromise the FC advantages, which are a

[1] See `https://reasonstreet.co/business-model-pay-per-use/`. Last accessed: 10 October 2019.

[2] Topological proximity means that the communication path between end devices and Fog resources is short. We believe that it is worth distinguishing this concept from that of geographical proximity, which is instead expressed in terms of physical distance. Indeed, while the topological proximity typically entails the geographical one, the opposite is not always true.

[3] See `https://www.iiconsortium.org/`. Last accessed: 10 October 2019.

result of the Fog proximity to the end devices. When a device moves, the topological distance to the associated Fog node (i.e., the node hosting the Fog service) may increase, possibly impairing the distinctive benefits of FC. The objective is to support the mobility of IoT devices or, in other words, to guarantee the Fog benefits also when such devices move from one place to another. To this purpose, a possible approach is to migrate the Fog service from a Fog node to another, keeping it close enough to the associated application component on the mobile IoT device. Hence, the Fog service would behave as a "companion" of the mobile application; we refer to this as **Companion Fog Computing**. In what follows, we highlight the different contributions that we make with regard to FC and, more specifically, with respect to the mobility challenge, also delineating the structure of this thesis.

1.1 Contributions and structure of the thesis

The first contribution (see Chapter 2) is a comprehensive **survey on FC for the IoT**, which provides the general background of this thesis. Specifically, we describe the principles and the research challenges characterising FC. The extension of Cloud systems towards the network edge creates new challenges and can have an impact on existing approaches employed in Cloud-based deployments. Research directions being adopted by the community are highlighted, with an indication of which of these are likely to have the greatest impact. An overview of existing FC software and hardware platforms for the IoT is also provided, along with the standardisation efforts in this area initiated by the OFC.

Next, in Chapter 3, we analyse the issue of **device mobility in FC**, which represents the focus of this thesis. To this purpose, we show the importance of the topic by describing some real use cases that require mobility support in the Fog. Then, we highlight the different aspects that need to be dealt with to achieve the objective of Companion Fog Computing and report the state of the art for each of them. Finally, we point out the open issues and future research directions in the field.

In Chapter 4, we deal with the first technical aspect related to mobility support in the Fog, i.e., a study of the existing service migration techniques. This is the most preliminary aspect to consider, since all the other aspects require a full understanding of the former. As we clarify in Chapter 4, we implement Fog services as *runC* containers and inspect the most significant differences with respect to Docker containers. Then, we provide a comprehensive overview of the **state-of-the-art container migration techniques**. The main contribution is the extensive **performance evaluation** of these techniques, which we conducted over a real FC testbed. The obtained results shed light on container migration within FC environments by: (i) clarifying, in general, which migration technique might be the most appropriate under certain network and service conditions; (ii) allowing to better understand the inner workings of container migration.

In Chapter 5, we propose and validate a FC platform that leverages container migration and provides all the basic mechanisms to support IoT device mobility. From now onwards, we refer to this platform as **Companion Fog Platform**. Validation is carried out over a real FC testbed and shows a considerable performance improvement with respect to standard FC, i.e., FC without service migration.

Simulation can be a time- and cost-effective way to evaluate service migration solutions in FC environments with mobile users. This is especially true when dealing with large-scale environments with a great number of users. To the best of our knowledge, the state-of-the-art FC simulators do not provide this possibility. In Chapter 6, we fill this gap by presenting **MobFogSim**. This is a simulator that extends *iFogSim* to model all the aspects related to device mobility and service migration in FC. Specifically, container migration in MobFogSim is modelled taking into consideration the insights obtained from the experiments in Chapter 4. Furthermore, the simulator is validated by comparing its container migration results with those produced over a real testbed and discussed in Chapter 4.

Finally, Chapter 7 draws the conclusions of this thesis and reports the lessons learnt.

Chapter 2

Fog Computing for the Internet of Things

The objective of this chapter is to provide a comprehensive survey on FC for the IoT and represent the general background of this thesis. Although FC is tailored to the IoT, we highlight that its use is applicable in a number of other contexts (e.g., content delivery, gaming, network control functions), which are out of the scope of this thesis. Table 2.1 summarises the most relevant surveys carried out in FC and organises them by contributions, also highlighting the distinctiveness of the coverage in this survey. We do not consider common contributions across these listed papers (e.g., description of FC principles, discussion of use cases for FC, review of the research challenges introduced by FC); we only highlight coverage that is unique in each case. The main contribution of our survey is the overview of existing FC platforms for the IoT. Several software and hardware systems are already available, but, to the best of our knowledge, none of the existing surveys discuss them. We believe that such a novel contribution may be of particular interest to engineers and developers. This is a changing landscape, and we provide a representative set of examples of systems.

The rest of the chapter is organised as follows. For the sake of comprehensiveness, we first provide a general overview of FC. Hence, in Section 2.1, we discuss the limitations of integrating IoT and Cloud systems which motivate the need for FC; in Section 2.2, we highlight the principles characterising FC, whereas, in Section 2.3, we analyse FC from a historical perspective. Section 2.4 highlights six IoT application domains that can benefit from FC, identifying existing literature for each domain. In Section 2.5, we analyse challenges associated with extending Cloud-based systems towards the network edge, summarising how the research community is addressing these challenges, and pointing out the main open issues and future research directions. Finally, Section 2.6 provides an overview of existing FC platforms for the IoT and outlines standardisation efforts being undertaken by the OFC.

Table 2.1: The main survey papers on FC classified by contributions.

Contribution	Papers
Focus on the IoT	Atlam et al. (2018); Perera et al. (2017); Yu et al. (2017); Ai et al. (2018); Ni et al. (2018), **this survey**
Discussion of existing software and hardware platforms	**this survey**
In-depth analysis of the state-of-the-art architectures and algorithms	Mouradian et al. (2018); Mach and Becvar (2017)
Focus on resource management and offloading of user tasks	Mao et al. (2017); Mach and Becvar (2017)
Standardisation efforts from the OFC	Ai et al. (2018), **this survey**
Standardisation efforts from ETSI	Mao et al. (2017); Mach and Becvar (2017); Ai et al. (2018); Shirazi et al. (2017)
Security and privacy issues	Stojmenovic et al. (2015); Khan et al. (2017); Roman et al. (2016); Ni et al. (2018); Mukherjee et al. (2017); Shirazi et al. (2017)
Focus on developers and engineers	Perera et al. (2017); Mahmud et al. (2017); Mach and Becvar (2017); Ai et al. (2018); Ni et al. (2018); Shirazi et al. (2017), **this survey**
Historical context & background of FC	**this survey**
Summary of recent work (i.e., from 2017 onward)	Mouradian et al. (2018); Atlam et al. (2018); Mao et al. (2017); Yu et al. (2017); Mach and Becvar (2017); Ai et al. (2018); Khan et al. (2017); Ni et al. (2018); Mukherjee et al. (2017); Shirazi et al. (2017), **this survey**

2.1 The need for Fog Computing

The integration between CC and the IoT allows resource-constrained IoT devices to offload data and complex computation onto the Cloud, taking advantage of its computational and storage capacity. However, the centralised nature of a Cloud DC can lead to a considerable topological distance between CC resources/services and the vast majority of end (user) devices. This mostly depends on where the Cloud DC is located and/or on the area it covers. As such, private Clouds are more rarely affected, unless they cover considerably wide areas (e.g., the private Cloud managed by a municipality for Smart City services) and/or are off-premises. On the contrary, public Clouds are aimed at providing global coverage, and it is not rare to be served by public Clouds located in another country or even continent. In this section, we discuss the main shortcomings of the Cloud-IoT integration, which are all due to the great distance separating the Cloud from the IoT devices.

Latency

Some IoT application domains fall under the Ultra-Reliable Low-Latency Communications (URLLC) category, where extremely low and predictable response times are of utmost importance. Road safety and autonomous driving services require latencies of less than 50 ms, while Smart Grids of up to 20 ms; Smart Factories have the most stringent requirements, with latencies varying from 250 µs to 10 ms (Schulz et al., 2017). The distance between IoT devices and the Cloud often leads to a high communication latency that makes it difficult to satisfy application time constraints. For instance, the average Round Trip Time (RTT) between an Amazon Cloud server in Virginia (USA) and a device in the US pacific coast is 66 ms; it is equal to 125 ms if the end device is in Italy; and reaches 302 ms when the device is in Beijing (Amazon, 2017).

Bandwidth consumption

The number of "smart" objects producing and/or consuming data is projected to exponentially increase within the next few years. ABI Research estimates that data captured by IoT devices in 2014 surpassed 200 exabytes (i.e., 200 billion gigabytes) and is expected to exceed 1.6 zettabytes (i.e., 1600 billion gigabytes) by 2020 (Research, 2015). For example: a smart factory might produce over a thousand terabytes (i.e., one million gigabytes) a day; self-driving cars may generate one gigabyte a second; and smart meters in the United States collect energy consumption data at 53.6 petabytes (i.e., 53.6 million gigabytes) a year (Kanellos, 2016). Delivering such a high volume of data up to the Cloud would quickly saturate the backbone network bandwidth.

Privacy and security

The use of IoT devices leads to the inevitable collection of sensitive data (e.g., health-related data) which needs adequate protection. Transmitting these data over the public Internet to a Cloud DC can incur privacy risk (Zhou et al., 2017). Due to limited (user) control over identifying a data path from the IoT device to the Cloud DC, and as IoT devices do not have enough resources to encrypt/decrypt data, challenges of confidentiality, integrity and availability (referred to as the C-I-A triad) are important. Legal implications may be raised when sensitive data collected in one country are transmitted to a Cloud DC in another country where regulations are different – an aspect that has become more significant with the recent General Data Protection Regulation (GDPR) legislation in Europe and the California Data Privacy Law (in the US).

Context awareness

Context is defined as "any information that can be used to characterise the situation of an entity" (Abowd et al., 1999). Examples of context information may be: (i) the set of nearby nodes and/or services; and (ii) local network conditions and traffic statistics. Context awareness enables provision of improved services and resources utilisation (Perera et al., 2014). Due to a disaggregation between a Cloud DC and the sensor/actuator nodes (primarily due to geographical location and lack of proximity), limited context is shared between them. For instance, if a Cloud-hosted service detected a car accident at an intersection, it would not be able to inform other vehicles in the vicinity of the accident, due to lack of local context.

Hostile environments

Some IoT devices are employed in critical domains (e.g., traffic and emergency management) where environment and people's safety are key concerns. In such scenarios, the availability of services and data must be constantly guaranteed. However, there exist contexts referred to as hostile environments (e.g., rural areas or developing countries with a weak networking infrastructure, military settings, areas afflicted by natural or man-made disasters) in which the IoT experiences intermittent or no network connectivity towards the distant Cloud, and in which, as a result, the service gets interrupted, has very low performance, or is simply not available (Satyanarayanan et al., 2013).

2.2 Fog Computing principles and strengths

FC was proposed in 2012 by Cisco in order to overcome the limitations of integration between Cloud DCs and the IoT (Bonomi et al., 2012). This section examines the principles and strengths of FC, focusing on the definition from the OFC (Consortium, 2017): "*Fog computing is a horizontal, system-level architecture that distributes computing, storage, control and networking functions closer to the users along a cloud-to-thing continuum*".

Closer to the users along a cloud-to-thing continuum

As outlined in Section 2.1, drawbacks of Cloud-IoT integration are caused due to centralisation of a Cloud DC. When talking about FC, it is worth noting that the expression "towards the network edge" does not mean "only at the network edge", as Fog services may be distributed anywhere along the continuum from Cloud to Things, hosted on nodes known as Fog Nodes (FNs) (Marìn-Tordera et al., 2017). Any device that has enough computing, storage, and networking capabilities to run advanced services can be a FN (Consortium, 2018b). Hence, FNs may be: (i)

Table 2.2: FC advantages over the simple Cloud-IoT integration.

Cloud-IoT limitation	How the Fog can overcome it
Latency	FNs perform data analytics close to where data are collected and actions should be performed. This enables predictable response times, which are essential to many IoT applications.
Bandwidth consumption	Since a good portion of the data is communicated to nearby FNs, a reduced amount is exchanged with a Cloud DC. Moreover, FNs behave as a broker between the Things and the Cloud, further reducing data transmitted to a Cloud DC. Overall, FC helps to efficiently manage the volume of Big Data, by significantly reducing bandwidth consumption. (Simmhan, 2017).
Privacy and security	Sensitive data can be locally stored and analysed by a FN, instead of being sent over the Internet up to the Cloud. However, the Cloud might need access to (part of the) sensitive data. In this case, such data may pass through the Fog for privacy enforcements that are not feasible for the resource-constrained IoT devices (e.g., extraction and transmission of metadata, complex encryptions). Therefore, the Fog can considerably improve privacy and security in modern applications and services.
Context awareness	FNs are located in closer proximity to IoT devices, improving context awareness. Exploiting context information enables improved services and/or optimises resource utilisation.
Hostile environments	FC proves to be fundamental when a service needs to be always available, but IoT devices experience intermittent or no connectivity to the Cloud. Instead, such a critical service may be provided by a nearby FN to which the IoT devices are able to connect.

resource-rich end devices (e.g., vehicles, smart traffic lights, video surveillance cameras, industrial controllers); (ii) advanced edge nodes (e.g., switches, gateways, Wi-Fi access points, cellular base stations); and (iii) specialised "core" network routers[4].

Table 2.2 identifies the advantages of FC over a simple Cloud-IoT integration, which are all consequences of the topological closeness of a Fog service to the associated IoT nodes. It is worth noting that these are all well-known strengths of FC and that the contents in Table 2.2 are taken from (Satyanarayanan et al., 2013; Satyanarayanan, 2017; Shi and Dustdar, 2016; Cisco, 2015).

System-level paradigm

The Fog is a system-level paradigm in the sense that it "*extends from the Things, over the network edges, through the Cloud, and across multiple protocol layers – not just radio systems, not just a specific protocol layer, not just at one part of an end-to-end system, but a system spanning between the Things and the Cloud*" (Consortium, 2017). Hence, FC fosters the development of systems where the overall service is generally not provided

[4]The core network, also known as backbone, connects different access networks with one another. Each access network comprises end devices and edge nodes, with the latter providing the former with an entry point to the core network.

by a single resource-rich computer. Instead, the service is typically decomposed and provided by a hierarchy of FNs such that each of them runs a specific portion of the overall service, while cooperating with the other FNs. This pyramid-like organisation is one of the guiding principles of the OpenFog Reference Architecture (OFRA) (Consortium, 2017), as discussed in Section 2.6. However, as stated by the OFC (Consortium, 2017), "*computational and system hierarchy is not required for all OpenFog architectures, but it is still expressed in most deployments*".

As shown in Figure 2.1, the lowest layer in the hierarchy comprises the Things and the end devices in general, which might themselves behave as FNs if they are powerful enough. The higher layers lead from the network edge up to the core, and their number and composition depends on the actual application domain and purpose (Consortium, 2017). Finally, the topmost layer might be represented by the Cloud. Indeed, - and this is of paramount importance - FC does not replace the Cloud, but typically coexists and cooperates with it, as many services require the characteristics of both the Fog and the Cloud (Bonomi et al., 2012). Interactions in such hierarchical systems may be of any type, both within the same layer and among nodes belonging to different layers (Consortium, 2017). Each node makes its own contribution to the overall service, and the nature of its role highly depends on its position in the pyramid. This is summarised in Table 2.3, which is the result of an integration of coverage across (Bonomi et al., 2012; Consortium, 2017; Cisco, 2015).

This hierarchical organisation, together with proximity to end devices, is the main characteristic of FC, which makes it particularly suitable for the IoT. The IoT domain is often identified by wide-area deployment of sensors and actuators that

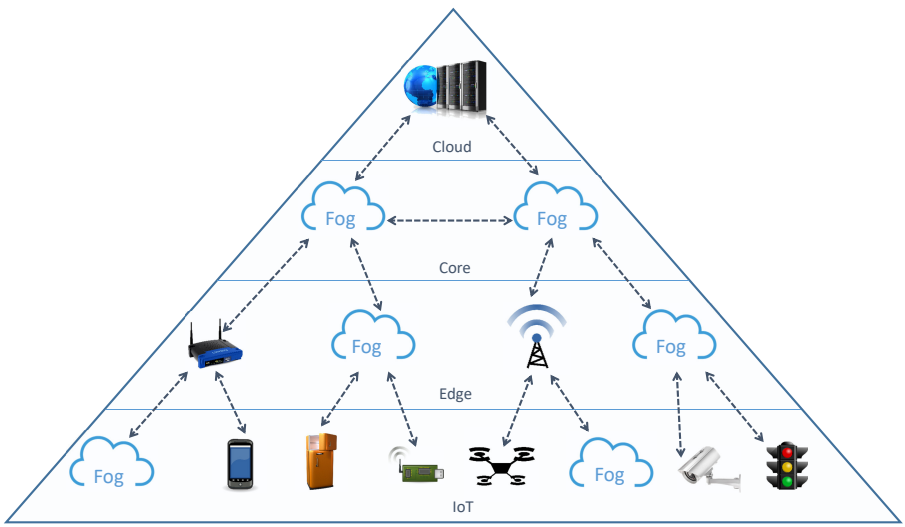


Figure 2.1: FC hierarchical organisation.

Table 2.3: Nodes have different properties and roles according to their position in the hierarchy.

	FNs closest to the IoT	**FNs at the core network**	**Cloud**
Fog benefits (see Table 2.2)	The FC advantages are evident.	They become less evident.	They are null.
Geographical coverage	These FNs are widely distributed in order to ensure close proximity to the Things. Hence, each of them covers a small area, controlling few IoT devices.	The farther from the true edge, the fewer the FNs. Therefore, each of them covers a rather wide geographical area.	CC resources are highly concentrated in few DCs all over the world. Thus, the Cloud features a global coverage, as each DC has to manage a huge area.
Data persistence	Time-sensitive data are sent to these FNs for instant (i.e., O(milliseconds)) decision-making and actuation. Hence, such data are transient.	Data which can wait (seconds to minutes) from the time of sensing to that of actuation are sent to these FNs.	Data persist in the Cloud for days, weeks, or even months for historical analysis.
Computing power	These FNs are typically the least powerful, as they have to process transient data from a limited area.	The higher the level in the pyramid, the more powerful the nodes. There is therefore a need to process more persisting data from a wider geographical area.	The Cloud is the most powerful. Furthermore, the insights realisable in the Cloud are the greatest due to the size of datasets available.
Contribution	These FNs collect the data, process them, and issue actuation commands. They may also filter the data to be kept locally and transmit the rest to the higher layers. Thus, the only type of interaction at this level is Machine to Machine (M2M).	These nodes typically perform data filtering, compression, and transformation. They may also issue less time-sensitive commands to the actuators. Finally, they can provide visualisation and reporting services to end users. Hence, this level features both M2M and Human to Machine (HMI) interactions.	The Cloud collects data from hundreds or thousands of nodes. It performs long-term storage, historical analysis and forecasting, and Big Data analytics. The Cloud typically interacts with the final users for insights delivery, although also IoT nodes might directly communicate with it.

can cover areas of hundreds or more square miles. Moreover, IoT applications and services are always more complex, as they may involve aspects such as: time-critical

control, visualisation and reporting, and historical analysis of Big Data. Spanning from the Things up to the Cloud, the Fog hierarchy enables all this. Examples of FC hierarchies applied to transportation systems and to the food processing plant can be found in (Consortium, 2017), while (Bonomi et al., 2012) reports an example related to Smart Grids.

Horizontal paradigm

FC can also be viewed "horizontal" in the sense that it is generic enough to be applied in a number of different application scenarios, e.g., content delivery, gaming, network control functions (Zhu et al., 2013; Fan et al., 2016; Lai et al., 2016). However, this survey only focuses on the contribution of FC to the IoT.

2.3 Historical background

FC is an evolution of early proposals with the objective to best answer the needs of the IoT. This section explores the Fog and the "similar concepts" from an historical perspective, with the purpose to clarify the reasons that led to the characteristics of each of these concepts and focus more on their similarities rather than on their differences.

It all began in the early 2000s with a big contradiction in one of the most emerging trends of that period: *Mobile Computing*. On the one hand, mobile devices have the potential to make emerging services in several fields (e.g., healthcare, gaming, entertainment, social networking) always available; though, on the other hand, they usually have limited computing capabilities, as they have to be often light and small and require a long battery life (Dhingra, 2014). Therefore, it is difficult for them to provide resource-intensive services by just relying on their own facilities.

Hence, how is it possible to release the full potential of Mobile Computing despite its limitation? In 2001, Mahadev Satyanarayanan (professor of Computer Science at the Carnegie Mellon University) proposed the concept of *Cyber Foraging* as a possible solution to the problem (Satyanarayanan, 2001). This paradigm suggested to offload data and intensive computation from a mobile device onto a more powerful server belonging to the fixed infrastructure. Such a server was supposed to be in close proximity to the associated mobile node, but this assumption was not made explicit by Prof. Satyanarayanan at that time. Although Cyber Foraging is the real ancestor of FC, other paradigms bringing content or computation closer to the end devices, such as Content Delivery Networks (CDNs) (Pathan and Buyya, 2007) and in-network processing (Chen et al., 2006), were being proposed in those years.

Among the many open issues raised by Cyber Foraging, one was particularly tricky: who and why should have made those servers available? The answer to this question was found few years later with the introduction of CC, whose characteris-

tics have been already discussed in the Introduction of this thesis. The integration between Mobile Computing and CC is referred to as *Mobile Cloud Computing* (MCC) (Fernando et al., 2013).

Although MCC was a promising paradigm, it presented all the limitations discussed in Section 2.1. Therefore, in 2009, Satyanarayanan et al. suggested to cope with such shortcomings (and in particular with the high and unpredictable latencies) through the concept of *Cloudlet* (Satyanarayanan et al., 2009), which was the de facto birth of a paradigm known as *Mobile Edge Computing* (MEC) (Mach and Becvar, 2017). A Cloudlet is defined as "*a trusted, resource-rich computer or cluster of computers that is well-connected to the Internet and available for use by nearby mobile devices*". A resource-constrained mobile device can behave as a thin client and, rather than relying on the distant Cloud, can offload all the significant computation onto a nearby Cloudlet located at the network edge. This still provides all the benefits of CC, such as virtualisation and efficiency, though without the characteristic delays. If no Cloudlet is present nearby, the mobile device can temporarily rely on the Cloud as a fallback option or, in the worst case, on its own resources (Satyanarayanan et al., 2009). More in general, the use of Cloudlets to support any type (i.e., either mobile or fixed) of resource-limited end devices or groups of devices is simply referred to as *Edge Computing* (EC) (Horwitz, 2017).

Since MEC emerged as a worthy solution to enable computation-intensive mobile applications, the European Telecommunications Standards Institute (ETSI) created an Industry Specification Group (ISG) in 2014 with the purpose to define and integrate a standard implementation of MEC into cellular networks, which was called ETSI Mobile Edge Computing (ETSI MEC) (Hu et al., 2015). According to the ETSI, such a standard lets operators "*open their Radio Access Network* (*RAN*) *edge to authorised third-parties, allowing them to flexibly and rapidly deploy innovative applications and services towards mobile subscribers, enterprises and vertical segments*" (ETSI, 2018). More recently, the ETSI renamed ETSI MEC in *Multi-Access Edge Computing* to emphasise the novel intention to also address non-cellular operators' requirements (ETSI, 2017b).

Finally, in order to clarify the last step towards FC, it is fundamental to highlight the following aspect. At least in its infancy, MEC did not consider the overall service to be decomposed and provided by a hierarchy of nodes including also the Cloud; instead, the whole service is entirely provided by a nearby Cloudlet (if available), as we have already mentioned. This is why the OFC states that "*fog works with the cloud, whereas edge is defined by the exclusion of cloud. Fog is hierarchical, where edge tends to be limited to a small number of layers*" (Consortium, 2017). This characteristic of MEC is reasonable in the context of Mobile Computing, where an application typically involves a single user. However, the IoT is often defined by sensors and actuators covering wide areas and by the need for long-term storage and Big Data analytics (i.e., all elements that may require the Cloud). At the same time, proximity is nec-

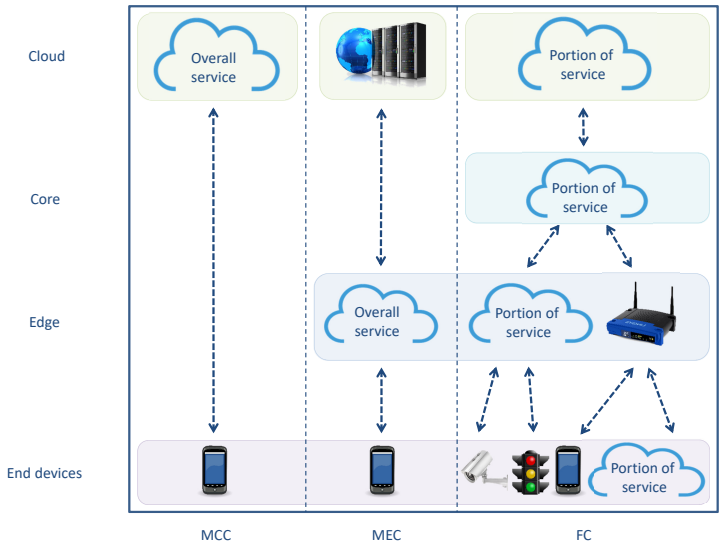


Figure 2.2: A comparison among the definitions of MCC, MEC, and FC.

essary in order to enable low and predictable response times together with all the other benefits reported in Table 2.2 (which require resources towards the network edge). As a result, in order to best suit such requirements, Cisco advanced the FC paradigm in 2012 (Bonomi et al., 2012) as a generalisation of EC in which it may still happen that a single, closer resource-rich computer provides the overall service, but most of the times any resource in the Cloud-to-Things continuum provides only a portion of the overall service, according to the facilities and position in the pyramid (see Section 2.2).

Figure 2.2 illustrates and compares the original definitions of MCC, MEC, and FC. The research community often tends to look for the differences between FC and EC. However, it might be more fruitful to emphasise the several similarities between these two paradigms. Indeed, on the one hand, they were born in different moments and were specifically conceived for different contexts, but, on the other hand, they are evolving over time towards an inevitable convergence (Satyanarayanan, 2017; Satyanarayanan et al., 2015, 2014). As a proof of this, the ETSI and the OFC recently signed a Memorandum Of Understanding (MOU) with the intent to join forces for the development of Fog-enabled Mobile Edge applications and technologies (ETSI, 2017a).

2.4 IoT application domains

As detailed in Section 2.2, FC proves to be a promising paradigm to support the IoT. Taking inspiration from the classification found in (MarketsandMarkets, 2016), this section organises the IoT applications into six domains. Overall, we found 45 works proposing an integration between FC and the IoT in one of those categories[5]. We merely report these six domains from the most to the least investigated in terms of number of published papers. More specifically, the most examined is the Intelligent Transportation Systems (ITS) domain (12 papers, 26.7%), followed by Smart Healthcare (11 papers, 24.4%). Next, there are the public safety sector (7 papers, 15.5%), Smart Grids (6 papers, 13.3%), and Industry 4.0 (5 papers, 11.1%). Finally, Smart Homes and Buildings conclude the list (4 papers, 8.9%). The objective is to highlight how each of these domains may benefit from FC and provide a comprehensive overview of the state of the art in the employment of the Fog within each of them. Table 2.4 summarises the main aspects of each considered work by: (i) outlining the major contribution with keywords; (ii) reporting which devices are employed as FNs; and (iii) pointing out the maturity level of the proposal. The maturity level may be one of the following: *Theory*; *Simulation*; *Prototype*; *Pre-product* (i.e., already available for use but still under active development); and *Product*.

[5]We consulted the main scientific literature databases and search engines (i.e., IEEE Xplore, ACM library, ScienceDirect, and Google Scholar) from August 2017 to January 2018. Search queries were formulated in order to be as comprehensive as possible within each considered application domain. For example, the following is the search query defined for the ITS domain: (*Fog Computing OR Edge Computing*) *AND* (*ITS OR vehicle OR RSU OR traffic OR road OR transport OR driver OR parking*). Through this methodology, we found contributions whose publication years are not earlier than 2014. Finally, with the aim to consider only the most relevant and recent works, we filtered the obtained results as follows: (i) given two similar works from the same group of authors, of which one is a conference paper and the other is a journal article, we selected the latter; (ii) in case these two works are both conference papers or journal articles, we selected the most recent one.

Table 2.4: Papers employing the Fog in one of the considered IoT application domains.

Domain	Paper	Keywords	FNs	Maturity
ITS	Tanganelli et al. (2017)	Look-up service; DHT	n/a	Prototype
	Kim et al. (2015)	Parking; Matching theory	n/a	Simulation
	Liu et al. (2017a)	Architecture for urban traffic management; SDN; 5G	Cellular base stations	Simulation
	Liu et al. (2017b)	Architecture for urban traffic management; SDN; 5G; IEEE 802.11p	Cellular base stations	Simulation
	Truong et al. (2015)	Architecture; SDN; Data streaming; Lane change	Cellular base stations; RSUs; Road Side Unit Controllers (RSUCs)	Theory
	He et al. (2016)	Architecture for load balancing; SDN	Cellular base stations; RSUs	Simulation
	Ge et al. (2017)	Architecture; SDN; 5G	Cellular base stations; RSUs; vehicles; RSUCs	Simulation
	Shin et al. (2016)	Architecture for urban traffic management; Pub-Sub; Semantic Web	n/a	Theory
	Brennand et al. (2016)	Urban traffic management	RSUs	Simulation
	Bruneo et al. (2016)	Stack4Things; Complex Event Processing	Single-board computers	Prototype
	Hou et al. (2016)	Vehicular Fog Computing	Vehicles; cellular base stations; RSUs	Simulation
	Ye et al. (2016)	Service offloading in bus networks; Genetic algorithm	Vehicles (i.e., buses); RSUs	Simulation
Smart Healthcare	Sareen et al. (2017)	Zika virus; fuzzy k-nearest neighbor	n/a	Prototype
	Mei et al. (2017)	UV radiation measurement; Android	n/a	Prototype
	Cao et al. (2015)	Fall detection; Android	Smartphones	Prototype
	Fratu et al. (2015)	COPD patients; Mild dementia	n/a	Prototype
	Masip-Bruin et al. (2016)	COPD patients; Dynamic adjustment of the oxygen dose	Portable oxygen concentrators; gateways	Prototype

Table 2.4: Papers employing the Fog in one of the considered IoT application domains.

Domain	Paper	Keywords	FNs	Maturity
Smart Healthcare	Zao et al. (2014)	Brain monitoring; Semantic Web	Personal Computers; home gateways	Prototype
	Ali and Ghazal (2017)	Heart attack; vehicular networks; SDN	Cellular base stations; RSUs; RSUCs	Prototype
	Monteiro et al. (2016)	Parkinson's disease; speech treatments	Embedded systems	Prototype
	Ahmad et al. (2016)	Security and privacy of health-related data; CASB	n/a	Prototype
	Elmisery et al. (2016)	Security and privacy of health-related data	Personal gateways	Prototype
	Rahmani et al. (2017)	Smart e-Health Gateway	Gateways in a Smart Home or hospital	Prototype
Public safety	Sapienza et al. (2016)	Critical events in a Smart City	Cellular base stations	Theory
	Rauniyar et al. (2016)	Disaster management; crowdsourcing	n/a	Theory
	Mayer et al. (2017)	Architecture for social sensing services in hostile environments	n/a	Theory
	Brzoza-Woch et al. (2016)	Smart levee monitoring system	Industrial controllers; single-board computers	Prototype
	Motlagh et al. (2017)	Crowd surveillance; UAVs	Cellular base stations	Prototype
	Dautov et al. (2017)	Intelligent surveillance system	Smart cameras	Prototype
	Chen et al. (2016)	Smart urban surveillance; target tracking	Tablets; smartphones; laptops	Prototype
Smart Grid	Yan and Su (2016)	Smart metering infrastructure; Big Data	Smart meters	Prototype
	Nazmudeen et al. (2016)	Data aggregation for bandwidth efficiency; Power Line Communication	Routers	Simulation
	Beligianni et al. (2016)	Data aggregation for preserving privacy of energy consumption	n/a	Theory
	Han and Xiao (2016)	Algorithm to detect NTL fraud	n/a	Simulation

Table 2.4: Papers employing the Fog in one of the considered IoT application domains.

Domain	Paper	Keywords	FNs	Maturity
Smart Grid	Yaghmaee et al. (2017)	Power consumption schedule; Demand Side Management	n/a	Simulation
	Tao et al. (2017)	V2G; EVs; 5G	EVs; local aggregators; control centres	Simulation
Industry 4.0	de Brito et al. (2017)	Docker-based service orchestration; oneM2M; P2P communications	n/a	Simulation
	Suto et al. (2015)	Energy-efficient FNs for industrial WSNs	Servers in a Wireless Computing System	Simulation
	Peralta et al. (2017)	Reduction of sensor energy consumption; MQTT	IoT gateways operating as MQTT brokers	Simulation
	Wu et al. (2017)	Machine health and process monitoring	Gateways in factory floors	Prototype
	Nebbiolo (2017)	FC platform tailored to the industrial automation sector	Modular computers	Product
Smart Home and Smart Building	Vallati et al. (2016)	Awareness of the home context; Device-to-Device	Home gateways; set-top boxes; end user devices	Simulation
	Dutta and Roy (2017)	A single FN for the whole building	Wi-Fi routers	Prototype
	Seitz et al. (2017)	FNs in multiple rooms of a building	n/a	Theory
	Liu et al. (2016b)	FC platform tailored to the Smart Home and Smart Building domain	Wi-Fi access points; set-top boxes	Pre-product

Intelligent Transportation Systems

The world urban population is dramatically increasing. At present, the number of megacities (i.e., cities with a population exceeding 10 million people) is 28 and is projected to reach 41 by 2030 (United Nations, Department of Economic and Social Affairs, Population Division, 2014). As a consequence of this, urban environments are more and more overcrowded with vehicles, and traffic congestions, time losses, accidents, and pollution altogether contribute to a non-negligible reduction in the experienced safety and Quality of Life (QoL). The employment of Information and Communication Technologies (ICT) within the transport domain gives birth to the ITS, where a wide range of services and applications may be conceived in order to face the aforementioned issues (Dimitrakopoulos and Demestichas, 2010). Hence, ITS allow to considerably improve traffic efficiency, drivers' and passengers' safety,

and freight transport.

FC can play a crucial role in this context (Kai et al., 2016). Indeed, as we have already mentioned in Section 2.1, road safety and autonomous driving services require response times to be lower than 50 ms (Schulz et al., 2017), which usually is not achievable with CC. Furthermore, as it is described in (Consortium, 2017), FC: (i) saves bandwidth, by avoiding that all the data collected by vehicles and by the fixed infrastructure are sent up to the Cloud; (ii) provides critical ITS services also in the presence of intermittent network connectivity towards the Cloud; and (iii) allows FNs to provide context-aware services to the vehicles in their proximity (e.g., alerting them of bad road conditions in that area). In (Tanganelli et al., 2017), the authors propose a look-up service for ITS based on a Distributed Hash Table (DHT) to be implemented by FNs. A Fog system to help drivers to find a free parking slot is presented in (Kim et al., 2015). Such a system features a pyramid-like organisation so that the more towards the network edge a FN, the smaller its coverage area, but the higher its context awareness.

Several works (Liu et al., 2017a,b; Truong et al., 2015; He et al., 2016; Ge et al., 2017; Shin et al., 2016) propose distinct Fog-based architectures for ITS. Except for (Shin et al., 2016), they all employ FC together with Software Defined Networking (SDN), which provides network flexibility and programmability. The resulting architectures are thus organised into four layers: (i) CC; (ii) SDN control; (iii) FC; and (iv) Infrastructure layer, which comprises the sensing and actuation nodes. Moreover, the architectures in (Liu et al., 2017a,b; Shin et al., 2016) are either validated or specifically envisioned for urban traffic management and control, which is the ITS major concern. Traffic management is the cornerstone also in (Brennand et al., 2016) where the authors propose FOX, a Fog-based system whose objective is to detect and minimise traffic congestions. Finally, in (Bruneo et al., 2016), the authors propose Stack4Things as a FC platform for Smart City applications. They exploit Cloud-based network virtualisation functionalities to implement a smart mobility use case in which smart cars can interact with Smart City objects to implement geolocalised services. For example, smart cars approaching intersections are able to communicate with smart traffic lights in order to acquire a certain level of priority with respect to other cars.

In (Consortium, 2017), traffic control is one of the reported use cases for FC. This paper, unlike the others that have just been introduced, points out an interesting aspect: the vision of vehicles as FNs and not only as sensing and actuation devices. This is further discussed in (Hou et al., 2016) where the authors present Vehicular Fog Computing (VFC) to exploit and aggregate the great amount of underutilised resources in nearby vehicles, together with those belonging to the fixed infrastructure, such as cellular base stations and Road Side Units (RSUs), to provide services of computation, storage, and networking. As a result, parked and slow-moving vehicles form a FC layer to enable several vehicular services and applications. To con-

clude, (Ye et al., 2016) might be considered as a particular case of (Hou et al., 2016), as the authors propose to extend the computing capability of the fixed infrastructure at the network edge by utilising buses and bus networks. The main reason for this is that the fixed trajectories and strong periodicity of buses are ideal in this direction.

Smart Healthcare

The healthcare domain is one of the toughest and most delicate as it deals with people's lives. The Internet of Healthcare Things (IoHT), together with CC, allows to envision several services for the improvement of patients' QoL. However, a simple sensor-to-Cloud architecture proves to be often too reductive and unsuitable for many emerging healthcare applications with critical requirements. FC can be the solution to the problem (Kraemer et al., 2017; Farahani et al., 2017), especially but not only in the following three ways: (i) it enables low and predictable response times, which can often make the difference between life and death for patients; (ii) it ensures that at least the most critical portion of the overall service is always available to the patient, also in the presence of hostile environments with intermittent or no network connectivity to the Cloud; and (iii) it protects the health-related sensitive data by keeping them locally (e.g., in a FN located within the hospital or the patient's house) rather than sending them to the Cloud through the Internet.

In (Sareen et al., 2017), the authors propose a Fog-based system for predicting and preventing the Zika virus outbreak. The Fog layer performs real-time processing of environmental sensor data as well as symptoms data collected by the users' smartphones. In (Mei et al., 2017), the authors conceive a service that takes advantage of the FC context awareness due to the proximity to users' smartphones in order to provide accurate and localised measurements of Ultraviolet (UV) radiations. Falls are among the major causes of mortality for stroke patients. Therefore, it is of vital importance to promptly detect falls and intervene. U-Fall (Cao et al., 2015) is a FC system to achieve this objective: the patient's smartphone behaves as the FN for a quick fall detection; sensor data are also transmitted to the distant Cloud for long-term storage and analysis. Some works propose to adopt FC in order to improve the QoL of Chronic Obstructive Pulmonary Disease (COPD) patients. In this context, the authors in (Fratu et al., 2015) extend the eWALL Cloud-IoT system[6] by implementing the Fog layer. A further contribution in this direction is made in (Masip-Bruin et al., 2016) where the authors propose to assist COPD patients also when these are performing physical exercises. To this aim, the oxygen dose is dynamically adjusted also to the patient's context and needs; hence, FC context awareness is required. The Fog can be similarly applied to enable services that monitor brain activity in, for example, stressed or Parkinson's disease patients (Zao et al., 2014). In (Ali and Ghazal, 2017), the authors propose a service exploiting resources

[6]See `http://ewallproject.eu/`. Last accessed: 12 October 2019.

at the network edge and SDN for the real-time detection of heart attacks in drivers. FIT is a FN conceived in (Monteiro et al., 2016) that preprocesses the speech data of a patient with speech impairments and forwards speech features to the Cloud in order to reduce the required bandwidth and computational burden on the Cloud.

Patients' health-related sensitive data need to be preserved and protected: FNs may behave as privacy and security enforcement points. In this direction, a Cloud Access Security Broker (CASB) may be executed at the Fog layer as in (Ahmad et al., 2016). Similarly, the authors in (Elmisery et al., 2016) develop an Enhanced Middleware for Collaborative Privacy (EMCP) to be hosted on FNs. More in general, the authors in (Rahmani et al., 2017) present UT-GATE, the prototype of a FN that provides the healthcare domain with all the benefits typical of FC.

Public safety

FC and the IoT are relevant paradigms also from the viewpoint of public safety and well-being. For example, in (Sapienza et al., 2016) the authors present a Fog-IoT architecture with this purpose. In order to guarantee public safety, two tasks have to be effectively performed: disaster management and crowd surveillance.

Natural or man-made disasters usually cause significant human, economic, and environmental damages. According to (IFRC, 2016), more than 6000 disasters happened in the last 10 years, causing almost 772,000 people killed, 1,917,557 somehow affected, and a total estimated damage equal to $ 1,424,814 million. Therefore, properly managing these situations is of vital importance. In this direction, the authors in (Rauniyar et al., 2016) propose a Fog-based architecture where crowd-sourced data are communicated to the Fog for quick processing and decision-making. FNs store emergency contact numbers and are accessible by the local public safety authorities that can plan rescue actions according to the produced insights. At the same time, affected people may contact the nearby FN in order to efficiently obtain crowd-sourced pictures and videos, thus to have an idea of the current situation. Both natural and man-made disasters may cause Internet connectivity to be unstable. Despite this, having uninterrupted access at least to the most critical part of the service is a must in such delicate situations. As we reported in Section 2.2, FC provides this important feature (Mayer et al., 2017). The authors in (Brzoza-Woch et al., 2016) present a levee monitoring use case involving the Fog. They conceive a three-layered architecture where edge nodes may: (i) locally make decisions; (ii) collaborate with one another; and (iii) optionally return preprocessed (i.e., filtered and/or compressed) results to the Cloud for further analysis and forecasting. Different versions of this system exist to best suit diverse environmental, infrastructure, and economic conditions.

Crowd surveillance is essential to guarantee public safety. Indeed, it allows for example to: (i) identify non-authorised accesses and suspicious activities; (ii) de-

tect the fall of an elderly or infirm person; (iii) pinpoint a terrorist or criminal; and (iv) find a missing person. The suitability of FC to crowd surveillance is evident, as the Fog grants low and predictable response times, bandwidth efficiency, and privacy preservation (Consortium, 2017). Taking this into consideration, the authors in (Motlagh et al., 2017) propose an Unmanned Aerial Vehicle (UAV)-based IoT platform and present a use case where drones transmit surveillance videos to edge nodes that locally perform face recognition tasks. Similarly, in (Dautov et al., 2017), the authors present a case study based on a distributed intelligent surveillance system scenario in a crowded area, implemented on clustered Fog devices that are able to horizontally offload tasks among themselves. To conclude, (Chen et al., 2016) discusses an urban speeding traffic monitoring system using FC. A drone monitors moving vehicles by recording a surveillance video that is sent back to the drone controller on the ground and displayed on a screen. If the police officer finds a vehicle moving at a suspicious speed, the system forwards the next video frames to a FN in order to track that vehicle.

Smart Grid

The traditional electrical grids distribute energy from few central power generators to a very large number of final customers. The Smart Grid is an evolution of the traditional power grid, as it is the result of the integration between the latter and the ICT (Fang et al., 2012). In a Smart Grid, energy is generated by several widely-distributed stations, and smart meters and other sensor nodes are employed to monitor and control the energy consumption. As a result, there is a continuous, bi-directional flow of both electricity and data that allows to conceive services for a more efficient, reliable, and secure energy management. Such services may greatly benefit both: (i) the electricity suppliers, e.g., to efficiently deliver and manage energy; (ii) the final customers, e.g., to easily monitor and/or reduce their energy consumption.

As it has been just mentioned, Smart Grids are characterised by a strong distribution of power generators, energy transformers, sensors, and actuators: it is not uncommon for a Smart Grid to cover an area of hundreds square miles. Moreover, Smart Grid sensors produce a vast amount of data, which can easily saturate network, storage, and processing resources. To further complicate matters, smart meters data may be exploited to deduce personal information (e.g., the number of people in a specific area, the habits of a family); therefore, privacy in Smart Grids is an important issue (Okay and Ozdemir, 2016). Last but not least, many Smart Grid services require quick and predictable response times, typically between three and 20 milliseconds (Schulz et al., 2017). All these features make Smart Grids an ideal domain where to apply FC.

Several works employ FC in Smart Grids. The authors in (Yan and Su, 2016)

propose a Fog-based Smart Grid solution where smart meters are grouped to form computing and storage clusters, thus realising a Fog layer at the extreme network edge. A hierarchy of FNs in the Smart Grid context may perform data aggregation (i.e., data are gathered and expressed in a summary form) in order to reduce the amount of data transmitted to the Cloud and thus save bandwidth (Nazmudeen et al., 2016). Data aggregation carried out by FNs can also preserve the privacy of customers' energy-related data (Beligianni et al., 2016). The authors in (Han and Xiao, 2016) propose a security analytic algorithm to be executed by cooperating FNs for the detection of Non-Technical Loss (NTL) fraud in Smart Grids. An attacker performs NTL fraud by tampering with a smart meter so that it reports fake energy consumption values. The proposed iterative algorithm divides the overall problem in sub-problems and assigns each of them to a FN; the solution to the overall problem is given by the local solutions of the sub-problems.

Some works are more application-oriented. The authors in (Yaghmaee et al., 2017) present a Fog-based approach for the optimisation of the power consumption schedule in Smart Grids, which results in an optimisation of both customers' and electricity supplier's costs. In more detail, the Smart Grid is organised in regions, and each region is managed by an edge node that finds the optimal power consumption schedule for its region, based on the collected data. The centralised Cloud is then responsible for the optimisation of the energy consumption schedule at a multi-regional level. To conclude, Vehicle to Grid (V2G) is an emerging set of services that allows Electric Vehicles (EVs) to both consume and return back electricity from/to the Smart Grid. Foud (Tao et al., 2017) is a computing model integrating the Cloud, the Fog, and 5G technologies in order to improve V2G services. In Foud, EVs may be both final users and components of the Fog layer, which is said to be temporary due to the vehicles mobility.

Industry 4.0

Since its very beginning in the late 18th century, industrial production has experienced several revolutions that have deeply changed its nature. First, mechanisation driven by steam power made its entrance. The second industrial revolution consisted in electrification and mass production, while the third era of industry started in the 1960s with the digital programming of automation systems. Nowadays, we are undergoing the fourth industrial revolution, which is either known as Industry 4.0, Smart Factory, or Smart Manufacturing. All these terms identify the same revolutionary trend: the employment of the IoT and, more generally, Cyber-Physical Systems (CPSs) in industrial automation for a smarter production (Drath and Horch, 2014).

Thanks to its advantages, FC may be the solution to several challenges raised in this context (Breivold and Sandstrom, 2015). In particular, the Fog is very useful

within a Smart Factory in order to satisfy the latency requirements that characterise such a context. Typically, these requirements are the most stringent among all the investigated domains, as they vary from from 250 µs to 10 ms. An exclusive reliance on the Cloud would not allow to respect such stringent latency requirements.

The authors in (de Brito et al., 2017) propose a solution based on the oneM2M technical specifications[7] that enhances peer-to-peer (P2P) communications between FNs and implements a Docker-based service orchestration mechanism in the industrial domain. The authors in (Suto et al., 2015) present a system for industrial Wireless Sensor Networks (WSNs) that minimises the power consumption, by controlling the FNs sleep scheduling and network connectivity, while satisfying the time constraints imposed by Smart Manufacturing applications. A Fog architecture is described in (Peralta et al., 2017) where FNs are IoT gateways operating as Message Queue Telemetry Transport (MQTT)[8] brokers able to predict future sensor measurements. As a result, sensors need to publish their data only in case of wrong predictions by the broker; this helps to reduce their power consumption while keeping latencies low. The authors in (Wu et al., 2017) discuss a Fog-based architecture for machine health and process monitoring in cyber-manufacturing systems. To conclude, Nebbiolo Technologies (Nebbiolo, 2017) launched a FC platform for the industrial automation sector; we will discuss it in Section 2.6.

Smart Home and Smart Building

FC is progressively entering the home context; in-home devices (e.g., home gateways, set-top boxes, end user devices) may behave as FNs, as they are becoming increasingly powerful, and virtualisation techniques are more and more efficient (Vallati et al., 2016). Smart Homes will enormously benefit from this trend. Indeed, response times would be further reduced, which is essential for time-sensitive Smart Home systems such as those who deal with surveillance and access control. Moreover, the presence of a FN in the house would ensure resilience when there is no Internet connectivity to the Cloud. Last but not least, privacy and bandwidth efficiency would be both improved, as the many (sensitive) data collected would be mainly kept within the house.

Similarly, FNs may be also present inside buildings to enable improved Smart Building services. Depending on the actual needs, there could be a single FN for the whole building, or there could be an internal hierarchy with a FN for each floor or even one for each room (Consortium, 2017). The authors in (Dutta and Roy, 2017) propose a Smart Building system with a single FN for the whole building. The FRODO architecture proposed in (Seitz et al., 2017) is more sophisticated, as FNs may be deployed in multiple rooms of a building for decentralised decision-making.

[7] See `http://www.onem2m.org/`. Last accessed: 12 October 2019.

[8] MQTT is a Publish-Subscribe lightweight messaging protocol. See `http://mqtt.org/`. Last accessed: 12 October 2019.

Each of them provides highly context-aware services to the occupants of its room, taking into account their personal preferences together with objective, room-related parameters (e.g., the type of sensors and actuators present). Last but not least, the authors in (Liu et al., 2016b) present ParaDrop, a FC platform that allows to manage and deploy services on wireless gateways (e.g., Wi-Fi access points, set-top boxes). This platform, which particularly suits the Smart Home and Smart Building domain, will be further detailed in Section 2.6.

2.5 Research challenges

New system, network, and environmental characteristics need to be considered when extending the Cloud towards the network edge – see Table 2.5. This section specifically focuses on challenges associated with these characteristics, identifying how the research community is addressing them. Please, note that the mobility support challenge, which is the main focus of this thesis, is thoroughly discussed in Chapter 3.

Table 2.5: Characteristics to be considered when extending the Cloud towards the network edge.

Characteristic	**Description**	**Introduced or influenced challenges**
Geographical distribution	FC leads from a situation in which resources and services are all concentrated in a Cloud DC to one in which they are distributed over a potentially wide area.	Mobility support; orchestration; deployment models; security and privacy
Higher heterogeneity	While Cloud servers are all very alike, FNs are usually heterogeneous, as they might feature different hardware specifications and capabilities, operating systems, or protocol suites (Varghese et al., 2017).	Orchestration; deployment models; security and privacy
Computing power	As reported in Table 2.3, FNs are in general less powerful than Cloud servers. However, there exists a wide range of diverse FNs with very different hardware capabilities, as outlined in Table 2.8.	Mobility support; orchestration; security and privacy
Network performance	While a Cloud DC relies on a high-bandwidth and low-latency LAN, FNs are typically interconnected with each other through a WAN and hence experience higher latencies with respect to those within a Cloud DC and an average bandwidth of 13 Mbps (Ha et al., 2017)[9].	Mobility support; orchestration
Vulnerable environment	With the aim to be closer to IoT devices, FNs are usually located in environments that are more vulnerable and less protected than Cloud DCs (Chiang and Zhang, 2016).	Deployment models; security and privacy

Orchestration

To orchestrate computing resources and services means to coordinate, arrange, and jointly manage them in order to satisfy specific functional and non-functional requirements. For instance, service deployment and resource allocation, service coordination, and load balancing are all orchestration activities. A suitable orchestration is essential in every complex system in order to ensure efficiency and efficacy.

Resource and service orchestration is influenced by some of the distinguishing characteristics of FC (see Table 2.5). As a result, the orchestration techniques that are widely adopted in a Cloud DC cannot be always applied "as is" in the Fog but need to be customised for it (Jiang et al., 2018). Let us begin by examining the impact of FC distinctive features on orchestration. Firstly, the heterogeneity of FNs imposes non-trivial orchestration issues (Wen et al., 2017). It is fundamental to consider this diversity when deploying and coordinating services, since not all FNs are able to run all services. Two FNs that are identical in terms of hardware and software capabilities may be very different from one another due to their geographical distribution featured by FC and the requirement of topological proximity. For instance, it may happen that only one of them is suitable to host a specific Fog service, as the other may be not close enough to the IoT devices requiring that service. To further complicate matters, the high distribution featured by the Fog and its hierarchical nature imposes other challenges, namely those regarding the management of large data volumes. Indeed, these are not only exchanged with a centralised Cloud DC but typically need to be orchestrated among the nodes along the continuum from Cloud to Things according to the nature and purpose of these data, the specific application scenario, and its requirements. Furthermore, the potentially wide-area distribution of Fog services and resources, together with the need for scalability and the strict requirements of the IoT, naturally arises from centralised orchestration as in the Cloud DC to a distributed one where multiple orchestrators are arranged according to a hierarchical or flat architecture (Jiang et al., 2018), and where each of them directly controls only a subset of nodes. Such orchestrators have to strongly coordinate with one another for the joint management of complex and distributed IoT applications. Moreover, the introduction of a great number of distributed FNs composing the Fog layer causes FC environments to be in general less energy-efficient than Cloud-only environments (Dastjerdi et al., 2016). More specifically, the energy consumed by FNs represents the 60-80% of the overall energy consumed by systems that span from the things up to the Cloud (Nan et al., 2017). To conclude, orchestration should be dynamic, i.e., should perform smart re-configurations in order to adapt to the continuous changes that occur in the system. FC environments are highly dynamic (Wen et al., 2017) due to their intrinsic limitations in terms of computing power and network performance with respect to those in the Cloud DC and because of the strict requirements of IoT applications in terms of Quality of Service (QoS) and Quality of Experience (QoE). In what follows, we identify Fog orchestra-

tion architectures and policies that have been proposed in literature and conclude with the main open issues in the field.

Architectures - The authors in (Hoque et al., 2017) first analyse how the existing container orchestration tools address the requirements of FC and the IoT. Based on the obtained insights, they propose a container orchestration framework that bridges the found gap. Such a framework extends Docker Swarm[10], which is extremely lightweight and rather complete, with an additional component called OpenIoTFog Agent, which is part of the OpenIoTFog toolkit[11]. The same authors detail an orchestration architecture for Fog environments in (Brito et al., 2017). This architecture is based on two essential components, namely the Fog Orchestrator (FO), which runs on a central node, and the Fog Orchestration Agent (FOA), which runs on every FN. It is worth highlighting that in those cases in which there is no connectivity towards the FO, a FOA can become a FO for a subset of FNs. It will return a simple FOA if and when the connection to the central node resumes. Moreover, this architecture presents two main strengths. The first is the compliance with the recently released OFRA, which is discussed in Section 2.6. The second is its conformity with Topology and Orchestration Specification for Cloud Applications (TOSCA)[12], which is the de facto standard for modeling service orchestration. The authors in (Yigitoglu et al., 2017) propose Foggy, a framework for dynamic resource provisioning and IoT applications deployment in FC environments. The orchestration server, which runs on a central node and manages the whole system, obtains each application module requirements (i.e., priority, privacy, computation, latency, output) in JSON format and continuously monitors the system in order to capture every dynamic change and adapt the module placement accordingly. On the contrary, (Jiang et al., 2018) proposes a distributed orchestration architecture where each orchestrator controls a subset of resources and services. All these orchestrators are equally important (i.e., flat architecture) and coordinate with one other for the orchestration of the overall system.

With regard to Fog orchestration architectures, SDN plays a fundamental role (Baktir et al., 2017). Indeed, by separating the control plane (i.e., where the network control logic resides) from the data plane (i.e., the set of network devices forwarding packets), this technology enables a great network programmability and flexibility (Tomovic et al., 2017). More specifically, SDN controllers have a comprehensive and constantly updated view of the dynamically changing network and computing resources and expose a northbound programming interface to network

[9]This does not change the fact that the topological distance between one or more IoT devices and a FN is much shorter than the one between the same IoT devices and the Cloud.

[10]See `https://docs.docker.com/engine/swarm/`. Last accessed: 12 October 2019.

[11]See `https://openiotfog.org/en/`. Last accessed: 12 October 2019.

[12]See `http://docs.oasis-open.org/tosca/tosca-primer/v1.0/cnd01/tosca-primer-v1.0-cnd01.pdf`. Last accessed: 10 October 2019.

management applications in order to adaptively orchestrate resources, services, and network traffic according to QoS/QoE requirements and current system conditions (Baktir et al., 2017). The communication between the SDN controller and the data plane devices is commonly achieved through the OpenFlow[13] protocol. The authors in (Tomovic et al., 2017) propose a Fog-IoT architecture where SDN is exploited as an orchestration and network control facility. This architecture conceives a cooperation among different SDN controllers, and FNs expose Application Programming Interfaces (APIs) to allow the remote monitoring and management of their resources. To conclude, both (Truong et al., 2015) and (He et al., 2016) present SDN-based architectures for Fog orchestration in an IoV domain. In particular, (He et al., 2016) focuses on load balancing strategies.

Policies - In (Wen et al., 2017), the authors present the preliminary results of their genetic algorithm for Fog orchestration. Although their proposal features some scalability limitations, it originally characterises security risks as a cost to be minimised when performing orchestration. Instead, in (Taneja and Davy, 2017), the authors focus on the efficient utilisation of computing resources as the primary concern in the formulation of their solution of Fog orchestration. The authors in (Skarlat et al., 2017) model Fog orchestration as an optimisation problem and propose a genetic algorithm to solve it. This work envisions a distributed orchestration architecture where each orchestrator controls a subset of FNs (i.e., a Fog colony) and can be in turn controlled by another orchestrator that resides at a higher level in the hierarchy. The top-most orchestrator is in the Cloud. Both (Mahmud et al., 2018b) and (Mahmud et al., 2018a) particularly focus on QoS- and QoE-aware orchestration policies. More specifically, (Mahmud et al., 2018b) is based on Fuzzy logic, and (Mahmud et al., 2018a) also performs energy-aware Fog orchestration. Indeed, it proposes to re-locate application modules in order to optimise the number of active FNs and thus minimise energy consumption. It is worth noting that (Skarlat et al., 2017; Mahmud et al., 2018b,a) all simulate their solutions in iFogSim, which indeed is the most utilised tool to simulate resource management techniques in Fog-IoT environments (Gupta et al., 2017). As (Mahmud et al., 2018a), also (Nan et al., 2017) proposes an orchestration algorithm to find the optimal compromise between QoS and energy efficiency. Going into details, the authors propose to power FNs first via green energy (e.g., produced by sun or wind) and, when this is not available due to inappropriate weather conditions, via brown energy (e.g., produced through fossil fuels). The energy cost is represented only by brown energy consumption. To conclude, (Aazam et al., 2018) defines a scheme that incorporates service usage patterns and the history of service customers in order to dynamically estimate resources and adapt resource orchestration accordingly. The dynamic deployment of multi-component IoT appli-

[13] See `https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.3.1.pdf`. Last accessed: 12 October 2019.

cations in Fog infrastructures is also addressed in (Brogi and Forti, 2017a), where the authors propose a system model and present algorithms to determine eligible deployments.

Open issues - There exist several northbound interfaces in between SDN controllers, on the one hand, and network management applications, on the other hand. Therefore, the definition of a vendor-independent northbound interface as a result of a standardisation effort would be a significant contribution (Baktir et al., 2017). Similarly, the research on the communication among peer SDN controllers is still at its beginning and thus is worth of investigation (Baktir et al., 2017). Finally, another research direction is the application of predictive analytics for a dynamic and proactive orchestration.

Deployment Models & Revenue Scenarios

An important challenge that needs to be faced in order to get a wider adoption of Fog-based systems consists in understanding the potential revenue and incentive models that can be supported through different deployment scenarios. Such models are needed to better understand why: (i) infrastructure providers would offer their resources to act as FNs; (ii) users would want to make use of these FC resources. We can consider FC deployments to be somewhat similar to the deployment of other types of edge infrastructures that currently exist, such as Wi-Fi deployments within cities, which may be operated and managed by a variety of different organisations, ranging from universities, coffee chains, transport operators/city councils, and so on. It is useful to note that not all such infrastructure deployments require payment from the end user. Understanding potential incentive models that encourage restaurant and café owners to operate Wi-Fi access points can be useful to understand this next generation of services which are operated towards the network edge. However, this is still an open issue and is likely to grow as the FC infrastructure becomes more resilient and mature (Petri et al., 2017; Weinman, 2018).

Revenue models can be related to the characteristics identified in Table 2.5, where geographical distribution, node heterogeneity, and security requirements influence how FNs can generate a potential revenue stream for providers. More importantly, without an adequate number of FNs being available, sustaining a suitable infrastructure that provides suitable computing power and network performance will be unrealistic. Providing incentive models for provision and maintenance of FNs is essential. We consider the following four types of deployment models. The description below attempts to provide context for the deployment model based on the particular deployment approach being used:

- **Dynamic FN discovery supported revenue model**: this model involves dynamic discovery of a FN as a user moves from one location to another. The

user device attempts to discover a FN in its "vicinity" using the advertised profile of the node (which can include: availability statistics, security credentials, and types of available services). Using this approach, the user does not have any guarantee that a suitable FN will be discovered to sustain an application session, but a negotiation can take place if multiple FNs are found. A user device can also cache previously seen FNs. The incentive for the provider is to gain revenue from each user session that is sustained using that FN. A user can purchase a subscription with particular FN types a priori (i.e., before discovery). A user is charged based on connection time, size of data, or range of services utilised. The deployment model in this case is the incentive for FN operators/owners to make services discoverable by IoT devices (including those that are mobile). The revenue earned by undertaking this would be the basis for the deployment model. Conversely, users/ owners of IoT devices need to determine whether a discovered service is suitable for their needs (taking account of a subscription cost to use the service). Discovering suitable services is akin to finding a service description match within a registry.

- **Pre-agreed contracts with Fog providers**: this deployment model involves generating pre-agreed contracts with operators of specific FNs – negotiated at a set price. Hence, there would be a preferential selection of particular nodes by a user if multiple choices are found. This also reduces risks for users, as security credentials would be included in these pre-agreed contracts and could be configured (e.g., use of particular encryption keys) beforehand. These pre-agreed contracts would need to comply with service level objectives (e.g., an availability profile) that an operator needs to meet. It is therefore possible that a FN operator may outsource their task to a Cloud provider. The incentive for the provider is to increase the number of potential subscribers by developing pre-agreed contracts. Capacity planning associated with such FNs is therefore dependent on accurately predicting potential future demand. The deployment model in this case involves agreeing a cost for entering into a contract with a Fog provider. This contract also allows preferential access to FNs owned by the provider.

- **FNs federation**: this deployment model involves multiple FN operators collaborating to share workload. In order to sustain potential revenue, this would imply federation between FNs that exist within a particular geographical area. There would be a preferred cost for sharing workload with other providers, enabling revenue sharing between providers. To enable such an exchange to take place, it is necessary to identify how workload "units" can be characterised. This is equivalent to alliances set up between airline companies, for instance, where specialist capability (and capacity) available along a particular route can be shared across multiple operators. In the same way, if an operator de-

ploys specialist GPUs or video analytics capability within a FN at a particular location, other operators could also make use of this in a seamless way and similarly share other capabilities in other locations. This type of geographic-centric specialisation could enable localised investment within particular areas by operators.

- **Fog-Cloud exchange**: this deployment model involves a user device not being aware of the existence of any FN. Instead, the user device interacts with a Cloud operator who then attempts to find a FN in the vicinity of the user. Therefore, the Cloud operator needs to keep a track of the user location and discover suitable FN operators that could be used to support the session at a particular location. In this instance, the Cloud operator will always try to complete the user request first; however, if a QoS target is unlikely to be met due to latency constraints, it can outsource the user request to a regional FN. The incentive in this instance is to enable Fog-Cloud exchange contracts to be negotiated between providers (Eivy, 2017).

Open issues - Some of the above deployment and revenue generation scenarios are not unique to FC and closely relate to other similar efforts in service-oriented systems. We identify three open issues that could have an impact on realising some of these deployment models in practice:

- The recent emergence of regulations such as the GDPR, which is being introduced in Europe, could have a significant impact on these deployment models. GDPR necessitates all external service providers who hold data about users to seek consent from users and state: (i) which data they hold; (ii) how these data are being used by the provider. More significantly, the user has the ability to revoke access to their data at any time. With the use of FNs, user data may be fragmented across different providers, depending on the mobility pattern of the user. Understanding how a group of FNs, which may not be part of a federated infrastructure, may seek consent of users remains a challenge.

- Vendors who own and operate an infrastructure at the network edge (e.g., cellular base stations) could become potential Fog providers in the future, as they are likely to provide the FN that a user interacts with. Deployment models that require interaction between such network operators and Cloud providers remain unclear at present.

- There is also potential for auction models that could operate in a FC environment when multiple FNs are available for a user to choose from. Understanding the metrics (other than price) that influence such auctions remains a challenge. Additionally, such auctions should not cause detrimental overhead on the performance of the application that makes use of the FC infrastructure.

The definition of services that manage and operate such algorithms is also an open issue.

Security and privacy

As reported in Table 2.2, one of the main advantages of FC over other approaches to Cloud-IoT integration is represented by security and privacy enforcements, especially with regard to the protection of sensitive data. Nevertheless, this advantage comes at the cost of new security and privacy challenges that are raised by some of the intrinsic characteristics of the Fog (see Table 2.5). More specifically, distributed systems are in general more vulnerable to attacks than centralised ones. Moreover, with the purpose to provide a better QoS/QoE and enable the distinguishing advantages of FC, FNs are usually deployed in environments that are less protected than Cloud DCs (Chiang and Zhang, 2016). To conclude, both the heterogeneity among FNs and their limited computing capabilities, if compared to Cloud servers, further complicate the situation. The security and privacy challenges afflicting FC have been significantly drawing the attention of the research community. This is demonstrated by the great number of works that have been proposed to face such challenges and, as a consequence, by the considerable number of surveys focusing on this topic (Stojmenovic et al., 2015; Khan et al., 2017; Roman et al., 2016; Ni et al., 2018; Mukherjee et al., 2017; Shirazi et al., 2017). Among these surveys, (Ni et al., 2018) is the only one that specifically discusses these challenges within the IoT context. Given this abundance of survey papers and for space reasons, what follows is a high-level overview of the main security and privacy concerns in a Fog-IoT environment.

The OFC dedicated an appendix of its OFRA document (Consortium, 2017) to a detailed discussion about several security aspects in a FC environment. According to this appendix, security is the largest cross-cutting technical concern within critical IoT systems, which necessitate common baseline and inter-operable standards to address security challenges within both hardware and software. Particularly interesting is the analysis of the hardware/firmware precautions that the Consortium suggests in order to implement a full-stack secure Chain of Trust comprised of trusted components. Among such components, IoT devices represent the most vulnerable elements of the FC hierarchy. Securing this part of the infrastructure is a promising research direction that has been only preliminary explored up to now, mainly relying on remote attestation techniques (Brasser et al., 2016; Celesti et al., 2017).

With regard to the possible attacks against FNs, man-in-the-middle is one of the most important and urgently needs effective countermeasures. Being deployed in the field, FNs are vulnerable to this type of attack that consists of compromising a FN with malicious code (Wang et al., 2015b) or even in replacing it with a fake FN (Marìn-Tordera et al., 2017).

From the point of view of the end users, privacy is beyond any doubt one of the most prominent requirements. An interesting research challenge (strictly connected to that of mobility support) in this field is related to the design and implementation of techniques able to guarantee the privacy of location and mobility data. As FC enables end users to offload their tasks to the nearest FNs, their location and trajectory can be retrieved by an attacker (Mukherjee et al., 2017) (e.g., a malicious FNs administrator). This could even be the result of internal policies of Cloud/Fog providers that might act in an "honest-but-curious" way (Ni et al., 2018).

Finally, it is worth mentioning that security and privacy solutions in FC also have to take into consideration the complex combination of regional and governmental requirements that must be satisfied due to the widespread distribution of the nodes in a Fog hierarchy, as also explicitly stated in (Consortium, 2017). This, however, is out of the scope of the present work.

2.6 Fog Computing platforms for the Internet of Things

As a proof of the increasing maturity of the Fog paradigm, several software and hardware systems are already available for use. In this section, we give an overview of existing FC platforms for the IoT. To the best of our knowledge, we are the first to make this novel contribution, which we believe may draw the attention of engineers and developers. Specifically, we classify platforms into three categories, namely: (i) software platforms; (ii) development frameworks; and (iii) hardware platforms. This section concludes with a discussion of OFC efforts towards a standardisation process that involves both FC software and hardware platforms.

Software platforms

We define a FC software platform for the IoT as "*a software environment providing at least the basic functionalities and mechanisms that are necessary for the deployment and execution of IoT applications over a Fog infrastructure*". We first discuss software platforms started as industrial initiatives, and then focus on open-source systems. Table 2.6 summarises and compares these platforms on the basis of a set of features – we

[14]See `https://github.com/Azure/iot-edge`. Last accessed: 13 October 2019.

[15]See `https://github.com/smartfog/fogflow`. Last accessed: 13 October 2019.

[16]See `https://github.com/ParadropLabs/Paradrop`. Last accessed: 13 October 2019.

[17]See `https://github.com/OpenEdgeComputing/elijah-openstack`. Last accessed: 13 October 2019.

[18]See `https://github.com/MDSLab/stack4things`. Last accessed: 13 October 2019.

[19]See `http://openvolcano.org/dokuwiki/doku.php?id=ov:download`. Last accessed: 13 October 2019.

do not include features such as orchestration, as these are common across all platforms. Furthermore, we only discuss those platforms that are already available for use, namely those whose maturity level is either *Pre-product* or *Product* (see Section 2.4). Nonetheless, there exist ongoing research activities likely to produce platforms in the near future (Lebre et al., 2017; OpenStack, 2018a; Montero et al., 2017).

Table 2.6: Comparison among the FC software platforms for the IoT.

Platform	**Open-source**	**Extension of a Cloud platform**	**Only runs on specific hardware**	**Maturity**
Nebbiolo			✓	Product
FogHorn Lightning				Product
Cisco IOx			✓	Product
Dell Edge Device Manager			✓	Product
IBM Watson IoT		✓		Product
AWS Greengrass		✓		Product
Microsoft Azure IoT Edge	✓[14]	✓		Pre-product
FogFlow	✓[15]			Pre-product
ParaDrop	✓[16]			Pre-product
OpenStack++	✓[17]	✓		Pre-product
Stack4Things	✓[18]	✓		Pre-product
OpenVolcano	✓[19]	✓		Pre-product

Commercial platforms - Nebbiolo Technologies was founded by Flavio Bonomi, who first advanced the concept of FC in 2012 (when he was with Cisco). The Nebbiolo Technologies FC platform (Nebbiolo, 2018a) is a commercial platform consisting of a closed-source software stack that runs on a proprietary hardware solution and particularly tailored to the industrial automation sector. The platform allows a Cloud-like centralised management of distributed mini DCs deployed at the network edge. Such mini DCs comprise computing, networking, and storage resources in the form of purpose-built hardware nodes called fogNodes. This software platform includes the fogOS software stack, a custom operating system providing virtualisation, SDN, data analytics, and security features. Moreover, the fogSM is a system manager, deployed in the Cloud or on-premises, that allows remote management of the fogNodes and assisted deployment of IoT applications.

FogHorn Lightning by FogHorn Systems (FogHorn, 2018) includes the FogHorn Manager that allows remote management, monitoring, and configuration of edge nodes, and deployment of IoT applications. Moreover, as described later, the company provides a powerful analytics framework enabling real-time and on-site stream processing of data coming from IoT devices. FogHorn Lightning does not exclusively run on a specific hardware.

As the biggest network appliance manufacturer, Cisco proposes a wide range of both FC software and hardware products for the IoT. The Cisco IOx (Cisco, 2018b) ecosystem provides uniform and consistent hosting capabilities for Fog applications across Cisco network infrastructure products. In particular, the Cisco IOx Fog Director provides users with the possibility to deploy, run, and monitor applications across the Fog infrastructure, while the Cisco IOx Client is a command-line utility for developers to control application life-cycle tasks within typical developer systems.

Similarly, Dell Technologies entered the market by proposing Dell Edge Device Manager (Technologies, 2018a), which enables secure registration of Dell hardware products and their remote management with automation of upgrades, task scheduling, real-time monitoring, and configuration.

Two other platforms are: (i) IBM Watson IoT (IBM, 2018), which extends IBM Cloud; and (ii) AWS Greengrass (Amazon, 2018), which extends AWS Cloud. Both extend pre-existing proprietary Cloud platforms towards the network edge and provide support for deploying and running application components on IoT devices, edge nodes, and the Cloud.

Open-source platforms - Similarly to IBM and Amazon, Microsoft has recently released its FC software platform for the IoT, namely Microsoft Azure IoT Edge (Microsoft, 2018), which extends Microsoft Azure Cloud. This platform is open-source but is still in a preview phase.

FogFlow (Cheng et al., 2018) is a FC software platform that is able to automatically and dynamically compose multiple tasks into high-level IoT services. Each task

is represented by a Docker container hosting the data processing logic and needs to be described by the software developer through NGSI, the standard exploited within the FIWARE European project[20] for context information management. Based on such a description, FogFlow performs the orchestration in an optimised way, deploying tasks anywhere along the continuum from Cloud to Things, only when actually required, and based on the locality of data producers and consumers. Availability and mobility criteria are also taken into consideration by the system for task deployment.

Another FC software platform, which specifically targets nodes at the extreme edge of the wireless networks (i.e., home Wi-Fi routers and wireless gateways), is ParaDrop (Liu et al., 2016a). The attention to this specific kind of nodes is mainly motivated by their peculiar contextual knowledge about end user devices that are directly attached to them (e.g., proximity, characteristics of the channel). This knowledge is useful for making decisions about application placement and orchestration. Specifically, this platform can "paradrop" services from the Cloud to the network edge in the form of self-contained units, called "chutes", that are deployed as near as possible to the IoT devices requiring them (e.g., sensors, actuators, end user mobile devices). As common in many of these kinds of platforms, chutes are implemented as Docker containers. Due to such specific design choices, ParaDrop is particularly suitable for Smart Home and Smart Building scenarios. Although the range of currently supported nodes is still limited to a custom Wi-Fi access point based on the PC Engines APU2 single-board computer and few more nodes belonging to the Intel NUC family, there is the possibility to deploy a "ParaDrop router" as a QEMU/KVM Virtual Machine (VM).

To conclude, three open-source platforms integrate the Fog in OpenStack[21], which is the most prominent open-source Cloud platform. The OpenStack project initiated in 2010 as a joint initiative of Rackspace Hosting and the NASA and is currently managed by the OpenStack Foundation, a non-profit corporate entity established in September 2012. More than 500 companies have joined the project since then, and the OpenStack development community currently counts more than 82,000 members from 187 countries around the world (OpenStack, 2018b).

The first FC software platform extending OpenStack with Cloudlets support is OpenStack++ (Ha and Satyanarayanan, 2015). It is the output of the Open Edge Computing (OpenEdgeComputing, 2018) initiative, launched in June 2015 by Vodafone, Intel, and Huawei in partnership with the Carnegie Mellon University. The second platform extending OpenStack with FC capabilities is Stack4Things (Longo et al., 2017), which was initially developed by the University of Messina and is now commercialised by SmartME.io Srl. It provides functionalities for the remote management of IoT device fleets irrespective of their physical location, their network

[20] See `https://www.fiware.org/`. Last accessed: 13 October 2019.
[21] See `https://www.openstack.org/`. Last accessed: 13 October 2019.

configuration, and their underlying technology. It is a Cloud-oriented horizontal solution providing IoT objects virtualisation, customisation, and orchestration. Last but not least, OpenVolcano (Bruschi et al., 2016; OpenVolcano, 2018) is an open-source platform, conceived in the context of the Horizon 2020 INPUT project[22], that specifically aims at supporting FC services in 5G-ready infrastructures. Besides extending OpenStack, it applies Network Functions Virtualization (NFV) and SDN through the OpenFlow protocol, thus enabling great network programmability and flexibility.

Development frameworks

We define a FC development framework for the IoT as *"a set of tools* (*e.g., libraries, microservices, abstraction layers*) *easing the development of Fog applications for the IoT and assisting the developer in focusing on the application logic rather than on the distributed nature of the Fog infrastructure on top of which the application will be deployed"*. Table 2.7 reports a comparison among the FC development frameworks for the IoT. Specifically, we report the information that we believe is more interesting from the point of view of the application developer, namely if the framework is released under an open-source license, the supported programming languages, and the deployment model. Prior to starting, we point out that most of the development frameworks are tightly coupled with a FC software platform discussed previously. To the best of our knowledge, only two frameworks are completely independent of the under-

Table 2.7: Comparison among the FC development frameworks for the IoT.

Framework	Open-source	Coupled with a software platform	Supported languages	Deployment model
EdgeX Foundry	✓[23]		Java (officially supported) + others (from the community)	Docker containers
macchina.io	✓[24]		C++	Custom C-based runtime environment
Nebbiolo SDK		✓	Python	Docker containers
FogHorn Lightning SDK		✓	C++ (micro edition) + other not specified languages (standard edition)	n/a
Cisco IOx SDK		✓	C/C++, Python, Ruby, Nodejs	Custom containers, Docker containers, KVM/QEMU VMs

[22]See `https://www.input-project.eu/`. Last accessed: 13 October 2019.
[23]See `https://github.com/edgexfoundry`. Last accessed: 13 October 2019.
[24]See `https://github.com/macchina-io/macchina.io`. Last accessed: 13 October 2019.

lying FC software platform, thus totally decoupling application development from FN management and service deployment.

The most important initiative within this second category of development frameworks is the EdgeX Foundry project (Foundation, 2018), which is hosted by the Linux Foundation. In April 2017, Dell Technologies, in conjunction with several partners and customers, launched the EdgeX Foundry project with the donation of about over 125,000 lines of code. The project is currently being actively developed by tens of companies including Samsung, Analog Devices, Toshiba, and others. EdgeX Foundry is a vendor-neutral open-source interoperability framework that allows developers to implement IoT applications in a hardware, Operating System (OS), and programming language agnostic way. It is composed of an ecosystem of microservices that can be combined and plugged together according to the application logic and/or easily replaced with open-source or proprietary solutions. The reference language is Java and at the core of the architecture lies a MongoDB database, which is used as a persistence mechanism for both the data collected by sensors and the metadata about the connected devices. A key aspect of the project is the certification program that aims at guaranteeing an overall ecosystem compatibility. Indeed, in order to be authorised to use the EdgeX trademark, vendors need the Project board to certify any commercial value-add that they build within the core framework, so that the core APIs are always supported (Strategy, 2017).

macchina.io (GmbH, 2018) is a toolkit that allows IoT developers to easily implement embedded applications on top of the most commonly used Linux-based single-board computers such as Raspberry Pi. It is based on a JavaScript and C++ runtime environment and provides several bundles implementing interfaces to devices and sensors, network protocols such as MQTT or COAP, interfaces to Cloud services (e.g., for sending SMS or Twitter messages), and a Web-based user interface. The core of the platform is represented by the POCO C++ libraries that implement essential features, e.g., platform abstraction, multithreading, stream, datagram and multicast sockets, HTTP server and client, SSL/TLS. macchina.io is released under the Apache 2.0 License.

What follows is the set of FC development frameworks for the IoT that are part of a software platform discussed previously. Within its proprietary ecosystem, Nebbiolo Technologies provides an SDK for the development of native applications on top of the fogOS software stack (Nebbiolo, 2018c). The reference language is Python, and the developers are provided with a set of tools that allow an application to be packaged within a Docker container and deployed onto the system in the form of a fogLet. A set of libraries are available to interact with the fogOS Pub/Sub Databus for data, events, and alarms propagation.

Similarly, the FogHorn Lightning platform provides developers with specific SDKs. This development framework is available in two editions, namely standard and micro, which primarily differ from one another for their footprint. In the micro

edition, a C++ SDK allows custom applications to implement data preprocessing, data visualisation, and machine learning features at the edge. In the standard edition, a polyglot SDK further provides support for multiple industrial protocols (e.g., MQTT, Modbus). No open documentation is available on the system architecture; therefore, no details about the supported deployment methods can be provided.

Within the IOx (Cisco, 2018b) ecosystem, Cisco provides the Cisco IOx SDK and other development tools, which help developers to correctly package their applications for execution on Cisco IOx. The SDK allows developers to use several high-level languages, e.g., C/C++, Python, Ruby, Node.js and supports different categories of applications. Specifically, both containerised applications and VM-packaged applications are supported. The developer can use either an ad-hoc LXC-compliant format or the Docker tooling to containerise applications. A KVM/QEMU hypervisor infrastructure is available for VM-packaged applications. Finally, the "IOx middleware services" provide high-level abstractions and APIs to facilitate the development of IOx applications.

Hardware platforms

In this section, we report the hardware solutions that are provided by the most prominent hardware manufacturers on the market and that can play the role of FNs. Table 2.8 reports a comparison among such FC hardware platforms on the basis of those features that we believe are of particular interest in an IoT context. Specifically, besides some information about the hardware resources and the approximate price, we include details on: (i) the network connectivity; (ii) the additional interfaces that can be used to connect with external sensors and actuators (which represents the main difference between this kind of hardware products and the standard Cloud DC solutions); and (iii) the presence of hardware-based security solutions, such as Trusted Platform Module (TPM). By looking at Table 2.8, it is evident that FNs are very heterogeneous, especially in terms of hardware capabilities and therefore price.

Nebbiolo Technologies offers a series of modular hardware solutions, fully compliant with their FC software platform, called fogNodes (Nebbiolo, 2018b). fogNodes exist with a wide range of form factors and different computing capabilities, including standard x86 CPUs, FPGAs, and GPUs. Ethernet connectivity is available by default, while Wi-Fi and LTE interfaces come with optional modules. Particularly interesting is the presence of a TPM device onboard to provide hardware security capabilities. TTTech produces the MFN 100 (TTTech, 2018), a device that can be employed as FN in industrial environments within the Nerve platform, which integrates the fogOS and the fogSM from Nebbiolo Technologies.

Cisco provides a series of network infrastructure products supporting the IOx ecosystem and thus allowing seamless deployment and execution of Fog applications. Specifically, the Cisco 800 Series Industrial Integrated Services Routers (Cisco,

Table 2.8: Comparison among the FC hardware platforms for the IoT.

Manufacturer	Model	Hardware resources	Network connectivity	Interfaces for external sensors and actuators	Hardware-based security	Price
Nebbiolo Technologies	fogNode	4-8 cores x86 i5/i7 CPUs, 8-16 GB RAM	Ethernet (Wi-Fi and LTE are optional)	No	✓	n/a
TTTech	MFN 100	Intel Atom 4 cores 1.8 GHz CPUs, 4-8 GB RAM	Ethernet	2 USB ports		n/a
Cisco	800 Series Industrial Integrated Services Routers	Intel Atom 2 cores 1250 MHz CPU, 2 GB RAM	Ethernet, LTE (Wi-Fi is optional)	2 asynchronous serial interfaces	✓	2000$
Cisco	Compute Modules for the 1000 Series	AMD GX-410VC 4 cores 800 MHz CPU, 4 GB RAM	Ethernet	1 USB port		2000$
Dell	Edge Gateway 5000	Intel Atom E3825 CPU, 2 GB RAM	Ethernet, Wi-Fi, BLE, LTE	6 different serial interfaces	✓	1000$
Dell	Embedded Box PCs	4 cores x86 i5/i7 CPU, 4-32 GB RAM	Ethernet, Wi-Fi, BLE, LTE	5 USB ports, 3 different serial interfaces, GPIO	✓	1000$
HPE	GL20 IoT Gateway	Intel 4300U 2 cores i5 CPU, 8 GB RAM	Ethernet, Wi-Fi (LTE is optional)	5 USB ports, 2 different serial interfaces		2000$
HPE	Edgeline EL1000-/4000	1-4 Intel Xeon D 8-16 cores each, up to 128 GB RAM	Ethernet	via PCIe expansion slots		3800$
Raspberry Pi Foundation	Raspberry Pi 3 Model B+	1.4GHz 4 cores ARM Cortex-A53 CPU, 1GB RAM	Ethernet, Wi-Fi, BLE	4 USB ports, 40 GPIO pins, Camera Serial Interface		35$
Qualcomm	DragonBoard 820c	2.35GHz 4 cores CPU, Adreno 530 GPU, 3GB RAM	Wi-Fi, Bluetooth	3 USB ports, pins, Camera Serial Interface		200$
Qualcomm	DragonBoard 410c	1.2GHz 4 cores ARM Cortex-A53 CPU, Adreno 306 GPU, 1GB RAM	Wi-Fi, Bluetooth	3 USB ports, pins, Camera Serial Interface		75$
Intel	Edison	500MHz 2 cores CPU, 100MHz MCU, 1GB RAM	Wi-Fi, Bluetooth	Total of 40 GPIO pins		50$

2018a) are compact routers providing IoT gateway functionalities. They offer integrated 4G LTE connectivity, Ethernet ports, and a couple of asynchronous serial interfaces for sensors/actuators. The Cisco Compute Modules for the Cisco 1000 Series Connected Grid Routers (Cisco, 2018c) are field-replaceable modules that bring FC capabilities to already operational networks. They are specifically tailored to industrial IoT markets such as utilities, manufacturing, and Smart Cities.

Being primarily a hardware manufacturer, Dell Technologies provides enterprises with a portfolio of IoT-focused infrastructure products that allow them to build and deploy complete, secure, and scalable solutions from end IoT devices, to the network edge, and up to the Cloud (Dell Technologies, 2018). In this regards, Dell Technologies portfolio includes the following products. On the one hand, the Dell Edge Gateway 5000 (Technologies, 2018b) is the flagship product of a family of IoT gateways that is equipped with a wide range of I/O connectors to bridge both legacy systems and modern sensors to the Internet but also provides enough computing/storage power to aggregate data and perform local analytics. On the other side, the Dell Embedded Box PCs (Technologies, 2018c) seem to prioritise performance and adaptability to different use cases, rather than I/O connectivity. They are highly reliable devices for a variety of use cases, including process and discrete manufacturing, fleet management, kiosks, digital signage, surveillance, and automated retail solutions. Dell also provides Cloud DC solutions for advanced analytics, data management, storage, and computation, but these are out of the scope of this survey.

Among the main hardware manufacturers, also HPE provides customers with a set of products that are specifically designed with the FC use case in mind. The HPE GL20 IoT Gateway (LP, 2018b) is a compact solution targeting verticals such as manufacturing, Smart Cities, oil and gas. Similarly to other manufacturers' IoT gateways, it comes with a set of I/O interfaces for connecting to IoT devices and with enough power to quickly elaborate data and react to critical situations. Products belonging to the HPE Edgeline family (LP, 2018a), such as the EL1000 and EL4000, instead, feature a reduced set of I/O interfaces but possess an expansible amount of hardware resources, which makes them similar to standard DC solutions.

There exists also a considerable number of less powerful FNs, which can be therefore employed in a more limited range of scenarios but are much cheaper than the previously discussed solutions. For instance, in Table 2.8, we report information on the Raspberry Pi 3 Model B+ single-board computer (Pi, 2018), which is powerful enough to behave as a FN. Indeed, the authors in (Bellavista and Zanni, 2017) demonstrate the feasibility of deploying Fog-IoT services as Docker containers on a Raspberry Pi. Other single-board computers that are worth mentioning as potential FNs are the Qualcomm DragonBoard 820c (Qualcomm, 2018b), the Qualcomm DragonBoard 410c (Qualcomm, 2018a), and the Intel Edison board (Intel, 2018).

Towards a standardisation

The proliferation of proprietary solutions in ICT inevitably leads to delays in innovation and development and to strong limitations to the potential economic impact that ICT might have. Looking at the IoT, the McKinsey Global Institute states that interoperability is required on average for 40% of the total potential economic value that the IoT enables (Manyika et al., 2015). Therefore, it is time for technology suppliers to give birth to interoperable ecosystems by cooperating on the definition of standard technologies, protocols, and architectures.

Following this direction within the FC field, the OFC was founded in 2015 by ARM, Cisco, Dell, Intel, Microsoft, and the Princeton University and currently has 62 members throughout the world (Consortium, 2018a). The stated objectives of the Consortium are: (i) to create an open, comprehensive reference architecture for the Fog; (ii) to promote the adoption of the Fog in the several application domains that may benefit from it; and (iii) to influence Fog standards development through liaisons with standardisation bodies. In February 2017, the Consortium released the OFRA, thus paving the way to a multi-vendor interoperable FC ecosystem. More recently, in June 2018, the IEEE Standards Association (IEEE-SA) adopted the OFRA as an official standard (Association, 2018), namely the IEEE 1934^{TM}.

We now provide a high-level overview of the salient characteristics of the abovementioned architecture; further details may be found in (Consortium, 2017). Eight core principles, known as pillars, guided the definition of the entire OFRA; they are: (i) security; (ii) scalability; (iii) openness; (iv) autonomy; (v) reliability, availability, and serviceability (RAS); (vi) agility; (vii) hierarchy; and (viii) programmability. Basically, the OFRA consists of five vertical perspectives and three horizontal views. Each perspective represents a cross-cutting concern that involves all the layers of the architecture. In other words, perspectives are the OpenFog pillars made integral part of the architecture itself. On the other hand, each of the three views is a set of layers that represents one or more specific aspects of the architecture. To be more precise, the Node View includes all the aspects of interest to the chip designers and the silicon manufacturers, as it clarifies the generic characteristics (e.g., computation, storage, networking) that a chip in a FN should possess. The actual FN is a composition of one or more chips (i.e., Node Views) with some additional elements. The higher the number of chips in a FN, the higher its expected positioning within the Fog hierarchy due to its greater capabilities. The OpenFog view that represents a FN is called System View, and typically the stakeholders interested in it are the system architects and the hardware Original Equipment Manufacturers (OEM). To conclude, the Software View characterises the software running on a FN. It includes the software for the management of the node and its communications, the application services, and the software required to support them (e.g., VMs and containers, software libraries, databases, message brokers). As such, this view is of interest to the software architects and the application developers.

Chapter 3

Mobility support in Fog Computing

This chapter provides the background on mobility support and service migration in FC, which represent the focus of this thesis. Section 3.1 describes the research problem and introduces the concept of Companion Fog Computing. Section 3.2 reports five FC use cases where mobility support proves necessary. Then, in Sections 3.3, 3.4, 3.5, and 3.6, we discuss the different aspects that characterise the issue of mobility support, provide an overview of the state of the art for each of them, and point out the open issues and future research directions.

3.1 Companion Fog Computing

As outlined in Section 2.5, extending the Cloud paradigm towards the network edge introduces novel aspects that may either lead to new research challenges or change the way existing challenges are to be dealt with. A new research problem that arises with the introduction of FC is that of mobility support. As already discussed, Cloud services are in general distant from the served users, irrespective of the position of the latter. Hence, end (user) mobility is not an issue in Cloud-only environments. On the contrary, the distinguishing feature of FC, which brings all the benefits described in Table 2.2, is its proximity to the end devices. Therefore, device mobility plays a fundamental role in the Fog because it may compromise the above benefits. When a device moves, indeed, the topological distance to the serving FN may increase, possibly impairing the distinctive benefits of FC and thus the QoS provided to the user.

The objective is to support device mobility in a Fog environment or, in other words, to guarantee the FC benefits also when devices move from one place to another. The most investigated approach in literature to achieve this purpose is by migrating the Fog service across the FC infrastructure, thus to let it follow the application component running on the mobile device (Rejiba et al., 2019; Wang et al., 2018). As a result, the Fog service would be a "companion" of the mobile application; we refer to this as Companion Fog Computing (CFC).

The problem of mobility support and service migration in Fog environments has attracted great attention in recent years. This is proved by the significant number of published works in the field. In the next sections, we discuss these works, organising them in the following four categories/aspects that characterise research on mobility support and service migration in the Fog:

- **hosting and migration techniques -** this research direction consists in investigating which is the most appropriate way (considering the Fog, with its characteristics, as the underlying system) to implement a Fog service as well as which are the best techniques to migrate services across FNs. This category is further discussed in Section 3.3;
- **migration policies -** this group of works focuses on the definition of policies that, considering information on the infrastructure and on mobile devices, decide when and where to migrate Fog services. Section 3.4 focuses on this aspect;
- **migration platforms -** this category gathers all those aspects related to the design and implementation of software platforms that provide mechanisms to support mobility through service migration. Further details can be found in Section 3.5;
- **simulation of mobility and migration -** Section 3.6 reviews the state-of-the-art simulators for FC environments, with a focus on the modelling of device mobility and service migration.

For each of the above categories, we overview the state of the art and outline the open issues and future research directions. This helps us to provide a clear picture of the research gaps in the field, some of which we try to fill through the works that are discussed in the following chapters.

3.2 Use cases

Mobility support is essential to enable several Fog-based IoT applications where devices are mobile. In what follows, we describe five example use cases with the objective to show the importance of this research topic in real-life scenarios. The first use case that we report is the more detailed, since it is considered as the reference use case in Chapter 5.

Smart assistant

This use case is adapted from (Huawei, 2017) and depicted in Figure 3.1. A visually impaired person wears a smart helmet embedding a smart camera. Due to

its resource limitations, the camera continuously streams the captured video to a resource-richer node. Here, Artificial Intelligence (AI) techniques are applied to the video in order to characterise the current context of the user (e.g., which objects or events surround him/her). Based on the obtained insights, the remote service returns voice guidance feedback to the visually impaired person. As reported in (Huawei, 2017), this application requires the RTT between the smart helmet and the serving node to be lower than 20 ms. This cannot be achieved by relying on the distant Cloud, and FC may be a solution. At the beginning of the considered scenario, the user is in the hospital for a check-up. The hospital network, to which the smart helmet connects through Wi-Fi, hosts two FNs. The service that executes the AI logic runs on *Fog node 0*. Once the medical check-up is over, the user takes a bus to go home. The bus network comprises only one FN (i.e., *Fog node 2*), which also behaves as Wi-Fi access point and connects to the Internet though 4G/LTE. The handover from the hospital network to the bus network causes a considerable increase in the topological distance to Fog node 0 and hence makes Fog service migration to Fog node 2 necessary. Similarly, when the user reaches home, another migration is needed. Therefore, at the end of the reported scenario, the Fog service runs on *Fog node 4*, which is one of the two FNs present in the residential network of the user.

Smart tourism

Augmented Reality (AR) and Virtual Reality (VR) services that perform video analytics in the Fog need to follow mobile users (ETSI, 2016). This is necessary mainly

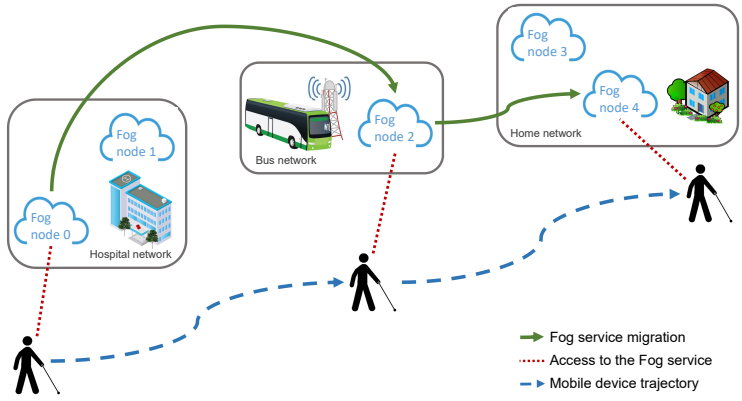


Figure 3.1: Mobility support in the smart assistant use case.

to guarantee low latencies and avoid saturating the backbone bandwidth with the video streams from mobile devices. An example of AR use case requiring mobility support is that of a tourist in a Smart City who points at streets and buildings with his/her smartphone or smart glasses and receives useful information in real-time about the points of interest in the view of the camera (e.g., historical background of monuments, shop special sales).

Smart healthcare

Smart healthcare is another application domain where mobility support is needed. In the near future, biomedical parameters (e.g., blood pressure, blood sugar level, body temperature, stress) will be measured by means of medical-grade wearables directly connected to the mobile network infrastructure (Ericsson, 2017), where time-critical analysis can be carried out for real-time actuation. Guaranteeing low and predictable latencies as the person moves is of paramount importance for his/her safety.

Smart factory

In an industrial environment, portable AR devices (e.g., smart glasses, tablets) could be used by operators to obtain real-time information on aspects such as: products, production procedures, instructions. Real-time AR systems exploit FC services to analyse and augment images with bounded low latency and without the need to offload them to the Cloud (Fernández-Caramés et al., 2018). In this case, for instance, migration could be exploited to support AR functionalities even when mobility is involved. Let us consider an operator moving through the production plant for inspection, equipped with smart glasses that show status information on the various equipment. In this case, migration of the AR service is mandatory to ensure that images are always collected and analysed in due time.

Smart vehicles

In the vehicular domain, applications implementing real-time situation awareness or cooperative functionalities will require FC and storage capabilities to implement time-critical and data-intensive tasks (Zhu et al., 2018). For instance, let us consider an autonomous driving vehicle and a supporting service that collects context data from the environment (e.g., from roadside sensors) and other vehicles to notify possible hazardous situations by applying mining techniques on the collected data. In this case, service migration is of paramount importance to keep the communication latency between the vehicle and the supporting service as low as possible and allow the vehicle to react promptly.

3.3 Hosting and migration techniques

As discussed in Section 2.5, new system, network, and environmental characteristics must be examined when extending the Cloud towards the network edge. Among the other aspects, these characteristics influence service hosting and migration in the following ways:

- differently from Cloud DCs, Fog environments are characterised by a high **heterogeneity** of nodes in terms of hardware capabilities, architectures, and operating systems. Hence, there is the need for a hosting technology for Fog services that is generic and lightweight enough to run on as many different types of FNs as possible;

- FNs are interconnected through a **WAN** and therefore experience higher latencies and lower throughputs than those present within a Cloud DC. Based on this, it is beneficial during migration to transmit the lowest possible amount of data;

- in the Cloud, the **total migration time** (i.e., the overall time required to complete the migration) is only a secondary consideration, whereas downtime (i.e., the time interval within migration during which the service is not up and running) is the primary concern. In the Fog, instead, limiting the total migration time may be of paramount importance, as there are situations in which protracted total migration times may lead to overall degraded performances (Ha et al., 2017). For instance, if migration took too long, another migration would be required immediately afterwards because, by that time, the mobile user has moved away;

- most Fog services, especially those deployed and running at the network edge, typically perform transient data analysis and time-critical control (see Table 2.3). Thus, they are not supposed to write to any **persistent memory** (e.g., the disk), unlike Cloud services (Cisco, 2015). Requests demanding semi-permanent or permanent storage are usually directed to a Cloud-hosted service (Sarkar et al., 2018), instead. As a result, what typically happens in Fog environments is that only the runtime (i.e., volatile) state is migrated and used at destination, together with a base service image representing the default disk state, to restore the service.

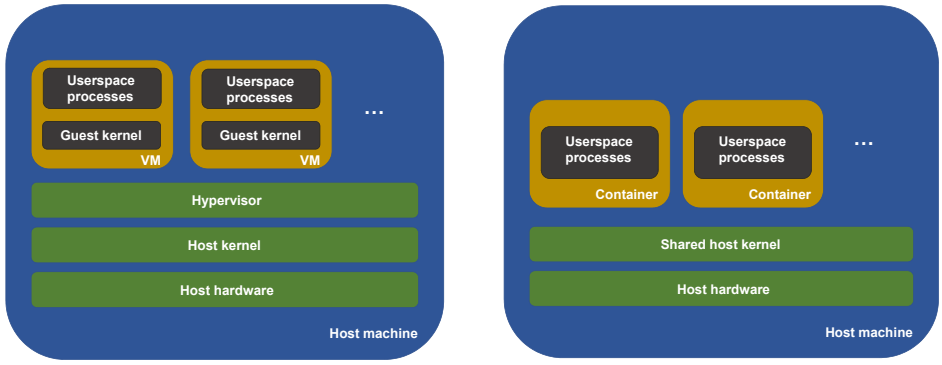
The above points are such that a hosting or migration technique that is well established in the Cloud might not be equally suitable for FC contexts. Hence, there is the need to identify hosting and migration techniques that are appropriate for the Fog. Then, such techniques may be taken into consideration by migration policies, implemented within migration platforms, and modelled in FC simulators.

One of the most popular hosting techniques in Cloud DCs is hardware-level virtualisation (also known as platform virtualisation – see Figure 3.2(**a**)), which allows the creation of **Virtual Machines** (VMs) (Habib, 2008). A VM acts like a real computer able to run an OS. More in general, software, when executed on VMs, does not have any access to (or visibility into) the underlying hardware resources of the host machine (Asvija et al., 2019). In platform virtualisation, *guest* is another name for VM, whereas the software in charge of VM instantiation on the host is called *hypervisor*, or *Virtual Machine Monitor* (VMM) (Desai et al., 2013).

OS-level virtualisation (Soltesz et al., 2007), better known as containerisation (see Figure 3.2(**b**)), is an approach enabled by a set of OS features where the kernel itself allows the coexistence of multiple and isolated instances of user-space environments, leaving the hardware abstraction layer as well as the enforcement for process sandboxing to the shared kernel co-hosting them. Virtualisation instances of this kind, the so-called **containers**, still look like real computers from the point of view of programs running in them, as the latter can see select resources (file systems, networks, hardware devices and capabilities), i.e., those exposed by an ordinary OS. However, software running inside a container can only see a subset of the resources made available by the kernel and specifically those assigned to the container.

The main difference between platform virtualisation and containerisation thus lies in the level at which abstraction and resource partitioning occur: at the kernel level for containers, at the hardware level for VMs. In other words, each VM has its own kernel, while containers all share the host kernel. This aspect leads to the following considerations. With their characteristics, VMs provide a better multi-tenant isolation than containers. Besides, VMs ensure better software compatibility than containers, meaning that, e.g., the same Ubuntu VM can run on both a Linux or a Windows host machine (Ha et al., 2017). Instead, Ubuntu containers can exclusively run on Linux machines, as they need to access the host kernel. Yet, containers are more lightweight and in general perform better (e.g., faster boot time, better execution speed, less data to store and transmit) (Karim et al., 2016; Ramalho and Neto, 2016). In light of this, even though the choice between VMs and containers may depend on the actual circumstances and needs, containers are in general preferred as hosting technology for FC (Morabito et al., 2018; Tang et al., 2018; Wang et al., 2018; Morabito, 2017; Bellavista and Zanni, 2017; Ismail et al., 2015). With their lightweight nature, indeed, containers better suit the Fog, where computing resources of nodes and network performance are in general lower than in the Cloud.

Another hosting technique that is worthy of notice is that of **mobile agents**. A mobile agent is a program able to autonomously migrate from a machine to another machine. Migration can be based either on a predetermined path or on one that agents themselves dynamically decide based on collected information. For mobile agent migration, most systems allow to capture the state of an agent only in the form of application-level data. As such, this mechanism does not allow the destination



(**a**) (**b**)

Figure 3.2: Platform virtualisation (a) vs. containerisation (b).

node to restore the state of threads as it was before migration (i.e., *weak mobility*). On the contrary, migration of VMs and containers transfers the execution state at a fine granularity, namely at instruction-level, and therefore allows to restore the thread context at the exact point before migration (i.e., *strong mobility*) (Poggi and Tomaiuolo, 2011). This is the main disadvantage of migrating mobile agents. Another issue with mobile agents is that their employment in FC and MEC environments is still in its infancy, with no available framework at the moment of writing (Wang et al., 2018). On the other hand, live migration of VMs is a well-established technology. Besides, container migration, although at a preliminary stage, is moving its first steps through the usage of *Checkpoint/Restore In Userspace* (CRIU) (see Chapter 4 for more details). Nonetheless, mobile agents present important advantages, being very lightweight and rapid to boot/run as well as able to migrate across the most diverse platforms and architectures (Wang et al., 2018). This latter characteristic is particularly interesting in FC environments, which are highly heterogeneous.

For the sake of clarity, Table 3.1 summarises the main advantages and disadvantages of working with the above mentioned hosting techniques. By looking at this table, containerisation may represent the hosting technology that suits the requirements in most of the FC scenarios. For this reason, in the next chapters, we focus on container migration.

Open issues - Being platform virtualisation the oldest among the hosting techniques, VM migration is the most consolidated. On the other hand, migrations of containers and mobile agents are currently more preliminary. An open issue is the implementation of a framework for agent technology in FC, which would ease the development of mobile agent-based IoT applications for FC scenarios. With respect to container migration, few works in literature provide knowledge in the field (Tang

Table 3.1: Advantages and disadvantages of the hosting techniques.

Hosting technique	Advantages	Disadvantages
VMs	High multi-tenant isolation and software compatibility; strong mobility	High quantity of data to store and transfer; slow at booting and executing
Containers	Low quantity of data to store and transfer; fast at booting and executing; strong mobility	Low software compatibility
Mobile agents	Low quantity of data to store and transfer; fast at booting and executing; high software compatibility	Weak mobility; no framework available

et al., 2018; Ma et al., 2018; Nadgowda et al., 2017; Govindaraj and Artemenko, 2018). Moreover, none of them evaluates and compares the existing container migration techniques, i.e., cold, pre-copy, post-copy, and hybrid migrations (see Chapter 4), not in the Cloud, nor in the Fog. To the best of our knowledge, there is only one work in literature that makes such a contribution, focusing on FC environments (Kakakhel et al., 2018). Nonetheless, it only assesses a subset (i.e., cold and pre-copy migration) of the available techniques and concludes that, being at a first stage, container migration results in unexpected errors. Therefore, in Chapter 4, we propose a work to fill this gap in literature. Finally, another research direction may be the definition of novel migration techniques or enhancements to the existing ones to better suit the distinguishing characteristics of FC environments.

3.4 Migration policies

An essential element within any mobility support solution is the execution of a migration policy. Migration policies decide: (i) when is the moment to migrate the Fog service (i.e., **When-to-Migrate** decision); (ii) to which FN the Fog service should be migrated (i.e., **Where-to-Migrate** decision). Such decisions can be made based on aspects such as: (i) users' mobility; (ii) requirements of the applications; (iii) status of computing and network resources; (iv) operational costs; (v) energy consumption considerations; and (vi) security and privacy concerns. When making migration decisions, a trade-off between the potential benefits of migrating the service, on the one hand, and the potential costs determined by such a migration, on the other hand, must be always made. For example, migration might reduce the RTT towards the service, but, at the same time, it consumes network bandwidth and leads to service downtime. To model such trade-offs and find optimal solutions to the problem, a number of works have been proposed, as we detail in what follows.

Markov Decision Process (MDP) is a commonly used framework in literature for

the definition of migration policies. In (Ksentini et al., 2014), the authors model the service migration problem as a distance-based MDP. In this first proposal, user's mobility, which is not deterministic, is modelled and predicted through a one dimension (1D) mobility pattern. On the other hand, both (Taleb et al., 2016) and (Wang et al., 2015a) formulate the service migration problem as a distance-based MDP where 2D mobility scenarios are captured. The same authors of (Wang et al., 2015a) advance an alternative solution method in (Urgaonkar et al., 2015). They establish a decoupling property of their initial MDP which transforms it into two independent MDPs on disjoint state spaces. Lyapunov optimisation can then be applied so that what is obtained is a simple deterministic (rather than stochastic) optimisation problem. Another work from the same authors (Ciftcioglu et al., 2015) contextualises the mobility support issue in military environments rather than in commercial ones. Since military environments demand stronger security guarantees, a new parameter (i.e., security cost) is considered to make MDP-based migration decisions, together with the usual parameters (i.e., transmission and migration costs). The security cost of a migration increases when services of different users are hosted on the same physical node. In (Zhang et al., 2016), the authors employ an MDP to decide where (and not when) to migrate the Fog service; this work is worth mentioning because, unlike the aforesaid contributions, it also considers the network and FN states as parameters on which to base migration decisions. In (Plachy et al., 2016), which also uses MDPs, the authors propose to handle user's mobility by either migrating the Fog service or by finding a new, more suitable communication path between it and the mobile node. Although MDPs are by far the most common way to define a migration policy, they are not the only one. For example, in (Yao et al., 2015), the authors suggest making migration decisions in order to minimise the overall bandwidth consumption in a Fog-enabled Vehicular Cloud Computing (VCC) context; the problem is herein formulated as a Mixed-Integer Quadratic Programming (MIQP) problem.

Open issues - Even though there are several contributions in terms of migration policies in FC, there are still unexplored possibilities and room for further improvements. Firstly, it is worth highlighting that most of the discussed works do not make clear assumptions on (nor evaluate the impact of) the placement of the decision-making logic. However, aspects such as the timeliness of migration-related actions and the availability of insights from the decision-making logic could be much influenced by the placement of the logic (i.e., in the Cloud, in the Fog, on mobile devices). Another open issue that is worthy of investigation is the definition of migration policies that consider both end (user) devices and FNs as mobile devices. Although this is less common, also FNs may indeed be mobile, as it is proposed for example in (Hou et al., 2016) and in (Ye et al., 2016). Finally, another research direction pertains the When-to-Migrate decision. Such a decision should always consider

the wireless connection handoff (i.e., the change of the access point, due to mobility) as the cornerstone. This is because handoff is that mechanism that may considerably change the topological distance between the mobile device and the current FN. Based on the works in literature, a Fog service could be migrated either *proactively* or *reactively* with respect to the wireless handoff (as also described in Section 6.1). Proactively means that service migration occurs before the handoff, based on user's mobility prediction. If prediction is correct, proactive migration improves overall performances, as the service would be immediately available to the mobile device once this is connected to the new access point. However, performance might degrade if mobility prediction is erroneous. On the other hand, reactive migration consists in migrating the service only after the handoff has started. This approach has in general worse performance than the former (the mobile device connected to the new access point still has to access the Fog service on the old FN) but proves more beneficial when user's mobility is highly unpredictable. An interesting research direction would be the definition of an hybrid approach that combines the benefits of the two. In Chapter 5, we propose an example of reactive migration policy. However, we highlight that the main contribution of that chapter is not the policy but rather the migration platform.

3.5 Migration platforms

Migration platforms can be defined as software environments that provide all the functionalities required to support device mobility through service migration. Migration platforms are deployed over the distributed Fog infrastructure and usually include mechanisms to measure or predict aspects such as the status of network and computing resources as well as application-related metrics, on which migration decisions are based. Each migration platform implements one or more of the hosting and migration techniques from Section 3.3 and executes one or more migration policies. In what follows, we review the state-of-the-art migration platforms and point out the open issues and future research directions in the field.

In (Bao et al., 2017), the authors present Follow Me Fog (FMF), a FC platform in which a Software as a Service (SaaS) Server is hosted on each access point and provides resource-intensive services to mobile IoT devices. What is migrated here is not the actual service but the pending jobs offloaded by the mobile device. This platform migrates the jobs any time that a handoff occurs, which is not always necessary, and does not handle common scenarios in which there are two or more potential FNs given a specific access point. sFog (Bao et al., 2018) enhances FMF by introducing congestion control mechanisms. Follow Me Cloud (FMC) (Taleb and Ksentini, 2013) mainly focuses on content and session migration across FNs and heavily exploits elements and functionalities available in cellular networks.

In (Saurez et al., 2016), the authors propose Foglets. This platform uses an ap-

proach similar to mobile agents in order to implement service migrations: the execution state of a service is captured at a coarse granularity by the application itself and then migrated to a new node. On the contrary, (Bellavista et al., 2017) presents a platform capable of migrating the execution state of a service at a fine granularity, by proactively migrating VMs. More specifically, this proposal extends the Openstack++ Edge Computing platform (see Section 2.6) in order to enable mobility support. As in (Bao et al., 2017), the authors of this work only consider situations where for each access point there is only one FN, which is co-located with the access point. Furthermore, they do not contemplate parameters such as the state of FN hardware resources in their migration decision-making.

As we better detail at the beginning of Chapter 4, VMs and containers can be migrated either in a stateless or in a stateful way. The former approach is so called because it causes the loss of the whole VM/container state. Therefore, once on the destination FN, the service restarts from scratch. Stateful migration, instead, is such that the whole state of the VM/container is made available at destination once migration is completed. Some migration platforms use stateless container migration as service migration technique (Farris et al., 2017; Dupont et al., 2017; Zanni et al., 2018). Specifically, the first proposes a stateless replication of containers across FNs, while the other two destroy a container on the source node and statelessly reinstantiate it on the destination node. In particular, (Zanni et al., 2018) is worthy of notice, as it makes two novel contributions. Firstly, it implements a module on each FN that is not only able to monitor resource consumption but also to predict whether a FN is likely to become congested soon. Secondly, it conceives of Fog services as multi-container applications managed through Docker Compose[25]. Table 3.2 summarises the main aspects of the analysed platforms.

Open issues - By looking at Table 3.2, it is possible to note how the existing works do not implement stateful container migration in their platforms. As detailed in Section 3.3, containerisation in FC environments is drawing much interest, and container migration represents a lightweight approach to achieve strong mobility. Therefore, in Chapter 5, we propose and validate a migration platform that leverages containerisation as hosting technique and implements both stateful and stateless container migrations as migration approach. To conclude, another research opportunity is the definition, within migration platforms, of federation mechanisms among different Fog providers. Such mechanisms should allow mobile users that are associated to a Fog provider to utilise computing resources of other providers on the basis of pre-established Service Level Agreements (SLA). Federation may indeed prove beneficial, e.g., by granting access to a federated FN in an area where there are not available FNs of the original provider.

[25]See `https://docs.docker.com/compose/`. Last accessed: 28 October 2019.

Table 3.2: Comparison among the migration platforms.

Reference	Migrates	Policy	Evaluation
FMF/sFog (Bao et al., 2017, 2018)	Pending jobs (weak mobility)	Proactive	Testbed
FMC (Taleb and Ksentini, 2013)	Content and session (weak mobility)	Reactive	Simulator (ns-3)
Foglets (Saurez et al., 2016)	Execution state at a coarse granularity (weak mobility)	Proactive; Reactive	Testbed
(Bellavista et al., 2017)	VMs (strong mobility)	Proactive	Testbed
(Farris et al., 2017)	Containers (stateless replication)	Proactive	Testbed
Cloud4IoT (Dupont et al., 2017)	Containers (stateless destruction and re-instantiation)	Reactive	✗
(Zanni et al., 2018)	Containers (stateless destruction and re-instantiation)	Reactive	Testbed

3.6 Simulation of mobility and migration

Simulation can be a time- and cost-effective way to evaluate service migration solutions in FC environments (especially the large-scale ones) with mobile users. However, to the best of our knowledge and how we detail in what follows, the state-of-the-art Fog simulators do not provide this possibility.

VirtFogSim (Scarpiniti et al., 2019) does not model aspects such as service migration, energy consumption, or pricing. However, it dynamically tracks the energy-delay application performance against abrupt changes due to failures or device mobility, e.g., mobility-induced changes of the available uplink/downlink bandwidth. The most distinctive functionality of VirtFogSim is that it allows to model cellular networks, which is useful when simulating 4G/5G scenarios. VirtFogSim is currently the only simulator that explicitly provides such a feature. Besides, this simulator includes a Graphical User Interface (GUI) that shows the simulation results in tabular, bar-chart, and coloured map graph formats.

Yet Another Fog Simulator (*YAFS*) (Lera et al., 2019) is a Python simulator for FC environments. It is particularly good at modelling network failures and therefore allows to evaluate service placement solutions in failure cases or to design robust networks. Network failures may be modelled in two possible ways: (i) through the runtime creation/deletion of FNs and network links; (ii) through *custom processes*, namely functions invoked at runtime for the implementation of real events. YAFS models mobility, sensors, and actuators but does not model aspects such as energy consumption or service migration. Finally, although it does not include a GUI for the description of Fog network topology, YAFS allows to import a simulation scenario as a JSON file.

The main objective of *FogNetSim++* (Qayyum et al., 2018) is to overcome the

limitations of the other simulators in network modelling. They do not (or only partially) take into account real-network properties and therefore simulate idealistic networks where no packet loss, congestion, or channel collision happen. Instead, FogNetSim++ extends OMNeT++[26], which is a well known framework for building network simulators, to model all these aspects. Moreover, it includes popular communication protocols for simulation, such as TCP, UDP, MQTT, and CoAP. Furthermore, FogNetSim++ models several other aspects, such as energy consumption, pricing, mobility, and handoff mechanisms.

FogTorch (Brogi and Forti, 2017b) is a Java tool that outputs all the possible deployments of application modules over a FC infrastructure, provided: (i) the specification of the application requirements; (ii) the description of the infrastructure, in terms of devices and network links; and (iii) the definition of a deployment policy. The FogTorch user then selects the best deployment out of the proposed alternatives. *FogTorch*Π (Brogi et al., 2017) is an extension of FogTorch that uses Monte Carlo simulations to model variations over time of the QoS of network links, which is expressed in terms of latency and bandwidth.

Table 3.3: Overview of the FC simulators.

	FogNetSim++ [27]	**FogTorch/ FogTorchΠ** [28]	**VirtFogSim** [29]	**YAFS** [30]	**EdgeCloudSim** [31]	**iFogSim** [32]
Mobility/ Handoff	✓		✓	✓	✓	
Service migration						
Cellular network			✓			
Energy consumption	✓		✓			✓
Security						
Failures			✓	✓		
Operational costs		✓				✓
Pricing	✓					✓
IoT	✓	✓		✓		✓
GUI	✓		✓			✓
Extends	OMNeT++				CloudSim	CloudSim
Language	C++	Java	MATLAB	Python	Java	Java
Released	2018	2017	2019	2018	2017	2016

[26] See `https://omnetpp.org/`. Last accessed: 25 October 2019.

iFogSim (Gupta et al., 2017) was the first FC simulator. It is implemented in Java as an extension of the most popular CC simulator, *CloudSim*[33]. iFogSim allows to test resource management and service placement strategies in terms of: (i) service latency; (ii) service throughput; (iii) network usage; (iv) energy consumption; (v) operational costs; and (vi) pricing. iFogSim provides a GUI to describe Fog network topologies (i.e., sensors, actuators, FNs, Cloud DC, and interconnections among them). Topologies can then be exported as a JSON file and imported again in a later moment. iFogSim presents several strengths. Firstly, iFogSim provides the widest range of functionalities, which span from network and resource management modelling to energy consumption and operational costs modelling. Secondly, it is by far the most used FC simulator in literature, which makes it the best candidate to compare own solutions with the related work. Thirdly, iFogSim extends CloudSim, which is the most popular CC simulator. Therefore, it is relatively easy to use for all those who have already had experience with CloudSim. However, iFogSim also has some limitations. The most evident is probably the lack of a detailed and consistent documentation, which might make iFogSim hard to use for those who have never worked with CloudSim before. Besides, network modelling in iFogSim is rather simplistic, as this simulator does not deal with real-networks aspects such as packet loss, network congestion, and channel collisions. Furthermore, iFogSim does not model mobility scenarios and service migration among FNs.

EdgeCloudSim (Sonmez et al., 2018) is another prominent simulator for FC environments. Also EdgeCloudSim is an extension of CloudSim. EdgeCloudSim models network delays more accurately than iFogSim, which considers network delays to be always fixed. EdgeCloudSim, instead, includes a networking module that calculates network delays, based on the current network load, when data need to be sent over the network. Even though EdgeCloudSim provides a better network modelling, it does not provide the same range of functionalities that are available in iFogSim. For example, energy consumption, operational costs, and pricing modellings are all missing in EdgeCloudSim. Device mobility is modelled, but service migration is not. Hence, EdgeCloudSim may be useful to those who work in Java and are mostly focused on evaluating network metrics (e.g., service latency, network usage).

Open issues - Table 3.3 summarises the main characteristics of the discussed simulators. As shown, most of the simulators model device mobility and handoff mech-

[33]See `https://github.com/Cloudslab/cloudsim`. Last accessed: 25 October 2019.

[28]See `https://github.com/rtqayyum/fognetsimpp`. Last accessed: 26 October 2019.

[29]See `https://github.com/di-unipi-socc/FogTorch` and `https://github.com/di-unipi-socc/FogTorchPI/tree/multithreaded`. Last accessed: 26 October 2019.

[30]See `https://github.com/mscarpiniti/VirtFogSim`. Last accessed: 26 October 2019.

[31]See `https://github.com/acsicuib/YAFS`. Last accessed: 26 October 2019.

[32]See `https://github.com/CagataySonmez/EdgeCloudSim`. Last accessed: 26 October 2019.

[33]See `https://github.com/Cloudslab/iFogSim`. Last accessed: 26 October 2019.

anisms, but none of them models service migration in FC environments. This is an open issue for which we propose a solution in Chapter 6. Moreover, we highlight that security modelling is still missing in the state-of-the-art FC simulators and therefore represents another research direction.

Chapter 4

Performance evaluation of container migration in Fog Computing

Containers and container migration are promising technologies for FC contexts. As discussed in Section 3.3, containers are a lightweight and performing hosting environment, and container migration may represent an efficient way to achieve strong mobility. However, container migration is at a preliminary stage, and quantitative comparisons among the state-of-the-art container migration techniques are still missing in literature. In this chapter, we investigate and quantitatively evaluate container migration, with a focus on FC environments. The objective is to determine whether there exists a technique that always performs the best or, otherwise, to delineate which container migration technique might be the most appropriate under certain network and service conditions.

Note that, in this chapter, we only consider techniques of **stateful container migration**, namely approaches that allow the whole state of the container to be available on the destination FN once migration is finished. Hence, in what follows, we do not deal with stateless container migration, which instead causes the loss of the whole container state[34]. Moreover, we consider only the runtime (i.e., volatile) state, and not the persistent one, of the container to be migrated between FNs. While it is not possible to make this assumption in Cloud DCs, Fog environments are characterised by services that typically perform transient data analysis and time-critical control and are thus not supposed to write to any persistent memory (see Table 2.3 and Section 3.3). As a result, we transfer only the runtime state and use this, together with a base service image representing the default disk state, to restore the container at destination.

The rest of the chapter is organised as follows. Section 4.1 provides a descriptive overview of the existing stateful container migration techniques, highlighting their characteristics and the differences among them. Then, Section 4.2 discusses the rela-

[34]Stateless migration simply consists in the following two steps: (i) the start of a new container on the destination FN; (ii) the deletion of the old container on the source FN.

tionship and the differences between runC and Docker containers with the purpose to clarify why we chose to implement the former type of containers. Next, Section 4.3 reports the experiment setup and presents the real testbed that we used to evaluate container migration techniques. Finally, Section 4.4 analyses the obtained results and elaborates on the lessons learnt.

4.1 Stateful container migration techniques

As we detail in this section, stateful container migration techniques can be distinguished in *cold* and *live* techniques. The former is so called because it stops the execution of the container during the entire migration process. On the contrary, limitation of service downtime is the primary concern of live migration techniques (i.e., pre-copy, post-copy, and hybrid migrations). With live techniques, the container keeps on running while most of its state is being transferred to the destination FN. The container is typically suspended only for the transmission of a minimal amount of the overall state, after which the container runs at destination. When the downtime is not noticeable by the end user, live migration is said to be "seamless". For the sake of clarity, Table 4.1 at the end of this section compares the discussed techniques in terms of what is transmitted in each migration phase.

Cold migration

The steps for cold migration are depicted in Figure 4.1, where the source FN performs the blue steps, the destination FN performs the green one, and the grey step involves both nodes. This migration technique is said to be "cold" because it: (i) first freezes/stops the container to ensure that it no longer modifies the state; (ii) then dumps the state and transfers it while the container is stopped; and (iii) finally resumes the container at destination only when all the state is available. As such, cold migration features a very long downtime. As shown in Figure 4.1, downtime even coincides with the total migration time. This aspect of cold migration goes against one of the main objectives of service migration in both the Cloud and the Fog, i.e., the limitation of downtime. However, we highlight that this technique transfers each memory page only once, and this should significantly reduce both the total migration time and the overall amount of data transferred during migration.

Cold migration of containers, like all the other container migration techniques, is strongly based on CRIU[35]. This started as a project of Virtuozzo[36] for its OpenVZ containers but has been getting so much popularity over time that now it is used by all the most prominent container runtimes, such as runC. In detail, CRIU is a software tool for Linux that is written in C and mainly allows to: (i) freeze a running

[35]See `https://criu.org/Main_Page`. Last accessed: 14 January 2019.
[36]See `https://virtuozzo.com/`. Last accessed: 14 January 2019.

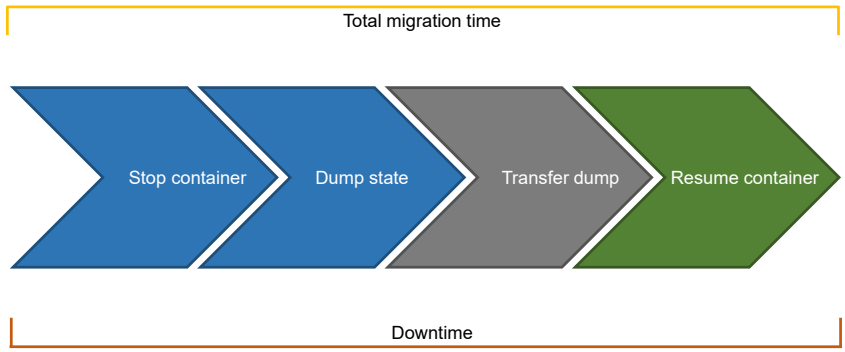


Figure 4.1: Cold migration.

container; (ii) checkpoint/dump its runtime state as a collection of files on disk; and (iii) use those files to restore the container and run it exactly as it was before being stopped. With respect to Figure 4.1, CRIU misses the third step, namely the state transfer to the destination node. Indeed, CRIU by itself only allows to restore a checkpointed container on the same host node; therefore, the actual state transfer has to be performed by exploiting tools such as *rsync* or Secure Copy Protocol (*SCP*).

Pre-copy migration

Pre-copy migration is so called because it transfers most of the state prior (i.e., *pre*) to freezing the container for a final dump and state transfer, after which the container runs on the destination node. It is also known as *iterative migration*, since it may perform the pre-copy phase through multiple iterations such that each iteration only dumps and retransmits those memory pages that were modified during the previous iteration (the first iteration dumps and transfers the whole container state as in cold migration. The modified memory pages are called **dirty pages**. Typically, iterations in the pre-copy phase are convergent, i.e., of shorter and shorter duration. If iterative, the pre-copy phase generally concludes when a predetermined number of iterations is reached. The container is then suspended on the source node in order to capture the last dirty pages along with the modifications in the execution state (e.g., changes in the CPU state, changes in the registers) and copy them at destination without the container modifying the state again. Finally, the container resumes on the destination node with its up-to-date state. Figure 4.2 shows the steps of pre-copy migration with a one-iteration pre-copy phase. The reason for this is that our current implementation of pre-copy migration, which is then evaluated in Section 4.4, presents only one pre-copy iteration. This implementation is based on CRIU, which provides all the basic mechanisms (e.g., the *--pre-dump* option) that are nec-

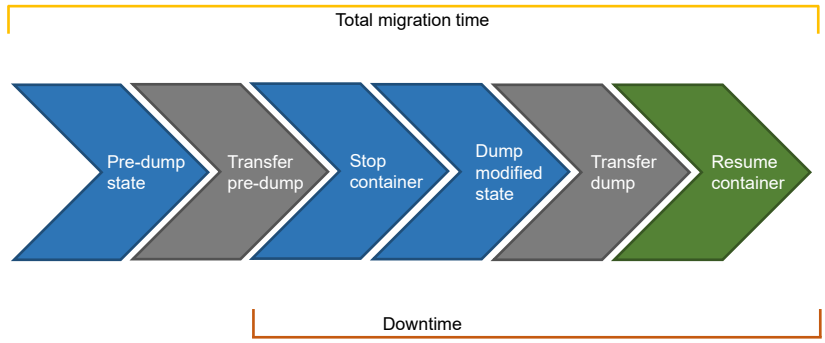


Figure 4.2: Pre-copy migration.

essary to pre-dump the runtime state of a container and restore it afterwards[37]. All the following considerations on pre-copy migration and hybrid migration assume a one-iteration pre-copy phase.

The main difference between cold and pre-copy migrations lies in the nature of their dumps. The dump in cold migration represents the whole container state (as the pre-dump in pre-copy migration) and thus always includes all the memory pages and the execution state. The dump in pre-copy migration, instead, only includes those memory pages that were modified during the pre-copy phase, together with the changes in the execution state. As such, downtime for pre-copy migration should be in general shorter than that for cold migration because less data are transferred while the container is stopped. However, downtime for pre-copy migration is not deterministic, as it significantly depends on the number of dirty pages. Therefore, we expect pre-copy migration to be afflicted by the two factors that may increase the number of dirty pages: (i) the **page dirtying rate** featured by the container-hosted service, namely the speed at which the service modifies memory pages; (ii) the **amount of data** that are transferred during the pre-copy phase, since the more data are transferred in that phase, the more time the service has to modify pages. It is worth noting that both these factors should be always considered against the available **throughput** between the source and the destination node. Specifically, this is more obvious for the second factor, since this needs to be considered against the available throughput in order to estimate the time that the service has to modify pages during the pre-copy phase. With regard to the first factor, we expect that pre-copy migration performances are impaired by a page dirtying rate

[37] See `https://criu.org/Iterative_migration`. Last accessed: 15 January 2019.

that starts approaching the available throughput (i.e., same order of magnitude), as in that case memory pages are modified at a rate which is comparable to that of page transfer. This would result in a state dump that is in general comparable to the state pre-dump or, in other words, to a downtime which is comparable to that for cold migration. As a final remark, we highlight that, unlike cold migration, pre-copy migration might transfer each memory page several times, with possible negative consequences on the overall amount of data transferred during migration and thus on the total migration time.

Post-copy migration

Post-copy migration is the exact opposite of pre-copy migration. Indeed, it first suspends the container on the source node and copies the execution state (i.e., CPU state, registers) to the destination so that the container can resume its execution there. Only after that (i.e., *post*), it copies all the remaining state, namely all the memory pages. Actually, there exist three variants of post-copy migration, which differ from one another on how they perform this second step. In this thesis, we only describe the *post-copy migration with demand paging* variant, better known as **lazy migration** (see Figure 4.3), which is the only one that may be currently implemented using the functionalities provided by CRIU[38], e.g., the *--lazy-pages* and *--page-server* options. With lazy migration, the resumed container tries to access memory pages at destination, but, since it does not find them, it generates **page faults**. The outcome is that the *lazy pages daemon* at destination contacts the *page server* on the source node. This server then "lazily" (i.e., only upon request) forwards the faulted pages to the destination.

Post-copy migration copies each memory page only once. Therefore, it should transfer a data volume that is comparable with that of cold migration and with that of the pre-copy phase of pre-copy migration. Besides, similarly to cold migration, downtime for post-copy migration is irrespective of the page dirtying rate featured by the container-hosted service and of the overall amount of data that need to be transferred. This is due to the fact that the dump in post-copy migration is simply the execution state and does not contain any dirty memory pages. However, post-copy migration is afflicted by two drawbacks that are worthy of consideration. Firstly, **page faults degrade service performances**, as memory pages are not immediately available at destination once the container resumes. This could be unacceptable to the many latency-sensitive services present in FC environments. Secondly, during migration, this technique **distributes the overall up-to-date state** of the container between both the source and the destination node (before completion of post-copy migration, the source node retains all the memory pages, but some of them may be out-of-date because they have already been copied at destination and modified by

[38] See `https://criu.org/Lazy_migration`. Last accessed: 15 January 2019.

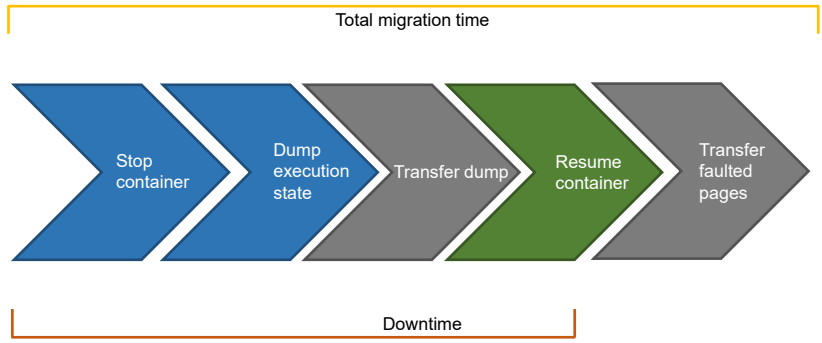


Figure 4.3: Post-copy migration.

the resumed container), whereas approaches like cold or pre-copy migrations retain the whole up-to-date state on the source node until the termination of the migration process. Therefore, if the destination node fails during migration, it may be no more possible to recover the up-to-date state of a post-copied container.

Hybrid migration

As discussed, both pre-copy and post-copy migrations present some shortcomings: (i) pre-copy migration has a non-deterministic downtime; (ii) faulted pages in post-copy migration degrade service performances. Hybrid migration, which is illustrated in Figure 4.4, combines pre-copy and post-copy with the objective to subdue their limitations and sharpen their strengths. Going into detail, the first two steps of hybrid migration coincide with those of pre-copy migration, namely a pre-dump of the whole state and its transmission at destination while the container is still running on the source node. Then, the container is stopped, and its state is dumped in a way that combines the dumps of pre-copy and post-copy migrations. Indeed, the dump in hybrid migration is represented by the modifications in the execution state that occurred during the pre-copy phase. Once the dump is transferred to the destination node, the container can be restored. At this step, the destination node has the up-to-date container execution state along with all the memory pages. Nonetheless, some of them were dirtied during the pre-copy phase. As a result, the last step in hybrid migration consists in the page server at source lazily transmitting the dirty pages to the lazy pages daemon on the destination node. It is worth noting that the number of memory pages that are lazily transmitted in hybrid migration is generally less than that of post-copy migration since only the dirty pages are transferred.

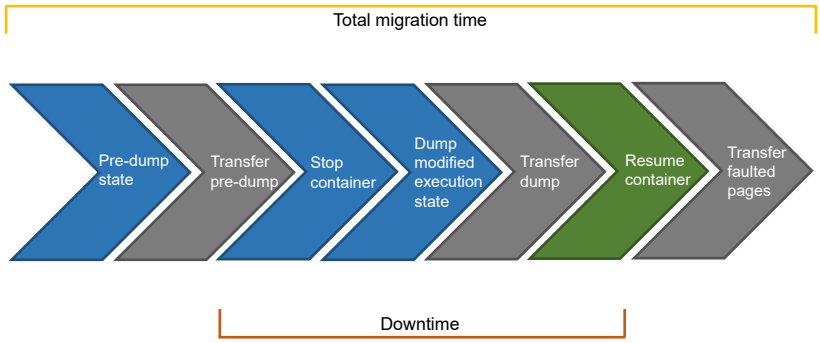


Figure 4.4: Hybrid migration.

From now on, we will refer to these dirty pages as faulted pages, in line with the name used in post-copy migration to indicate the data transferred in this last phase. As pre-copy migration, we expect also hybrid migration to be affected by the page dirtying rate and the amount of data that are transmitted during the pre-copy phase. However, these factors should influence the total migration time for hybrid migration but not the downtime, as they alter the number of faulted pages. Moreover, hybrid migration is affected by the same two drawbacks of post-copy migration. To conclude, hybrid migration can be implemented by merging the CRIU options for pre-copy and post-copy migrations.

Table 4.1: What is transferred in each phase of the migration techniques.

Technique	**Pre-Dump**	**Dump**	**Faulted Pages**
Cold		Memory pages and execution state	
Pre-copy	Memory pages and execution state	Dirty pages and changes in execution state	
Post-copy		Execution state	Memory pages
Hybrid	Memory pages and execution state	Changes in execution state	Dirty pages

4.2 runC and Docker containers

To manage containers, there exist several platforms and runtimes. Docker[39] is currently the most popular of such platforms and is used in both industrial and academic works. However, in this thesis, we run containers by leveraging the runC[40] container runtime rather than Docker. In this section, we discuss runC and its relationship with Docker, illustrating the advantages and disadvantages of using runC as a standalone tool. The goal of this section is to present runC and explain our implementation choice.

Docker is an open platform that manages the life cycle of containers (i.e., creation, run, deletion). In its infancy, Docker was a monolithic platform. However, in 2016 (starting from Docker v1.11), it was split into several components (A. Varkockova, 2018). Figure 4.5 depicts the resulting Docker architecture. In what follows, we only focus on the relationship between Docker and runC, without providing the details of each layer of the architecture. The core of Docker, known as *Docker Engine*, is based on a client-server architecture where the *Docker CLI* (i.e., the client) uses the REST API exposed by the *Docker daemon* (i.e., the server) to instruct it on what to do (Docker, 2017b). Upon request of the client, the Docker daemon (through *containerd* and *containerd-shim*) interacts with a container runtime. The latter leverages all the Linux kernel facilities (e.g., cgroups, namespaces) that are necessary to manage the life cycle of containers. In the very beginning, Docker exploited LinuX Containers (*LXC*) as container runtime. However, LXC was later replaced by *libcontainer*. In June 2015, the Open Container Initiative (*OCI*[41]) was established as a Linux Foundation project to promote open standards around containerisation technology. Docker donated the libcontainer project, as well as all the modifications needed to run it independently of Docker, to the OCI (Open Container Initiative, 2017). The result of this collaboration was runC, a command-line tool that includes and directly interacts with libcontainer to spawn and run containers according to the OCI specifications. Since then, runC has become the most popular container runtime (D. J. Walsh, 2018).

runC may be used as part of the Docker platform or as a standalone tool to directly manage containers. The second alternative has some advantages. Firstly, it allows to run containers without the overhead of the whole Docker platform and, being runC a lower-level implementation, represents a better choice for debugging purposes. Secondly, runC should be preferred to Docker when trying to use new tools and features for containers. This is because such tools are first tested on runC and, only in a later moment, are ported to Docker. One of these tools is CRIU, which, as described in Section 4.1, is essential to perform stateful container migrations. After several attempts, we were able to use CRIU within Docker only to perform cold

[39] See `https://docs.docker.com/`. Last accessed: 06 March 2019.
[40] See `https://github.com/opencontainers/runc`. Last accessed: 06 March 2019.
[41] See `https://www.opencontainers.org/`. Last accessed: 06 March 2019.

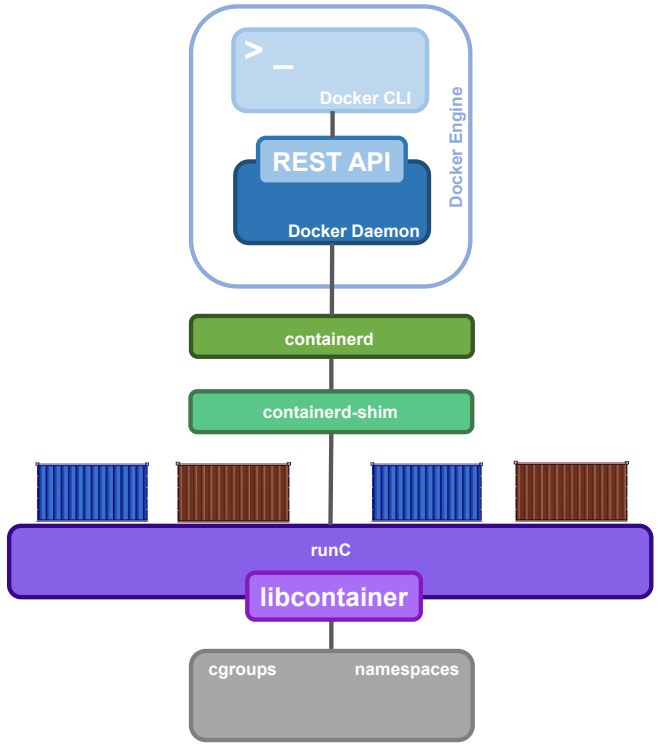


Figure 4.5: Docker layered architecture.

migrations but failed with live migration techniques. On the other hand, we successfully leveraged CRIU within runC to migrate containers according to both cold and live migration techniques. This is the main reason why we work with runC containers rather than Docker.

However, using runC as a standalone tool has also some disadvantages. Being built on top of runC, Docker hides lower-level details, thus allowing to configure containers in an easier way. Furthermore, it leverages all the functionalities of runC and enriches them with additional features. For example, Docker introduces the concept of *layered storage* (Docker, 2017a), which organises the storage of a container base image in read-only layers such that each layer is the set of differences from the layer underneath. When a Docker container is started from a base image, a new writable layer called *container layer* is created for that container. Any changes (i.e., creation, modification, deletion of files) made by the container are written to its container layer. Once the container is deleted, also its container layer is deleted, and the underlying base image remains unchanged. This allows multiple containers to share the same base image, hence guaranteeing resource efficiency. On the contrary,

runC does not have the layered storage functionality. As such, any changes made by a runC container to the filesystem persist after the container deletion and would be visible to other containers that are instance of the same image. To run multiple runC containers from the same *OCI image* (i.e., the base image in runC), it is therefore necessary to first unpack the image to a separate filesystem (i.e., *OCI bundle*) for each container; then, each container is run from its own OCI bundle. This solution causes a significant storage overhead.

4.3 Experiment setup

This section discusses the experiment setup and illustrates the real testbed that we used to evaluate the existing container migration techniques in the Fog.

The overall testbed, which is depicted in Figure 4.6, comprises two FNs and one end device. The former are two **Raspberries Pi 3 Model B** (i.e., ARMv8-A architecture) with Debian 9.5 and Linux kernel 4.14.73-v8+. They both run: (i) **CRIU 3.10** for the checkpointing and restore functionalities; (ii) **rsync 3.1.2** as file transfer mechanism; and (iii) **runC 1.0.1** as container runtime. Besides, they are both deployed within an office in the Department of Information Engineering of the University of Pisa. On the other hand, an *Asus ZenBook* notebook with Windows 10 emulates an IoT device that forwards data to a container-hosted Fog service and

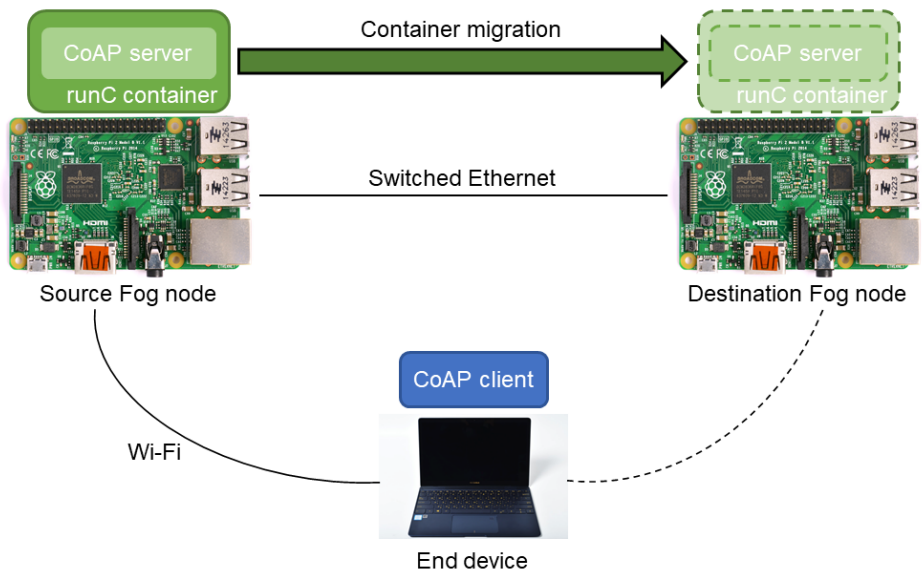


Figure 4.6: Overview of the real testbed.

receives actuation commands from it.

The core part of the experiment setup consisted in the appropriate tuning of the **throughput** between the FNs as well as in: (i) the choice of the **runtime state size** of the container[42]; (ii) the selection of the values for the **page dirtying rate** of the container-hosted service. Indeed, as explained in Section 4.1, we expect pre-copy and hybrid migrations to be afflicted by these two factors, both to be considered against the available throughput. Instead, we do not expect these or any other specific factor to affect cold or post-copy migrations. By calibrating and combining the aforementioned factors, we implemented the following **four configurations**, which allow to carry out a comprehensive performance evaluation:

- **A -** this configuration presents a page dirtying rate and a throughput of different orders of magnitude, with the throughput higher than the page dirtying rate. However, given the throughput, the size of the runtime state leads to a prolonged pre-copy phase and thus gives the service plenty of time to modify memory pages. Therefore, this configuration is mainly aimed at evaluating the effects of a considerable runtime state size on the migration techniques;
- **B -** this configuration resembles configuration A in terms of runtime state size but features a page dirtying rate of the same order of magnitude of the available throughput. As such, this configuration is mainly aimed at investigating the effects of a high page dirtying rate on the migration techniques;
- **C -** this configuration shows a runtime state size that, considering the throughput, causes a shortened pre-copy phase and hence gives the service little time to modify memory pages. Besides, the page dirtying rate and the throughput are of different orders of magnitude, with the throughput higher than the page dirtying rate. Thus, the main objective of this configuration is to assess the migration techniques when both the factors are low, given the available throughput;
- **D -** this configuration resembles configuration C. The only difference is that the page dirtying rate in C and in D are of different orders of magnitude, with the former lower than the latter. However, the page dirtying rate in D is still lower than the throughput and of a different order of magnitude. The main purpose of this configuration is to estimate whether there are evident effects on the migration techniques when the page dirtying rate increases, though still being considerably lower than the throughput.

Going into detail, we achieved the previous four configurations by choosing and combining values as follows. Firstly, we selected two different throughput values

[42]We highlight that, given the rsync settings described at the end of this section, the runtime state size represents the actual amount of data transferred during the pre-copy phase. Therefore, throughout this chapter, we use these two terms interchangeably.

that may both occur in real-life use cases within a Fog environment. One throughput value may be present in a use case where the destination FN connects to the Internet through 4G/LTE (e.g., a FN within a public bus) - we refer to this use case as *LTE use case*. The other throughput value, instead, may exist in another use case where both FNs connect to the Internet through Ethernet (i.e., *Ethernet use case*). Table 4.2 reports these values, together with the RTT values associated to them in the aforementioned use cases. Indeed, we also considered RTT values among the FNs in order to more accurately replicate the use cases network conditions in our testbed. To obtain the values presented in Table 4.2, we proceeded as follows. For the LTE use case, we considered a computer connected through Ethernet to the University of Pisa network as source FN and a smartphone connected to the Internet through 4G/LTE as destination. For the Ethernet use case, instead, we employed two fixed computers belonging to a bridged LAN of the University of Pisa and installed in two different buildings placed about 1 km far apart. Throughput values were then obtained by performing 10 runs with the *iperf3*[43] tool, sending 50 MB each time. Similarly, RTT values were calculated over 10 runs, with 20 measurements per run. Since, in our testbed, the two Raspberries behaving as FNs are located in the same office, we had to emulate the values described in Table 4.2. Therefore, we exploited Linux Traffic Control Hierarchy Token Bucket (*tc-htb*)[44] to limit the throughput and Linux Traffic Control Network Emulator (*tc-netem*)[45] to artificially set RTT values between the Raspberries.

Table 4.2: Considered throughput and RTT values (95% confidence intervals).

Use Case	**Throughput (Mbps)**	**RTT (ms)**
LTE	11.34 ± 2.31	122.95 ± 5.57
Ethernet	72.41 ± 3.87	6.94 ± 0.61

We then chose the other values based on the identified throughput values. Specifically, we implemented a distributed application where both the client and the server are written in Java using the *Californium*[46] *CoAP* framework. This required the installation of *openjdk8* on all the devices of the testbed. CoAP is a specialised Web transfer protocol for use with constrained devices and constrained networks in the IoT. CoAP is inspired by HyperText Transfer Protocol (HTTP) and is therefore based on the Representational State Transfer (REST) paradigm. According to this paradigm, servers expose resources under a Uniform Resource Locator (URL), and clients access these resources using one of the following methods: GET, PUT, POST, and

[43] See `https://iperf.fr/`. Last accessed: 30 December 2018.

[44] See `https://linux.die.net/man/8/tc-htb`. Last accessed: 30 December 2018.

[45] See `https://www.systutorials.com/docs/linux/man/8-tc-netem/`. Last accessed: 30 December 2018.

[46] See `https://www.eclipse.org/californium/`. Last accessed: 15 January 2019.

DELETE (Bormann, 2014). Let us now describe the considered client-server application in more detail. The server runs within a runC container in the Fog and, once started, allocates 75 MB of RAM for random data. For our purpose, 75 MB is a suitable runtime state size; indeed, it determines a pre-copy phase that lasts tens of seconds with the lowest throughput and only few seconds with the highest one. The client, instead, runs on the Asus notebook and sends a POST request to the server every second to represent sensor data. Every time the server receives a request from the client, it modifies some of the memory pages with new random values. The server may perform this task with the following two page dirtying rates, which were identified by taking the throughputs into consideration. The lowest page dirtying rate is 10 KBps, which is about two orders of magnitude lower than the LTE use case throughput and about three orders of magnitude lower than the Ethernet use case throughput. The highest page dirtying rate, instead, is 500 KBps, which is of the same order of magnitude of the throughput in the LTE use case and about one order of magnitude lower than the throughput in the Ethernet use case. By combining the aforementioned values as reported in Table 4.3, we obtained the four configurations that were previously described. We highlight that all the implemented configurations share the same value of the runtime state size.

During the experiments, we evaluated all the migration techniques that are discussed in Section 4.1. In particular, we tested each technique five times for each of the four configurations. For each migration technique, we observed all the metrics that characterise it. As a result, the metrics that were overall observed are the following:

- **Pre-dump time -** taken in the pre-copy phase to dump the whole state on the source node while the service is still running;
- **Pre-dump transfer time -** needed in the pre-copy phase to transfer the generated pre-dump from the source to the destination node. It is not to be confused with the pre-dump time;
- **Dump time -** necessary in the dump phase to stop the container and dump its (modified) state on the source node. As described in Table 4.1, each migration

Table 4.3: How values were combined to obtain the four configurations.

Throughput and RTT	Page Dirtying Rate	Configuration
LTE	Low	A
LTE	High	B
Ethernet	Low	C
Ethernet	High	D

technique presents a different concept of state dump;

- **Dump transfer time -** needed in the dump phase to transfer the generated dump from the source to the destination node. It is not to be confused with the dump time;
- **Resume time -** taken to restore the container at destination based on the state that was transferred up to that moment;
- **Faulted pages transfer time -** required in the last phase to transfer the faulted pages from the source to the destination node. Table 4.1 presents the different meanings that the term "faulted pages" assumes for post-copy and hybrid migrations;
- **Pre-dump size -** transferred during the pre-dump transfer time;
- **Dump size -** sent from the source to the destination node during the dump transfer time;
- **Faulted pages size -** transferred during the faulted pages transfer time.

We exploited the Linux *time* command to measure the times (i.e., the first six metrics) and the rsync *--stats* option to collect statistics regarding the amount of data transferred through rsync (i.e., the last three metrics). It is worth noting that we disabled the rsync data compression functionality during all the experiments. This was done because compressibility depends on data, and we did not want the experiment results to be influenced by this aspect. Similarly, we deleted the dump and the eventual pre-dump from the destination FN after every experiment run. This was done to avoid that, by finding any of them at destination the next time, the incremental data transfer performed by rsync could transmit less data, thus influencing the experiment results. To conclude, we stored raw data in .csv files for the next phase of results analysis and plotting, which was performed in Python through *Plotly*, an open-source graphing library for Python.

4.4 Results

We now analyse and discuss the results obtained from the experiments. More specifically, we compare the container migration techniques in terms of: (i) total migration times; (ii) downtimes; and (iii) volumes of transferred data. As we clarify in what follows, each of these three metrics is the result of adding up some of the metrics from Section 4.3. All the following results are presented with a 95% confidence level. At the end of this section, we summarise the main lessons learnt.

Total migration times

Figure 4.7 depicts the total migration times, highlighting their components for each migration technique. It is evident how the times to perform local computations (i.e., pre-dump and dump the state, resume the container) are negligible with respect to those needed to transfer the state at destination (i.e., pre-dump transfer, dump transfer, and faulted pages transfer times). This is clearly more visible under configurations A and B, where the available throughput is significantly lower than that of configurations C and D.

Let us now compare the migration techniques. Cold migration presents the lowest total migration times, irrespective of the specific configuration. We were expecting this result, as cold migration transmits each memory page only once unlike pre-

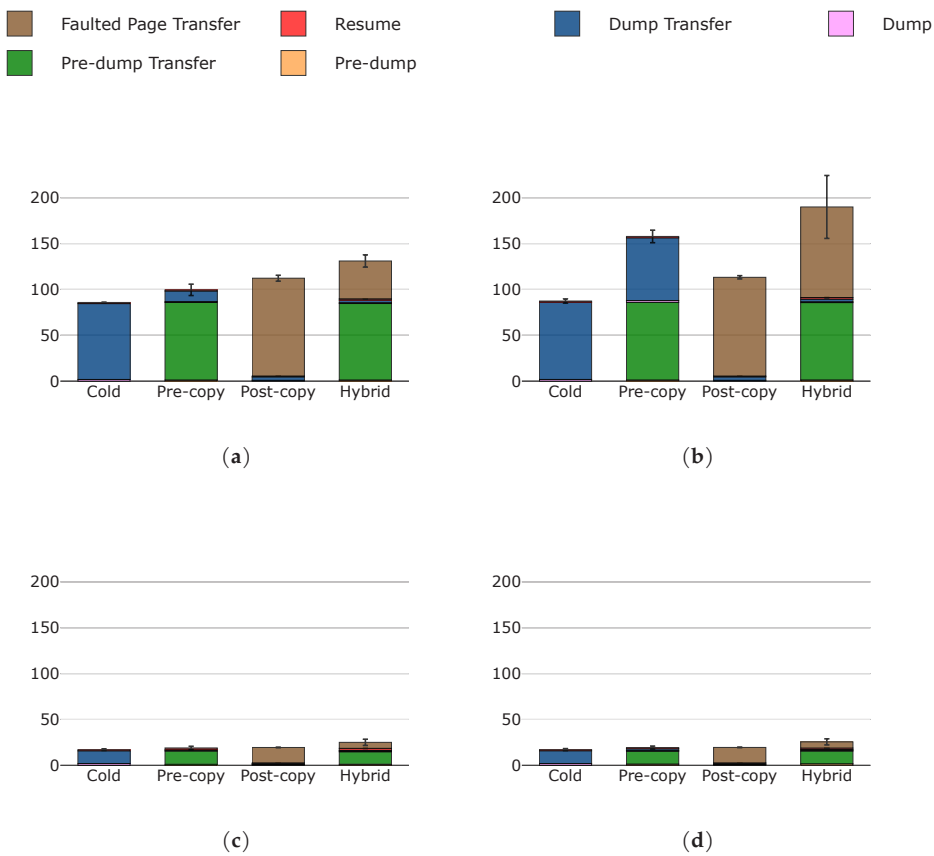


Figure 4.7: Total migration times (s) with their components in evidence under: (**a**) configuration A; (**b**) configuration B; (**c**) configuration C; and (**d**) configuration D.

copy and hybrid techniques, which instead may transmit a memory page more than once (i.e., if it is dirtied). Less pages transferred result in a shorter total migration time, indeed. However, this is not always true. In Figure 4.7a, post-copy migration presents a longer total migration time than pre-copy, even though it transmits less data (as described later in this section). Similarly, total migration times for post-copy migration are always longer than those for cold migration, even though these two techniques overall transmit similar amounts of data, as shown later. This unexpected result can be explained as follows. Post-copy migration is currently implemented according to the "lazy migration" variant, and hence faulted pages are transferred by the page server on the source node only upon request of the lazy pages daemon running at destination (see Section 4.1). Thus, **the time to perform such requests, which are not present in cold and pre-copy migrations, increases the overall total migration time**. This is particularly noticeable under configurations A and B, where RTT between the FNs is considerably higher than that of C and D.

Total migration times for cold and post-copy migrations are never influenced by an increase in the page dirtying rate. We expected this result as neither of these two techniques transfers dirty pages in any of its phases (see Table 4.1). Also pre-copy and hybrid migrations are not affected in terms of total migration time when page dirtying rate raises from configuration C to D. This important result shows how **there are no evident effects on the total migration times for pre-copy and hybrid migrations when the page dirtying rate increases, though still being lower and of a different order of magnitude from the throughput.** Besides, under these conditions, pre-copy migration performs similarly to post-copy in terms of total migration time. Nonetheless, the increment in the page dirtying rate from configuration A to B significantly prolongs the total migration times for pre-copy and hybrid migrations. Therefore, these two techniques are strongly affected in terms of total migration time by a page dirtying rate that, as for example under configuration B, reaches the same order of magnitude of the available throughput. The reason for this is that, under these conditions, the amount of pages dirtied during the pre-copy phase is comparable to that of pages transferred in that phase, namely to the whole state. This results in a state dump of considerable size for pre-copy migration and in a significant number of faulted pages for hybrid migration (see Table 4.1) and therefore in a substantial protraction of the total migration times.

Finally, we remark that hybrid migration always has the longest total migration times. This is because this technique inherits both the drawbacks of pre-copy and post-copy migrations in terms of total migration time, namely: (i) the fact that a memory page may be transferred more than once, as in pre-copy migration; (ii) the fact that also the time needed to request faulted pages needs to be considered, as in post-copy migration.

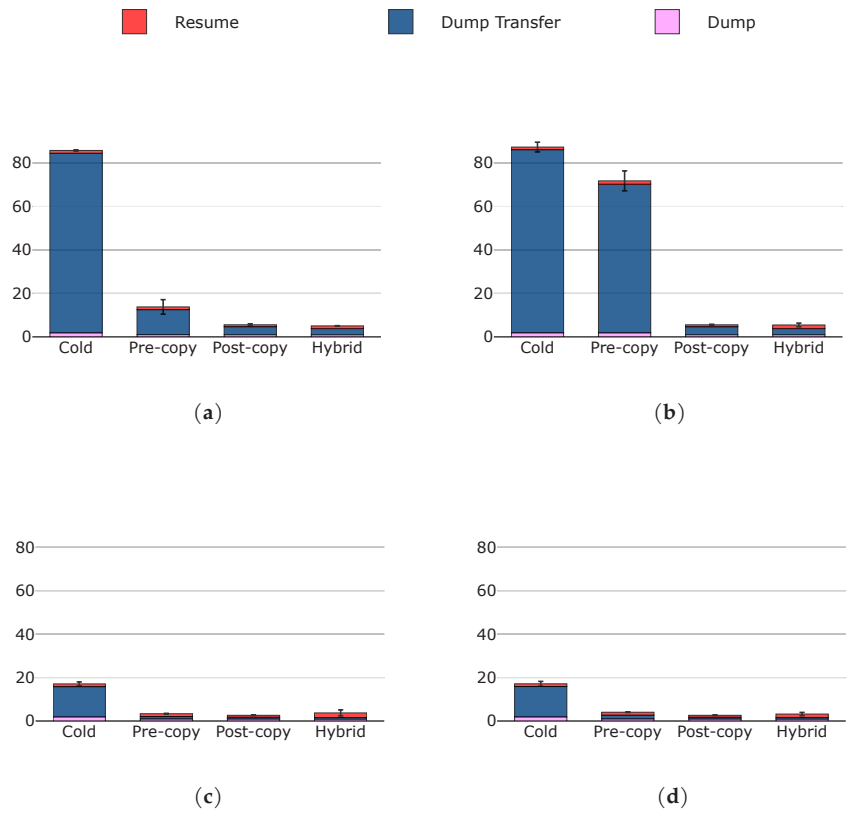


Figure 4.8: Downtimes (s) with their components in evidence under: (**a**) configuration A; (**b**) configuration B; (**c**) configuration C; and (**d**) configuration D.

Downtimes

Figure 4.8 depicts the downtimes for the migration techniques under the four considered configurations. As shown, the downtime for any technique is given by the sequence of the following times: (i) dump time; (ii) dump transfer time; and (iii) resume time. Cold migration always presents the highest downtime. This even coincides with the total migration time and proves to be unacceptable for most applications, especially for the critical ones that may exist in a Fog environment.

Under configurations C and D, the other three migration techniques show similar performances in terms of downtime. This is because, under these conditions, few memory pages are modified during the pre-copy phase; therefore, the dump size (and hence the dump transfer time) in pre-copy migration is comparable to those in post-copy and hybrid migrations. Besides, none of the four techniques seems

to be affected in terms of downtime by an increase in the page dirtying rate from configuration C to D, as already noticed and commented with regard to the total migration times. However, under configurations A and B, pre-copy presents significantly higher downtimes than post-copy and hybrid migrations. More specifically, under A, the downtime for pre-copy migration is longer because, considering the lower throughput, the size of the runtime state prolongs the pre-dump transfer time, giving the service more time to modify pages than under C or D. Therefore, the dump size in pre-copy migration grows as it strongly depends on the number of dirty pages, and the downtime does the same. A higher page dirtying rate under configuration B further increases the number of dirty pages and thus lengthens the pre-copy migration downtime. This even tends to that of cold migration, with the total migration time that, in addition, noticeably exceeds that of cold migration. It is evident, instead, how downtimes for post-copy and hybrid migrations are not influenced by the conditions characterising configurations A and B. This result was expected since neither of these two techniques includes dirty pages in its dumps, as reported in Table 4.1.

We now analyse dump times and resume times in more depth. Both these times are not clear by looking at Figure 4.8; therefore, we illustrate them in Figures 4.9 and 4.10, respectively. In general, dump times only depend on the amount of data that need to be dumped, while resume times depend on the amount of data from which the container must be restored. By looking at Figure 4.9, it is possible to notice how average dump times are always equal or less than 1 s except from those of cold migration (under all configurations) and that of pre-copy migration under configuration B. In these situations, indeed, the amount of data to be dumped (i.e., the dump size) is significantly higher than in all the others: the dump in cold migration is the whole container state, while that in pre-copy migration is of considerable size because of the conditions characterising configuration B. The condition on the runtime state size under configuration A, instead, is not sufficient on its own to cause an increase in the dump time for pre-copy migration. We also highlight that, with the only exception of pre-copy migration from configuration A to B, dump times are not affected by an increase in the page dirtying rate. This is due to the fact that, as reported in Table 4.1 and discussed in the previous sections, only the state dump in pre-copy migration includes dirty pages, and the increase in the page dirtying rate from configuration C to D does not determine an increase in the dump size that is significant enough to prolong the dump time.

Post-copy migration presents the shortest resume times (see Figure 4.10), as it restores the container at destination by only applying a very limited dump size to the base container image. All the other techniques have longer resume times. Going into detail, the cold and pre-copy techniques show similar values except from that of pre-copy migration under configuration B, which is caused by a higher amount of data to be applied to the base service image. We would have expected similar

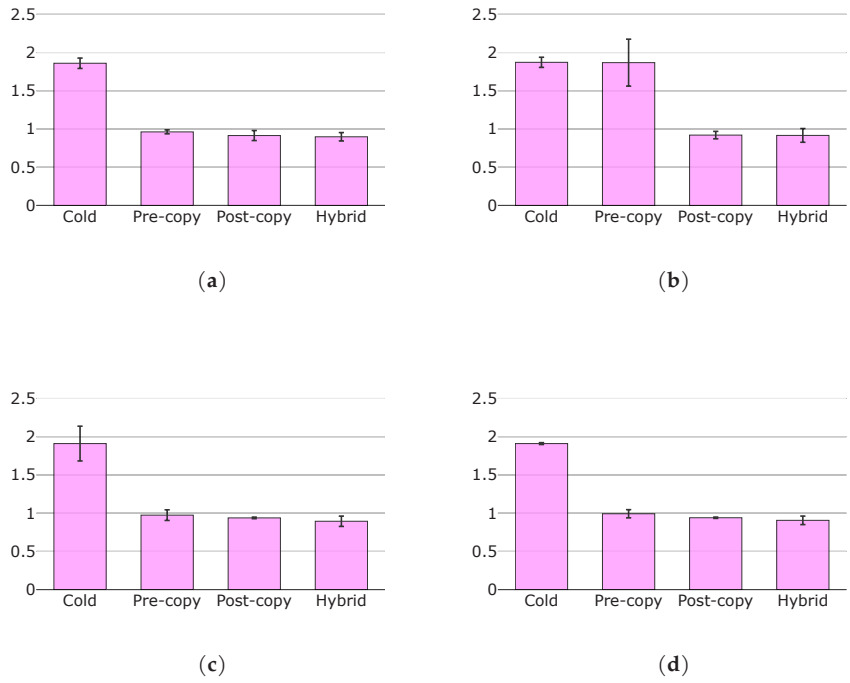


Figure 4.9: Dump times (s) under: (**a**) configuration A; (**b**) configuration B; (**c**) configuration C; and (**d**) configuration D.

values also for hybrid migration; however, results show that, in general, resume times for this technique are greater. A possible explanation of this outcome is that jointly applying the pre-dump and the dump to the base image in hybrid migration is computationally more intensive.

Transferred Data

In Figure 4.11, we illustrate the amounts of data transferred during the experiments. Firstly, it is easy to notice how most of the transferred state is: (i) the pre-dump for pre-copy and hybrid migrations; (ii) the dump for cold migration; and (iii) the faulted pages for post-copy migration. This is in line with what reported in Table 4.1. Secondly, the thinness of the dump layer, which is almost invisible, in the bar charts of post-copy migration shows another detail: **the execution state of a container is markedly negligible with respect to the size of memory pages.** The execution state size of the considered container is only 1.2 MB, indeed. Another consideration that can be made by looking at Figure 4.11 is that **a container is an environment that**

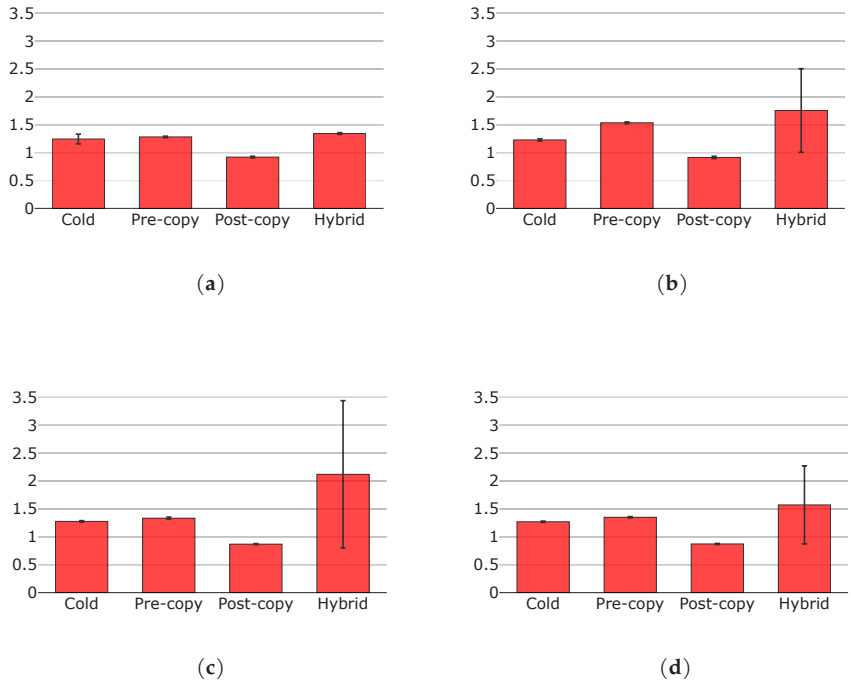


Figure 4.10: Resume times (s) under: (**a**) configuration A; (**b**) configuration B; (**c**) configuration C; and (**d**) configuration D.

occupies and updates more memory than that of the application running inside it. Indeed, as discussed in Section 4.3, the server running in the container allocates 75 MB of RAM, but more than 110 MB are transferred on average during container migration. Similarly, the dump size for pre-copy migration and the faulted pages size for hybrid migration are greater than what we were expecting, given the two considered page dirtying rates.

Let us now compare the migration techniques. Cold and post-copy migrations transfer the lowest amounts of data, irrespective of the specific configuration. This is due to the fact that they both transfer each memory page only once. Under configurations A, C, and D, pre-copy and hybrid migrations generate volumes of data that are comparable to those of cold and post-copy migrations, even though slightly higher. Going into detail, pre-copy and hybrid migrations perform at their best under configuration C, where there is the maximum difference between the throughput and the page dirtying rate (i.e., about three orders of magnitude) and, considering the available throughput, the runtime state size leads to a limited pre-copy

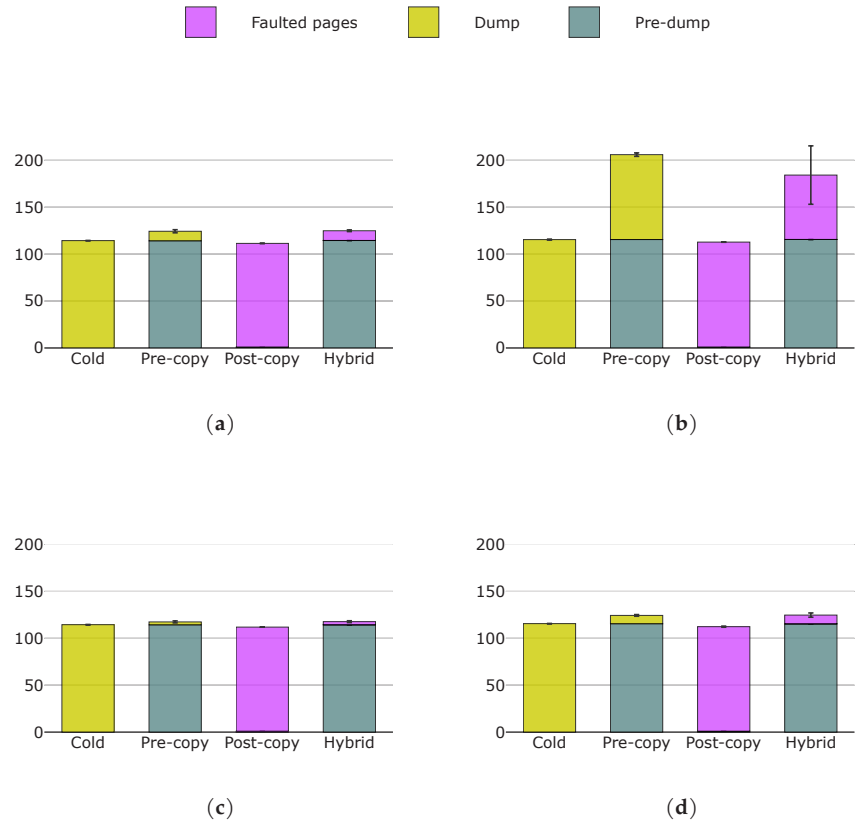


Figure 4.11: Amounts of transferred data (MB) with their components in evidence under: (**a**) configuration A; (**b**) configuration B; (**c**) configuration C; and (**d**) configuration D.

phase. An increase in the page dirtying rate from configuration C to D augments the amounts of transferred data only for pre-copy and hybrid migrations, but these increases are limited because the page dirtying rate is still of a different order of magnitude from the throughput. Instead, under B, a page dirtying rate of the same order of magnitude of the throughput causes these two migration techniques to modify a quantity of memory pages that is comparable to the whole state and thus transfer significantly greater volumes of data than those of cold and post-copy migrations. In particular, the dump size is what grows in pre-copy migration and the faulted pages size is what increases in hybrid migration, as these are the parts of the state containing dirty pages (see Table 4.1).

Lessons Learnt

For convenience, in Table 4.4 we summarise the most salient results relative to the total migration time, the downtime, and the amount of transferred data. As a closing remark, this work shows that **no migration technique is the very best** under all network and service conditions. However, based on the discussed results, we can conclude by stating that in general:

- **Cold** migration is to be avoided under all conditions because it always causes downtimes that are considerably higher than those of the other techniques;
- In situations where the throughput between nodes and the page dirtying rate are of different orders of magnitude, with the former higher than the latter, and the pre-copy phase does not have a prolonged duration (e.g., under configurations C and D), **pre-copy** migration may be the best option. Indeed, it has similar performances to those of post-copy and hybrid migrations, but it is not afflicted by the issues characterising these other two techniques (see Section 4.1);
- In situations where the page dirtying rate is of the same order of magnitude of the throughput and/or the pre-copy phase has a prolonged duration (e.g., under configurations A and B), pre-copy is to be avoided mainly because of rather long downtimes. **Post-copy** could be the best alternative, considering that it provides downtimes comparable to those of hybrid migration but performs better in terms of total migration time and amount of transferred data. It is worth noting, though, that post-copy presents a couple of non-negligible issues, which are explained in Section 4.1.

Table 4.4: Summary of the evaluated migration techniques.

Technique	Total migration time	Downtime	Transferred data
Cold	Always the lowest.	Always the highest. Coincides with the total migration time.	Always the least. Comparable to those of post-copy.
Pre-copy	Comparable or lower than that of post-copy when page dirtying rate and throughput are of different orders of magnitude, with page dirtying rate lower than throughput. Higher than that of post-copy when page dirtying rate is of the same order of magnitude of throughput.	Higher than those of post-copy and hybrid when page dirtying rate is of the same order of magnitude of throughput and/or pre-copy phase has a prolonged duration.	Much more than cold or post-copy when page dirtying rate is of the same order of magnitude of throughput.
Post-copy	Higher than that of cold, especially with a very high RTT between nodes. Comparable or higher than that of pre-copy when page dirtying rate and throughput are of different orders of magnitude, with page dirtying rate lower than throughput. Lower than that of pre-copy when page dirtying rate is of the same order of magnitude of throughput.	Always low and comparable to that of hybrid.	Always the least. Comparable to those of cold.
Hybrid	Always the highest.	Always low and comparable to that of post-copy.	Much more than cold or post-copy when page dirtying rate is of the same order of magnitude of throughput.

Chapter 5

Design and evaluation of a Companion Fog Platform

In this chapter, we propose a platform that provides the necessary functionalities to support device mobility in FC environments. This platform leverages containers as hosting environment and implements both stateless and stateful container migrations as mechanisms to let the Fog service follow a mobile (user) device. Note that our platform is generic enough to implement stateful container migration according to all the four techniques that are discussed in Chapter 4. We refer to such a platform as Companion Fog Platform (CFP), in line with the concept of CFC (see Section 3.1).

The rest of the chapter is organised as follows. Section 5.1 presents a model for CFC, from which we derive the insights that are necessary for platform design (see Section 5.2). Then, Section 5.3 reports the current platform implementation as well as a work-in-progress implementation based on OpenStack. Finally, Section 5.4 discusses the experiments that we carried out to validate the platform. The obtained results are promising, as our CFP significantly improves performances of standard FC (i.e., FC without service migration) in the presence of mobile devices.

5.1 Companion Fog Computing model

We now provide a semi-formal model for CFC, describing it as the combination of a distributed application and a CFP executing it. Modelling CFC helps us in the next design phase of our CFP. For convenience, Table 5.1 alphabetically reports the notation used in the model.

Application

For the sake of simplicity and without loss of generality, we assume a single instance of distributed application to be executed by the platform. In this paper, we model the

Table 5.1: Notation used in the CFC model.

Symbol	Description	Symbol	Description
AP	The set of the access points	h_i	The hardware requirement i of fs
ap_k	Access point k	$H_j(t)$	The set of availability status at instant t of the hardware resources on fn_j
$AP(t)$	The set of access points through which m can connect to the network at instant t	$h_{jz}(t)$	The availability status at instant t of the hardware resource z on fn_j
$ap(t)$	The actual ap_k through which m connects to the network at instant t	$H_m(t)$	The set of availability status at instant t of the hardware resources on m
$\overline{D}_k$	The set of distances $\overline{d}_{kj}$ between ap_k and each of the $fn_j \in FN_k$	$h_{mz}(t)$	The availability status at instant t of the hardware resource z on m
$\overline{d}_{kj}$	The average distance between ap_k and an $fn_j \in FN_k$ calculated over N samples	id_j	The unique identifier of fn_j
d_{max}	The upper limit on the average distance between fs and ma calculated over the whole application execution time	id_k	The unique identifier of ap_k
$\overline{d}_{mn}$	The average distance between m and $n(t)$ calculated over N samples	m	The mobile IoT device
δ_{mk}	The estimated distance between m and ap_k	ma	The mobile application component, which is lightweight and always runs on m
fa	The Fog application component, which is executed within a container	$n(t)$	The node hosting fs at instant t
FN	The set of the Fog Nodes	ξ	Indicates whether or not m can run fs when no other solution is possible
FN_k	The set of Fog Nodes that are in topological proximity to ap_k	o_{max}	The maximum percentage of the whole application execution time in which the distance between fs and ma may exceed d_{max}
fn_j	Fog Node j	π	The data protection level required by fs
fs	The Fog service, which consists in a container with fa inside. fs is the companion of ma	π_j	The data protection level provided by fn_j
H	The set of fs hardware requirements	$s_j(t)$	The status of fn_j at instant t
		σ	The nature of fs, whether stateful or stateless

application as a couple <ma, fa>: the application component ma always runs on the mobile IoT device m; on the other hand, fa is deployed in the Fog within a container according to the Infrastructure as a Service (IaaS) model. Hereafter, this container hosting fa will be referred to as fs, representing the actual Fog service behaving as

the companion of ma. It is worth noting that a third application component might reside at a higher layer of the Fog hierarchy (e.g., in the Cloud), but this is out of the scope of this thesis.

The Fog service fs is modelled as a 6-tuple $<H, d_{max}, o_{max}, n(t), \pi, \sigma>$ where:

- $H = \{h_i \mid h_i$ is the hardware requirement i of $fs\}$. h_i may be for instance the minimum amount of RAM or the number of virtual CPUs required;
- d_{max} is the upper limit on the average distance between fs and ma calculated over the whole application execution time. This parameter can be expressed for example as a RTT or a one-way latency;
- o_{max} represents the maximum percentage of the whole application execution time in which the distance between fs and ma may exceed d_{max};
- $n(t)$ is the node that hosts fs at instant t. It is worth noting that $n(t)$ may be also m itself, when feasible (see flag ξ). $n(t)$ and m may coincide in those cases (e.g., m is moving to a dead zone) in which no other solution is possible. However, fs should run on m only for short periods of time in order not to drain the battery of m and impair user experience;
- π is a natural number representing the data protection level required by fs. The higher π, the greater the data sensitivity and criticality, as in (UC Berkeley, 2012);
- σ indicates the nature of fs, namely whether it is a stateful or stateless service.

Companion Fog Platform

The CFP is modelled as a 3-tuple $<FN, AP, m>$.

$FN = \{fn_j\}$ is the set of Fog Nodes. Each fn_j is a 4-tuple $<id_j, H_j(t), s_j(t), \pi_j>$ where:

- id_j is the unique identifier of fn_j;
- $H_j(t) = \{h_{jz}(t) \mid h_{jz}(t)$ is the availability status at instant t of the hardware resource z on $fn_j\}$. $h_{jz}(t)$ may be for example the left amount of RAM or the number of unallocated virtual CPUs at instant t;
- $s_j(t)$ is the status of fn_j at instant t. It may be 1 to represent the "up" state or 0 to represent the "down" state;
- π_j is a natural number representing the data protection level provided by fn_j. The higher the provided data protection level, the bigger the minimum set of implemented security and privacy measures, as in (UC Berkeley, 2013).

$AP = \{ap_k\}$ is the set of access points, namely those nodes through which m can connect to the network. More precisely, an access point might be defined as the first node encountered along the communication path from m to $n(t)$, i.e., a cellular base station or a Wi-Fi access point. Each ap_k is a 4-tuple $<id_k, FN_k, \overline{D}_k, \delta_{mk}>$ where:

- id_k is the unique identifier of ap_k;
- $FN_k = \{fn_j \mid fn_j$ is a Fog Node in topological proximity to $ap_k\}$. It is worth noting that a specific fn_j may belong to the FN_k of one or more ap_k;
- $\overline{D}_k = \{\overline{d}_{kj} \mid \overline{d}_{kj}$ is the average distance between ap_k and an $fn_j \in FN_k$ calculated over N samples$\}$. A $\overline{d}_{kj}$ may be expressed for example as a RTT or a one-way latency. Let us suppose that m connects to the network through ap_k. Then, since the path from m to ap_k is always the same, $\overline{d}_{kj}$ is the only difference in the distances between m and each of the $fn_j \in FN_k$;
- δ_{mk} is the estimated distance between m and ap_k. This distance exclusively depends on the type of ap_k (i.e., cellular base station, Wi-Fi access point).

m is the mobile IoT device. It is a 5-tuple $<AP(t), ap(t), \overline{d}_{mn}, H_m(t), \xi>$ where:

- $AP(t) = \{ap_k \mid m$ may connect to the network through ap_k at instant $t\}$;
- $ap(t) \in AP(t)$ is the actual ap_k through which m connects to the network at instant t;
- $\overline{d}_{mn}$ is the average distance between m and $n(t)$ calculated over N samples. This information is not relevant when $n(t) = m$;
- $H_m(t) = \{h_{mz}(t) \mid h_{mz}(t)$ is the availability status at instant t of the hardware resource z on $m\}$. $h_{mz}(t)$ may be for example the number of unallocated virtual CPUs, the left amount of RAM, or the battery level at instant t;
- ξ is a Boolean that indicates whether (i.e., 1) or not (i.e., 0) m may run fs when no other solution is possible. m may be unable to do this due to hardware limitations (expressed by $H_m(t)$) or software incompatibility. For instance, Android kernel does not have namespaces support (LLC, 2018), which is an indispensable feature to run containers.

For the sake of readability, Figure 5.1 summarises the CFP model by showing the association of each parameter with the entities in the Fog hierarchy.

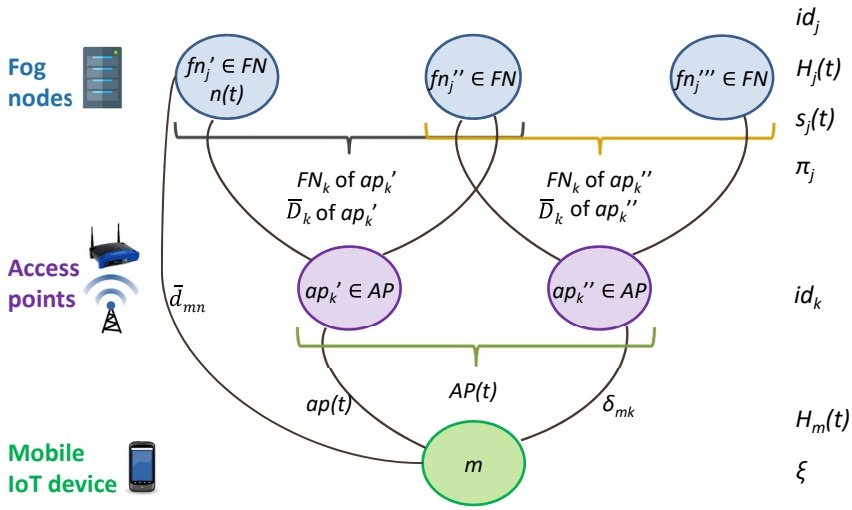


Figure 5.1: Graphical representation of the CFP model.

5.2 Platform design

Based on the CFC model, in what follows we discuss the design of our platform. Specifically, we present the derived reference architecture, then we propose a migration policy, and finally we describe the migration procedure (i.e., the execution of the migration policy by the reference architecture).

Reference architecture

Figure 5.2 depicts the reference architecture of the proposed CFP, while Table 5.2 alphabetically reports the acronyms for the architecture components. The Cloud is part of it and plays the role of an orchestrator. Nevertheless, it is not present in Section 5.1 since we do not need to further specify the Cloud in the CFC model.

For the sake of readability, modules are coloured differently based on their deployment: green modules are deployed on the mobile IoT device; blue components reside on the FNs, while azure modules are in the Cloud. Brown modules, instead, are the only ones that do not belong to the proposed platform, as they are provided by the application developer. Going into more detail, the **Mobile Application** is executed directly on the mobile IoT device, while the **Fog Application** runs within a container according to the IaaS model. In Figure 5.2, we want to highlight the fact that a FN may potentially host more than one container. However, as we mentioned in Section 5.1, in this work we focus on a single container/Fog service.

All the FNs and (potentially) the mobile IoT device embed a **Containerisation**

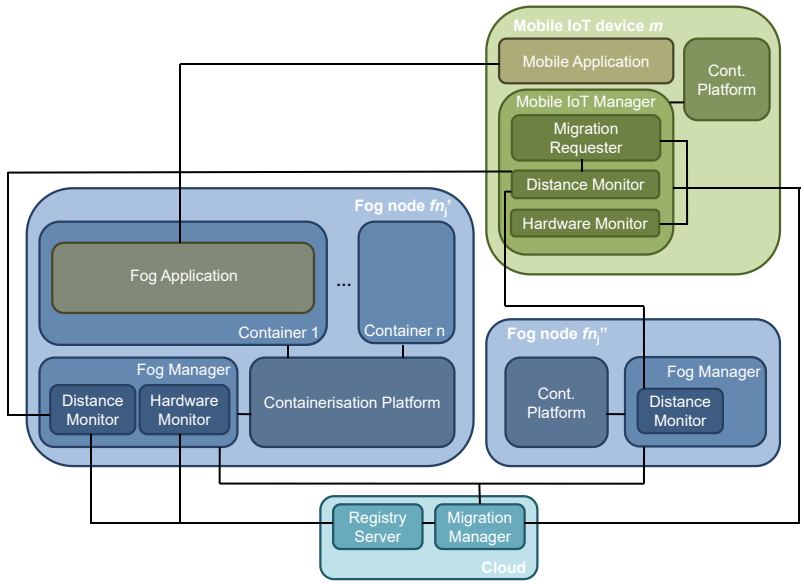


Figure 5.2: The proposed CFP reference architecture.

Platform (CP) for managing containers. Another fundamental module within each FN is the **Fog Manager** (FM), which comprises two submodules. The first is the **Distance Monitor** (DM), which interacts with the DM on the IoT device for the measurement of the topological distance between the IoT device and the FN. Furthermore, the DM in fn_j periodically calculates the distance $\overline{d}_{kj}$ (e.g., as an arithmetic mean) to each of the access points ap_k such that $fn_j \in FN_k$ of ap_k and reports

Table 5.2: Acronyms for the architecture components.

Acronym	**Full Name**
CP	Containerisation Platform
DM	Distance Monitor
FM	Fog Manager
HM	Hardware Monitor
MIoTM	Mobile IoT Manager
MM	Migration Manager
MR	Migration Requester
RS	Registry Server

the obtained values to the **Registry Server** (RS) in the Cloud. The period T_2 of such measurements is rather long: being in general both the FNs and the access points fixed, these distances rarely change over time. The second submodule in the FM is the **Hardware Monitor** (HM), which periodically monitors the availability status of the hardware resources (e.g., CPU, RAM) in the FN and reports them to the RS. As this period T_3 is shorter than T_2, each update from the HM is also considered by the RS as an "alive" message for the estimation of $s_j(t)$. An HM is also present on the mobile IoT device. However, this module does not periodically report its measurements to the RS; instead, it communicates with the **Migration Manager** (MM) in the Cloud upon request of the latter. The reason for this is that an estimation of the available hardware resources (e.g., CPU, RAM, battery level) on the mobile IoT device is rarely needed and required only in the very last steps of the proposed migration policy (which is presented later in this section).

The RS manages the data from each FM and updates a repository accordingly. The MM is the module in the Cloud that receives and satisfies the migration requests coming from the **Migration Requester** (MR). The latter is deployed on the mobile IoT device and, together with the DM and the HM, constitutes the **Mobile IoT Manager** (MIoTM). In order to satisfy a migration request, the MM interacts with the RS to retrieve the necessary information and executes the migration policy. Moreover, it interacts with the CPs (through the FM or the MIoTM) in order to perform the actual container migration, and, once the migration ends, it returns the identifier of the new $n(t)$ back to the MR.

We clarify that, for the sake of readability, Figure 5.2 depicts in fn_j'' only the modules that differ from those in fn_j'. Specifically, as detailed later in this section (i.e., migration procedure), the DM on fn_j' is contacted by the DM on the mobile IoT device to decide when to migrate, while the DM on fn_j'' is interrogated to decide where to migrate. Besides, the CPs on fn_j' and fn_j'' behave differently as the former is the source of the migration, while the latter is the destination.

Migration policy

In what follows, taking into consideration the model from Section 5.1, we propose a reactive migration policy for our CFP. As discussed in Section 3.4, a migration policy is a set of steps that allow to make the *When-to-Migrate* and the *Where-to-Migrate* decisions. We highlight that the proposed model and reference architecture are generic enough to allow the definition and employment of any other migration policy. To begin with, we provide the following definition, which is then recalled within the migration policy.

Definition of compatibility between fs and fn_j - let us set $\overline{d}_{mj}$ as the average distance between m and fn_j calculated over N samples. The Fog service fs and fn_j are

compatible with one another if and only if all the following conditions are met:

- $H \subseteq H_j(t)$;
- $\pi \leq \pi_j$;
- $d_{max} \geq \overline{d}_{mj}$.

A clarification is due. If $fn_j \in FN_k$ of $ap(t)$, then $\overline{d}_{mj}$ is the actual average distance. On the other hand, if $fn_j \in FN_k$ of an $ap_k \neq ap(t)$, then $\overline{d}_{mj}$ is estimated as $\overline{d}_{mj} = \delta_{mk} + \overline{d}_{kj}$. It is useless, indeed, to actually measure $\overline{d}_{mj}$ in the second case, as m accesses the network through $ap(t)$, while fn_j is a potential FN for m only when this connects to the network through $ap_k \neq ap(t)$.

Algorithm 1 reports the pseudo-code for the migration policy. Let us assume that $n(t) = fn'_j$. Every T seconds, the policy monitors $d_{mn}(t)$, namely the current distance between m and $n(t)$. The value of $\overline{d}_{mn}$ is then updated over the N most recent values of $d_{mn}(t)$. When $\overline{d}_{mn} > d_{max}$, it is time to migrate fs, since the third condition of compatibility is no more respected. As a first step, the migration target fn''_j is searched within the FN_k of $ap(t)$ (i.e., line 6 in Algorithm 1), as this FN_k contains the FNs that are currently the topologically closest to m. As described in Algorithm 2, such fn''_j must be compatible with fs and have $s_j(t) = 1$. If there exist more than one fn_j with such characteristics, that with the minimum $\overline{d}_{kj}$ is chosen. fs is migrated to that FN, and $n(t)$ is updated accordingly.

If there is not an fn_j that satisfies the requirements in the previous step, the second step of the migration policy starts. The migration target fn''_j is looked for within each FN_k of all the other $ap_k \in AP(t)$ with $ap_k \neq ap(t)$ (i.e., lines 11 and 12 in Algorithm 1). The conditions that such fn''_j must meet are the same as in step 1 with the only difference that the fn_j with the minimum $\overline{d}_{mj} = \delta_{mk} + \overline{d}_{kj}$ is chosen among many compatible nodes. This is due to the fact that, since the scope in this step is $AP(t)$, two different fn_j may belong to the FN_k of two different ap_k. Therefore, δ_{mk} is necessary in order to distinguish among the possibly different types of ap_k involved (i.e., cellular base station, Wi-Fi access point). It is worth noting that if fn''_j is found in this step, two are the possible results, but both of them cause a forced handoff (i.e., line 16 in Algorithm 1) since fn''_j is topologically close to an $ap_k \neq ap(t)$. If $fn''_j = fn'_j$, the instruction at line 15 in Algorithm 1 does not cause any migration, nor any update of the value of $n(t)$. Otherwise, fs is migrated from fn'_j to fn''_j, and the value of $n(t)$ is updated accordingly.

If both the first and the second steps of the policy fail to identify fn''_j and $\xi = 1$, fs is migrated to m as an extreme fallback solution. Running fs on m may spoil user experience and heavily consumes the battery of m. Therefore, fs migration is attempted every T_1 seconds (with $T_1 < T$) by starting from the first step of the migration policy (i.e., line 6 in Algorithm 1). Instruction at line 20 would result in no action carried out.

Algorithm 1: Migration policy.

```
1  while True do
2      shortPeriod = False;
3      calculate d_mn(t);
4      update d̄_mn over the last N samples;
5      if d̄_mn > d_max then
6          List_potentialTargets = FindPotentialTargets(ap(t));
7          shortPeriod = False;
8          if List_potentialTargets not empty then
9              n(t) = argmin_{fn_j ∈ List_potentialTargets} d̄_kj;
10         else
11             foreach ap_k ∈ AP(t) & ap_k != ap(t) do
12                 List_potentialTargets = FindPotentialTargets(ap_k);
13             end
14             if List_potentialTargets not empty then
15                 n(t) = argmin_{fn_j ∈ List_potentialTargets} d̄_mj;
16                 perform handoff to the new ap_k != ap(t);
17             else
18                 shortPeriod = True;
19                 if ξ = 1 then
20                     n(t) = m;
21     if shortPeriod = False then
22         wait T seconds;
23     else
24         wait T_1 seconds;
25         go to 6;
26 end
```

Instead, if the first and the second steps of the policy are not successful and $\xi = 0$, fs keeps on running on fn'_j. Also in this case, fs migration is attempted with period T_1 by starting from line 6.

Migration procedure

We now provide a step-by-step description of the migration procedure, which is the execution of the migration policy by the reference architecture. For the sake of conciseness, we assume that the migration target is found at the end of the first step of the migration policy (i.e., line 9 in Algorithm 1). Therefore, we report only the set of actions up to that point (see Figure 5.3). Colours in Figure 5.3 match those in Figure 5.2. Hence, green components are present on the mobile IoT device; blue ones run on the FNs, while azure modules are deployed in the Cloud.

Algorithm 2: Find potential targets.

Input: ap_k
Output: Inserts the potential migration targets, given ap_k, in $List_{potentialTargets}$

```
1 Function FINDPOTENTIALTARGETS(ap_k):
2     foreach fn_j ∈ FN_k of ap_k do
3         if fn_j is compatible with fs & s_j(t) = 1 then
4             insert fn_j in List_potentialTargets;
5         else
6             discard fn_j;
7     end
8 End Function
```

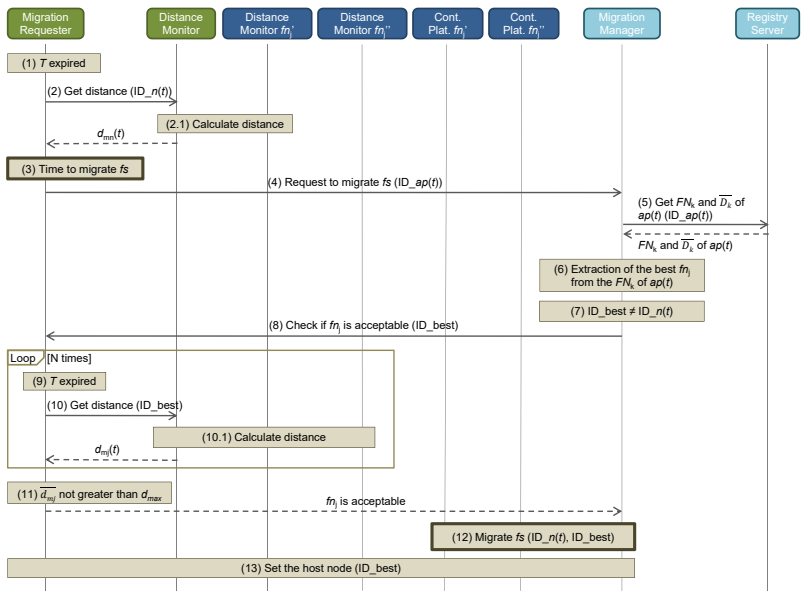


Figure 5.3: The procedure relative to the first step of the migration policy.

The procedure begins with the expiration of period T: it is time to measure $d_{mn}(t)$. Therefore, in step 2, the MR asks the DM on the mobile IoT device for this value. The DM on the mobile IoT device interacts with the DM on $n(t) = fn'_j$ in order to satisfy the request. There exist several ways to implement step 2.1. For example, if $d_{mn}(t)$ is expressed as a RTT, then it might be measured through a ping. Once the MR obtains $d_{mn}(t)$, in step 3, it updates $\overline{d}_{mn}$ over the N most recent values

(with $d_{mn}(t)$ being the most recent of all). The MR then compares $\overline{d}_{mn}$ with d_{max} and finds out that the third condition of compatibility (refer to the migration policy) is no more respected. Step 3 is highlighted because it is the actual moment in which the MR decides that it is time to migrate fs (i.e., the *When-to-Migrate* decision). As a result, in step 4, the MR contacts the MM with a request whose argument is the unique identifier of $ap(t)$. In step 5, the MM provides this identifier to the RS in order to retrieve the up-to-date FN_k and $\overline{D}_k$ of $ap(t)$. Step 6 is the cornerstone of the presented procedure. In this step, indeed, the MM scans the FN_k of $ap(t)$ in order to identify the migration target. To begin with, any fn_j that does not satisfy the first two compatibility conditions in the migration policy or that has $s_j(t) =$ "down" is immediately discarded. The MM: (i) compares the remaining FNs exclusively according to the distances $\overline{d}_{kj}$ in $\overline{D}_k$; (ii) selects fn''_j, which is the node with the minimum $\overline{d}_{kj}$ (i.e., the topologically closest); and (iii) discards all the others.

This decision solely based on $\overline{d}_{kj}$ presents two strengths. Firstly, the mobile IoT device does not have to calculate the distances $\overline{d}_{mj}$ to all the FNs in the FN_k of $ap(t)$, thus saving its resources. Secondly and most importantly, the distances $\overline{d}_{kj}$ are ready to use when needed as they are periodically provided by the DMs in the FNs, and this reduces the overall duration of the migration procedure.

At the end of step 6, a potential migration target is found in fn''_j. It is only a potential target because the MM still needs to be sure that fn''_j satisfies the third condition of compatibility from the migration policy. To this purpose, the MM first checks whether $fn''_j = n(t)$, in step 7. If this happens, the MM can start performing the second step of the migration policy, since it already knows that $n(t)$ does not respect the third condition of compatibility. However, in the considered scenario, $fn''_j \neq n(t)$. Therefore, in step 8, the MM proceeds with the first step of the migration policy and asks the MR on the mobile IoT device to verify the aforementioned condition. In the steps from 9 to 10, the current distance $d_{mj}(t)$ from m to fn''_j is measured N times with period T. Next, in step 11, the MR calculates $\overline{d}_{mj}$ (e.g., as an arithmetic mean) from the N samples of $d_{mj}(t)$ and finds out that $\overline{d}_{mj}$ is not greater than d_{max}. Therefore, the MM is notified by the MR that fn''_j satisfies the third condition of compatibility. As a result, in step 12, the MM chooses fn''_j as the migration target (i.e., the *Where-to-Migrate* decision) and interacts with the CPs on fn'_j and fn''_j in order to perform the actual migration of fs. For simplicity, Figure 5.3 depicts direct communications between the MM and the CPs; however, such interactions actually happen through the FMs in fn'_j and fn''_j. Besides, it is worth highlighting the fact that step 12 may result either in a stateful or stateless migration, depending on the value of parameter σ (see Section 5.1). Finally, once the migration ends, the procedure concludes in step 13, in which the MM locally sets $n(t)$ to fn''_j and communicates this information to the MR on the mobile IoT device.

5.3 Platform implementation

This section describes the CFP implementation. Specifically, there exist two implementations of our platform: one is a standalone prototypical implementation; the other is integrated in OpenStack and is currently a work in progress.

The first implementation, which is the one employed for platform validation in Section 5.4, is written in Python 2.7 and leverages *RPyC*[47] for the communication among architecture components. RPyC is a Python library that allows to implement client-server applications according to the Remote Procedure Call (RPC) protocol. With RPC, a program can require the execution of a procedure to a program running on another machine. Besides, RPyC allows symmetrical calls, which means that both client and server can serve requests. This enables the server to invoke callback procedures on the client side. With regard to containerisation, we use runC as container runtime (see Section 4.2). To statefully migrate runC containers, we use: (i) CRIU as the tool for checkpointing the container on the source node and restoring it on the destination one; (ii) rsync as state transfer mechanism (see Chapter 4). Our platform implementation allows containers to be migrated both statelessly and statefully (according to all the cold and live techniques described in Chapter 4).

The second work-in-progress implementation is based on OpenStack[48], which is one of the most prominent open-source Cloud platforms. The OpenStack project initiated in 2010 as a joint initiative of Rackspace Hosting and the NASA and is currently managed by the OpenStack Foundation, a non-profit corporate entity established in September 2012. More than 500 companies have joined the project since then, and the OpenStack development community currently counts more than 82,000 members from 187 countries around the world (OpenStack, 2013). As shown in Section 2.6, several initiatives exist that integrate FC capabilities in OpenStack. Among these stands *IoTronic* (Longo et al., 2017), an open-source service that extends OpenStack for the remote management of IoT devices (whether embedded systems, single-board computers, or even mobiles). IoTronic has been recently recognised by the OpenStack community at large and more specifically promoted to the shortlist[49] of projects that the OpenStack *Edge Computing Group* is agreeing upon to let the OpenStack ecosystem fully support FC/EC use cases in the near future. As such, IoTronic, albeit currently an unofficial project, is expected to be gradually included in OpenStack. With regard to its architecture, the IoTronic service interacts with *Lightning-rod*, its counterpart that resides on each IoT device. Lightning-rod runs under the device-native (typically SDK-enabled) environment available to developers and interacts with the (subset of) OS tools and services present on the device as well as with the sensing and actuation resources. A WebSocket-based

[47] See `https://rpyc.readthedocs.io/en/latest/`. Last accessed: 30 October 2019.

[48] See `https://www.openstack.org/`. Last accessed: 22 February 2019.

[49] See `https://wiki.openstack.org/wiki/Edge_Computing_Group#Related_OSF_Projects`. Last accessed: 22 February 2019.

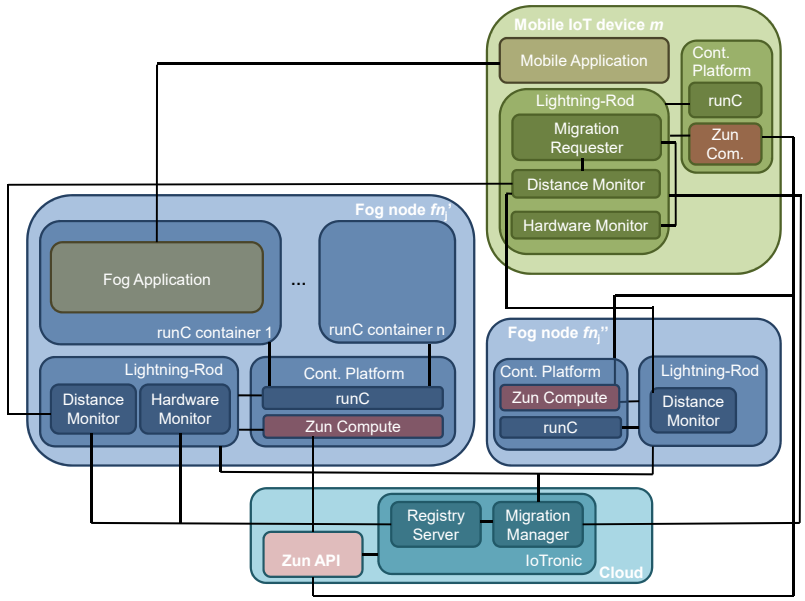


Figure 5.4: Architecture of the CFP OpenStack-based implementation.

tunnelling and Web Application Messaging Protocol (*WAMP*)-based messaging between Lightning-rod and IoTronic allow the management of IoT resources even when these are behind a NAT or a strict firewall. Specifically, WAMP is a sub-protocol of WebSocket that specifies a communication semantic for messages sent over WebSocket and provides both Publish/Subscribe and (simple or routed) RPC mechanisms. In IoTronic, WebSocket is also exploited to implement tunnelling for virtual networking overlays among IoT nodes.

In Figure 5.4, we show the architecture of the CFP OpenStack-based implementation, highlighting how its components map to those of the reference architecture. Going into detail, the Cloud components of the reference architecture (i.e., the RS and the MM) are being implemented as agents in the IoTronic service. The RS can exploit the IoTronic internal database to store metadata associated to the FNs, such as the status of hardware resources. On the mobile IoT device, the MIoTM is being implemented as a module within Lightning-Rod. It is worth noting that in order to match up with the reference architecture, this implementation needs to extend IoTronic with two novelties. Firstly, it has to introduce an intermediate layer composed of FNs in between the (mobile) IoT devices and the Cloud. More specifically, the FM in the reference architecture is being implemented as a specialised instance of Lightning-Rod, thus giving birth to a version of Lightning-Rod that is aimed at the

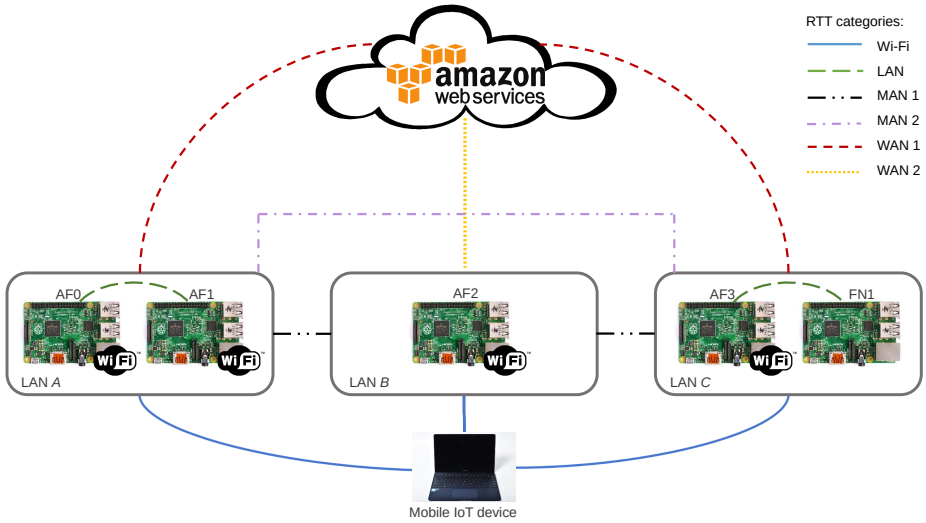


Figure 5.5: System overview.

Fog layer. Secondly, this CFP implementation is going to enhance IoTronic by embedding a containerisation platform in each FN and, when feasible, in mobile IoT devices. This is necessary to manage containers, from their deployment to their instantiation and migration. Specifically, the containerisation platform will comprise two modules: (i) runC, as lightweight container runtime; (ii) *Zun Compute*, which is part of *Zun*[50], namely the OpenStack container service that allows to manage the placement and life cycle of containers. The features of Zun are exposed through the *Zun* (*REST*) *API* that resides in the Cloud.

5.4 Platform validation

This section reports the set of experiments that we carried out in order to validate the proposed CFP. Specifically, we first detail the experiments setup and the tools employed; then, we analyse the obtained results.

Experiments setup

Figure 5.5 depicts the system that we configured for the experiments, in accordance with that described in Section 3.2 for the smart assistant use case. An *Asus ZenBook* notebook with Ubuntu 18.04 and Linux kernel 4.15.0-36 represents the mobile IoT device. The five FNs are *Raspberries Pi 3 Model B* with Debian 9.5 and Linux kernel

[50] See `https://wiki.openstack.org/wiki/Zun`. Last accessed: 30 October 2019.

4.14.73-v8+. As shown, four of the five FNs (i.e., those whose name starts with AF) also behave as Wi-Fi access points. To this purpose, we installed and configured *hostapd* and *dnsmasq* on each of them. We deployed all the Raspberries and the Asus notebook in an office within the Department of Information Engineering at the University of Pisa. The Cloud server, instead, is an *Amazon Web Services* (*AWS*) *Virtual Machine* (*VM*) with Amazon Linux and Linux kernel 4.14.62-65 that we deployed in Ireland (eu-west-1c).

As discussed, the FNs are all physically present in the same room and interconnected with one another through Ethernet. Therefore, in order to match up with the system reported in Figure 5.5, we had to emulate some of the RTTs (i.e., those for the MAN 1, MAN 2, and WAN 2 categories in figure) thus to logically arrange the FNs in three different LANs. LAN *A* comprises two FNs. LAN *B* contains only AF2 as FN, which, in the considered use case, connects to the Internet through 4G/LTE. Finally, LAN *C* includes two FNs of which only one (i.e., AF3) is also an access point. In order to artificially set RTTs where needed, we exploited Linux Traffic Control Network Emulator (tc-netem). We imposed these RTTs to real values that are reported in Table 5.3 together with those values that were not imposed, as already present in the system (i.e., those for the Wi-Fi, LAN, and WAN 1 categories). We obtained all these values as follows. For each RTT category reported in Figure 5.5, we considered two nodes in a real-world deployment[51] and measured RTTs between those two nodes over 10 runs, with 20 measurements per run. We then considered the average RTT per run to obtain the presented values.

In this context, the Fog service is a runC container that hosts a server written in C and listening to client requests on UDP. As in the smart assistant use case, d_{max} (see Section 5.1) is set to 20 ms. Besides, o_{max} is set to 5% of the overall experiment run time, which is 20 *min* as described below. Given the above Fog service, we carried out experiments considering four different patterns: (i) **Fixed Cloud**, where the

Table 5.3: Measured RTTs (95% Confidence Intervals).

RTT category (See Figure 5.5)	**Values** (ms)	**Set through *tc-netem***
Wi-Fi	7.902 ± 1.489	
LAN	0.745 ± 0.027	
MAN 1	122.952 ± 5.566	✓
MAN 2	18.843 ± 4.952	✓
WAN 1	43.448 ± 2.846	
WAN 2	152.398 ± 8.706	✓

[51] For instance, for MAN 1, we took into account a fixed device connected to the network of the University of Pisa through Ethernet and a smartphone connected to the Internet through 4G/LTE.

container runs in the Cloud throughout the experiment; (ii) **Fixed Fog**, where the container runs on AF0 throughout the experiment; (iii) **Stateful Migration**, where the container is statefully migrated, when necessary, from a FN to another; and (iv) **Stateless Migration**, where the container is statelessly migrated, when necessary, from a FN to another.

We emulated the IoT device mobility through automatic Wi-Fi handoffs. To this purpose, we employed the *os* Python module and the Linux command-line tool for controlling NetworkManager (i.e., *nmcli*). More specifically, handoffs occur at specific instants of time so that the mobile device is associated to each of the three LANs for 6 *min* 40 *sec*[52]. As a result, each experiment run is 20 *min* overall. With respect to patterns 3 and 4 (i.e., Stateful Migration and Stateless Migration), the handoff from AF0 to AF1 does not cause any container migration. On the contrary, migration to AF2 is necessary after the second handoff (i.e., that from AF1 to AF2). Finally, the third and last handoff (i.e., that from AF2 to AF3) determines a migration to FN1.

We exploited Linux Traffic Control Hierarchy Token Bucket (tc-htb) to limit the throughput from AF0 to AF2 and that from AF2 to FN1. This is necessary only for pattern 3, as the other patterns do not require any data transfer between FNs. Specifically, in order to choose the most appropriate value for the throughput, we considered AF0 and a smartphone connected to the 4G/LTE network, and we performed 10 throughput measurements using the *iperf3* tool, sending 50 MB each time. The final result is a 11.342 ± 2.306 Mbps throughput, with a 95% confidence level. This is in line with (Akamai, 2016) that estimates the average speeds for mobile connections in Italy to be 11 Mbps.

As described in Section 5.3, our CFP implementation allows to statefully migrate containers according to both *cold* and *live* migration techniques. In Chapter 4, we explained how the choice of which technique to use mainly depends on the following factors: (i) available throughput between the source and the destination FNs; (ii) service page dirtying rate; and (iii) amount of data transmitted during the pre-copy phase. We have already detailed how we set the throughput between FNs in the validation system. Instead, to calculate the second and third of the above factors, we proceeded as follows. By instantiating the Fog service on AF0 and migrating it 10 times to AF2 according to the pre-copy technique with one pre-copy iteration, we first found that the pre-dump size (i.e., the checkpoint size during the pre-copy phase of the migration) is 36.70 ± 0.02 MB, with a 95% confidence level. With the purpose to reduce the volume of data transferred during the pre-copy phase, we made the following improvements, which were then replicated also for the actual platform validation. Firstly, prior to transmitting data, we deployed and checkpointed the Fog service on AF2. This strategy is realistic in a FC environment provided that the user's mobility pattern or at least the geographic area where

[52]While in LAN *A*, the mobile device is connected to access points AF0 and AF1 for the same time interval, i.e., 3 *min* 20 *sec*.

he/she moves are known to a certain extent. By doing this, it is possible to leverage the incremental data transfer performed by rsync so that, thanks to the presence of a checkpoint on the destination FN, only 7.254 ± 0.005 MB out of the total pre-dump size need to be copied. Secondly, we employed the rsync compression functionality in order to further reduce the amount of transferred data during the pre-copy phase to 1.017 ± 0.003 MB. We also calculated the Fog service page dirtying rate as the relationship between the *dump size* and the *pre-dump transfer time* (see Chapter 4), obtaining 0.517 ± 0.026 Mbps. As concluded in Chapter 4, **pre-copy migration** represents the best option under these conditions. Therefore, we exploited this stateful migration technique in our experiments for platform validation.

As a final remark, we performed five experiment runs for each of the four patterns. In each run, we monitored the RTT between the mobile IoT device and the current FN every 5 *sec* and stored the couples <*instant t*, *RTT value*> in .csv files for the next phase of results analysis and plotting. This was performed in Python through *Plotly*, an open-source graphing library for Python.

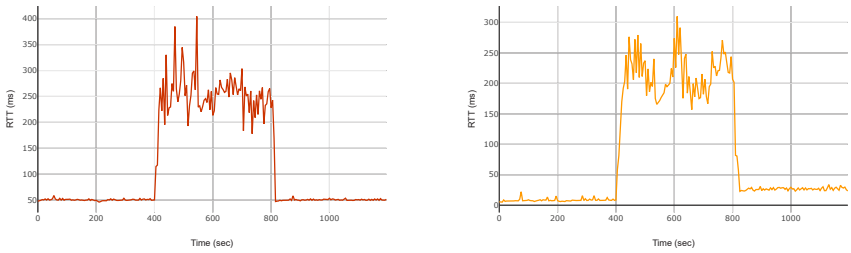
Results

We now examine the obtained results by first discussing how the RTT between the mobile device and the serving node varies over time in each of the four patterns. Then, we make further remarks to show how our CFP improves the overall performance with respect to both the Fixed Cloud and the Fixed Fog cases.

RTTs over time are reported in Figures 5.6 and 5.7. More specifically, Figure 5.6(**a**) reports RTT over the experiment run time in the Fixed Cloud pattern. In this case, RTT is always greater than d_{max}. Specifically, it reaches its peak (i.e., about 400 ms) when the mobile device is connected to LAN *B* through AF2, as Internet access occurs through 4G/LTE. Instead, when the mobile device is connected to LAN *A* or LAN *C*, RTT is lower but is still significantly above d_{max}.

FC may reduce RTT with respect to that towards the Cloud. As reported in Figure 5.6(**b**), when the mobile device is connected to LAN *A*, the RTT towards the serving FN (i.e., AF0) is much lower than d_{max}. However, standard FC does not perform appropriately in the presence of mobile devices. As shown, when the mobile device is connected to LAN *B*, RTT towards AF0 considerably increases, going beyond 300 ms at its peak. After the handoff to LAN *C*, RTT decreases but not enough to go below the required 20 ms.

Our CFP significantly improves FC performance by supporting device mobility. Figure 5.7(**a**) depicts how RTT varies over time when the Fog service is statefully migrated to follow the served mobile device. Given that the Fog service initially runs on AF0 and that the mobile device is connected to LAN *A*, the RTT towards AF0 is the same as that in the Fixed Fog pattern. When after 400 seconds a handoff to LAN *B* happens, RTT suddenly increases reaching 250 ms. However, differently



(**a**) (**b**)

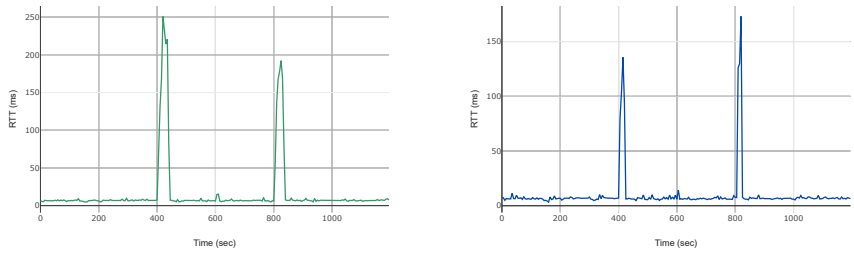
Figure 5.6: RTTs over time in the: (**a**) Fixed Cloud pattern; (**b**) Fixed Fog pattern.

from the Fixed Fog case, our CFP statefully migrates the Fog service from AF0 to AF2. Once migration is completed, the RTT towards the current FN (i.e., AF2) is again below d_{max}. A similar outcome also occurs when the mobile device moves from LAN *B* to LAN *C* and the Fog service is consequently migrated to FN1.

Performance further improves if the Fog service is statelessly migrated rather than statefully (see Figure 5.7(**b**)). Firstly, RTT values immediately after Wi-Fi handoffs are lower (i.e., around 150 ms) than those in the Stateful pattern. This is because, as discussed in Chapter 4, stateful container migration copies the runtime service state to the new node and as such causes a greater network overhead than that determined by a stateless migration, which instead is a restart from scratch of the container on the new node. For the same reason, the curves that follow handoff times are less wide in the Stateless pattern rather than in the Stateful one. This is due to the fact that copying the runtime state to the new node requires more time than simply re-instantiating the container statelessly.

Tables 5.4 and 5.5 summarise the most noteworthy aspects of the obtained results. In particular, Table 5.4 reports the average RTT in each pattern and also presents these values as a percentage of d_{max}. Table 5.5 instead lists the average outage times and shows them as a percentage of the experiment run time. We define outage time as the overall period of time in which the application experiences a RTT greater than d_{max}.

Performances in the Fixed Cloud and Fixed Fog patterns are far from acceptable. As shown, average RTTs in these two cases are respectively about six and four times d_{max}. Moreover, the percentage of the overall experiment run time in which the system is in outage considerably exceeds o_{max} (i.e., 5%) in both cases. In particular, in the Fixed Cloud pattern, outage time coincides with the experiment run time; for Fixed Fog, instead, outage time is lower but still more than 60% of the experiment run time.



(**a**) (**b**)

Figure 5.7: RTTs over time in the: (**a**) Stateful Migration pattern; and (**b**) Stateless Migration pattern.

On the contrary, our CFP leads to average RTTs that are below d_{max}. In particular, they are around 16 and 10 ms in the Stateful and Stateless patterns, respectively. Furthermore, outage times are limited to less than 5% of the experiment run time in the Stateful case and around 1.5% of the time in the Stateless one.

Table 5.4: Average RTTs (95% Confidence Intervals).

Pattern	**Values** (ms)	**% of** d_{max} (**20** ms)
Fixed Cloud	119.135	595.675
Fixed Fog	82.801 ± 5.119	414.006 ± 25.593
Stateful Migration	16.240 ± 1.821	81.199 ± 9.106
Stateless Migration	10.037 ± 0.775	50.186 ± 3.877

Table 5.5: Average outage times (95% Confidence Intervals).

Pattern	**Values** (s)	**% of experiment run time** (**1200** s)
Fixed Cloud	1200	100
Fixed Fog	765 ± 18.1	63.75 ± 1.508
Stateful Migration	53 ± 5.553	4.417 ± 0.463
Stateless Migration	18 ± 5.553	1.5 ± 0.463

Chapter 6

Simulation of device mobility and service migration in Fog Computing

Simulation may represent a time- and cost-effective way to evaluate service migration strategies in FC environments with mobile users. Several Fog simulators exist in literature (see Section 3.6), but, to the best of our knowledge, none of them models service migration. In this chapter, we propose **MobFogSim**, a simulator that extends iFogSim to model all the aspects related to device mobility and service migration in FC. Besides, we validate our simulator by replicating in it the testbed setup from Chapter 4 and by comparing simulation results of container migration with the results discussed in Chapter 4. MobFogSim is open source and available at `https://github.com/diogomg/MobFogSim`.

The chapter is structured as follows. Section 6.1 presents the model used in the simulator for connection handoff and service migration. Then, Sections 6.2 and 6.3 report the design and implementation details of MobFogSim, respectively. Next, Section 6.4 describes the experiments that we carried out to calibrate MobFogSim, namely to appropriately configure its input values for the replication in the simulator of the testbed conditions. Finally, in Section 6.5, we validate MobFogSim by analysing the simulation results against the testbed ones.

6.1 Migration and handoff model in MobFogSim

In this section, we present the model considered in MobFogSim for connection handoff and service migration in FC environments. Specifically, we identify eight events related to handoff and migration. We highlight that such events may occur in the order that is described next but also in other orders. Moreover, note that the following is a high-level description of these events; in Section 6.3, we describe the sequence of events occurring in a migration and handoff scenario, as per the implementation in MobFogSim. The eight events are as follows:

- **Event 1 (E1)** is when the Fog system decides that it is time to perform a migration (i.e., the When-to-Migrate decision). Many different approaches exist to make this decision, depending on the strategies being adopted. For instance, service migration may be triggered before the connection handoff starts (i.e., before E4), based on known information on user's current mobility (e.g., position, direction, speed) and/or based on known mobility patterns for categories of users (e.g., public transport buses) (Gonçalves et al., 2018). This is called proactive migration. Alternatively, migration can be triggered once connection handoff has already started, and the new access point of the user is known in a deterministic way. This is instead called reactive migration. Details on the When-to-Migrate approach that is currently implemented in MobFogSim can be found in Section 6.2. However, we highlight that developers can implement their own migration decisions in the simulator;

- **Event 2 (E2)** is the set of necessary procedures associated with preparing the service for migration (e.g., checking its size and page dirtying rate, checkpointing its state, monitoring network speed to estimate the migration time, etc.) and establishing a network connection between the source and the destination FN for migration. The way data are prepared for migration depends on the actual migration technique being used. Please, refer to the end of Section 6.2 (i.e., Before migration) for the details on the migration techniques that are currently implemented in the simulator;

- **Event 3 (E3)** refers to the beginning of the process of sending data from the source to the destination FN, i.e., the service migration process actually starts here;

- **Event 4 (E4)** is when the connection handoff starts. Handoff decision-making is defined by parameters of the network and/or the device being utilised. In our ideal case, this event occurs after the migration starts (i.e., after E3), so that it can be contained within the service migration phase. The duration of the connection handoff is very likely to be lower than that of service migration, indeed. However, it is worth noting that some migration policies may consider also a reactive migration, i.e., one where service migration starts after the beginning of connection handoff;

- **Event 5 (E5)** is the end of the connection handoff process. Now, the user's device is connected to the new access point. Meanwhile, ideally, service migration is still occurring;

- **Event 6 (E6)** is the end of the service migration process. The data transfer is completed and the service is now running on the destination FN;

- **Event 7 (E7)** represents the first access from the user to the service running on the new FN, which means that at this point networking connections must be re-established to reflect the new location;
- **Event 8 (E8)** represents the point in time where the whole process is complete. We highlight that E7 and E8 coincide if service migration finishes after the connection handoff (i.e., E6 occurs after E5). However, if service migration finishes before the connection handoff (i.e., E6 occurs before E5), E7 takes place before E8.

Based on the above events, we discuss different scenarios that can take place when users move, illustrating: (i) the source FN, which is hosting and running the service before migration; (ii) a wireless link that connects the mobile device to the access point where the source FN is connected; and (iii) the destination FN connected to the destination access point (i.e., that after the wireless handoff). The two FNs are connected through a cabled network (WAN or LAN), which is used to transfer the service. The connection between FNs can present different topologies depending on how they are deployed.

Scenario 1 (Proactive migration) - Figure 6.1 considers that the migration process starts (i.e., E1) before the user reaches the point of performing the wireless connection handoff. This way, when E1 occurs, the service is migrated without an abrupt connection interruption from the wireless handoff. Ideally, the migration and the handoff should end at the same time to reduce the delay to the user: note that the longer the migration takes to finish after the handoff, the longer the user will have to access the source FN through the new access point (which means more than one network hop). On the one hand, proactive migration may improve overall performance, since the user's service is ideally available at the destination FN once the wireless connection handoff is finished. On the other hand, though, it might worsen performance if user's mobility prediction is erroneous.

Scenario 2 (Reactive migration) - As the migration and handoff decision-making are assumed to be independent from one another, these can occur at any time along the user's path. Therefore, the events timeline can change, for example, with the migration process starting (i.e., E3) after the handoff process has begun (i.e., E4). Figure 6.2 shows the case in which the whole migration process takes place once the handoff process is finished. However, the two processes can partially overlap; this depends on the actual policies and algorithms adopted for migration. Reactive migration may lead to increased delays and potential drops in the QoE observed by the user. This is because, when the handoff process is finished, the user is still accessing the source FN resources through the new access point: the application goes through more than one hop to access the source FN. Service migration towards

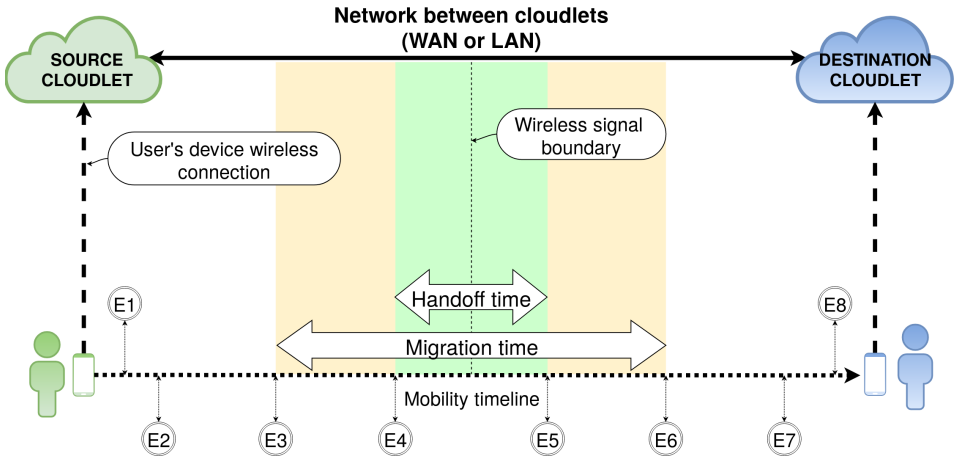


Figure 6.1: Proactive migration scenario - migration starts before the handoff.

the destination FN terminates after a while. Nevertheless, reactive migration may prove beneficial in situations where user's mobility pattern is highly unpredictable, and it is therefore better to know the user's new position prior to migrating the service.

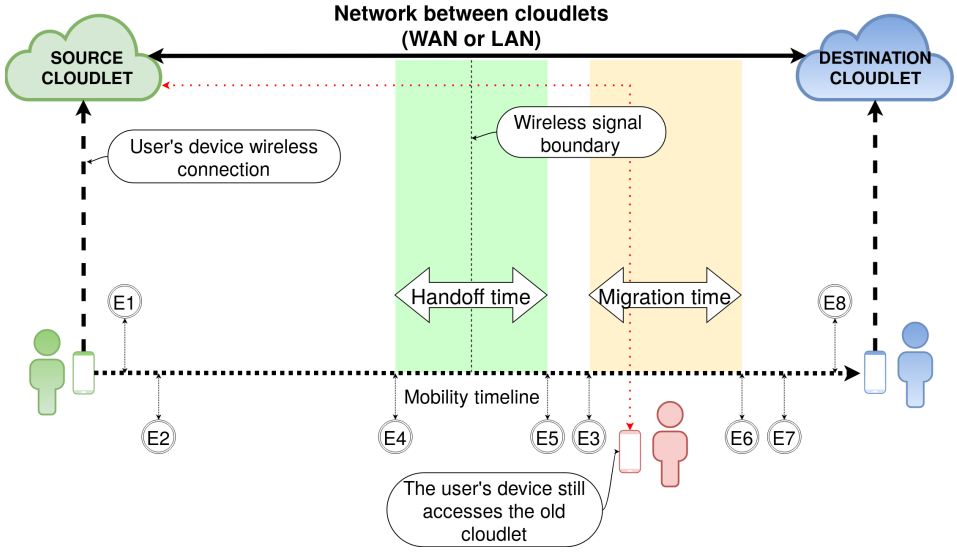


Figure 6.2: Reactive migration scenario - migration starts after the handoff.

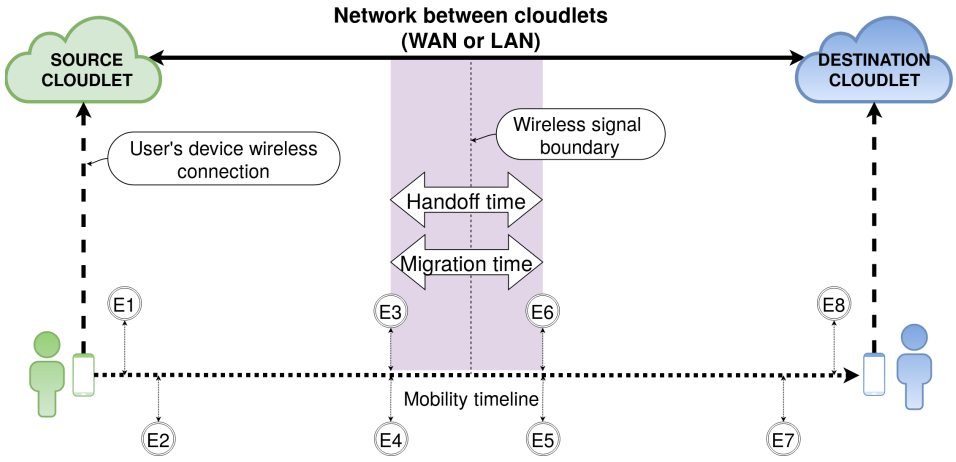


Figure 6.3: Concurrent migration scenario - handoff and migration start and end simultaneously.

Scenario 3 (Concurrent migration) - A third possible scenario is such that both handoff and migration start and end simultaneously (see Figure 6.3). Although this is feasible, it is unlikely, since service migration takes longer than a handoff process. We consider this scenario as a complement to Scenario 1: the shorter the time difference between migration and handoff, the better, as the application would always have access to the service on the closest FN during the migration process. In this scenario, differently from Scenarios 1 and 2, the service does not need to be accessed through another access point. In this case, the downtimes of the handoff and of service migration coincide, minimising the delays experienced by the user.

6.2 MobFogSim design

MobFogSim is a simulator that extends iFogSim to model device mobility and service migration in FC. In this section, we first provide a brief overview of iFogSim and then present the design of MobFogSim with respect to device mobility and service migration.

Overview of iFogSim

iFogSim (Gupta et al., 2017) is an extension of CloudSim (Calheiros et al., 2011) that allows the simulation of FC environments. It supports the configuration of a Fog/Cloud hierarchy by defining the connections among end devices, FNs, and a Cloud DC. The main classes of iFogSim, which are implemented in Java, are briefly

described below in order to introduce the simulator and present the necessary background to describe the modifications of MobFogSim later in this section.

iFogSim has kept the CloudSim core implementation to realise the processing of events among Fog components. Besides, iFogSim has created new classes and methods to run a FC simulation. Its main components are:

- **FogDevice -** it presents the hardware features of a FN or IoT device. It extends *PowerDatacenter* from CloudSim. RAM, MIPS, storage size, and bandwidth (uplink and downlink) are the main class attributes. The methods of this class perform specific tasks of a FN or IoT device to process the received tuples;
- **Tuple -** it extends *Cloudlet*[53] of CloudSim. This represents a task created by an IoT device that is sent to an AppModule to be processed;
- **Application -** it is designed according to a Directed Acyclic Graph (DAG). The vertices are the execution components (i.e., AppModules) that execute tuples. The edges are data dependencies between the vertices. The following classes are used to instantiate applications:
  - *AppModule* - it is a vertex that processes tuples. This class extends *PowerVm* from CloudSim;
  - *AppEdge* - it is an edge to link a couple of vertices (AppModules), thus to create a dependency between these entities;
  - *AppLoop* - it is a DAG flow that can be configured by users of iFogSim for monitoring latencies in that part of the DAG. For instance, the AppLoop may be set to start at the initial vertex of the DAG (entry node) and end at the last vertex (exit node).
- **Sensor -** objects of this class are responsible for creating the application tuples;
- **Actuator -** objects of this class receive the tuples processed by an AppModule.

iFogSim already implements important components for simulating FC environments. However, this simulator does not model: (i) mobile devices; (ii) geographical position; (iii) wireless access points; and (iv) service migration. Motivated by these lacks, in this chapter we propose MobFogSim, which builds upon iFogSim to support device mobility and service migration in FC simulations.

[53] In CloudSim, a Cloudlet is an application running in the Cloud.

Design considerations

An architecture for service migration in the Fog has been presented in (Bittencourt et al., 2015) and is composed of three main layers: (i) Cloud DC; (ii) FNs; and (iii) Fog-enabled IoT devices. Based on that, we devised the necessary steps for designing service migration between two FNs in this Fog architecture. We first provide a general overview of migration design in MobFogSim (see Algorithm 3) and then go into more details.

When the user gets close to crossing the wireless network boundary and, consequently, is close to a possible handoff process from the current access point to the next one, the Fog infrastructure should start the migration process. The **Migration Decision** process (i.e., line 2 in Algorithm 3) makes the migration decision based on policies and strategies that are established by the migration system. Migrations occur when the system identifies a better FN to place the user's service (it is based on a metric like lowest latency, lowest distance, or another metric, as we discuss in *Migration strategy* later in this section). In scenarios where there are no FNs available offering better conditions to the user, the current FN remains the only alternative to run the user's service (i.e., line 13 in Algorithm 3). If the policy/strategy judges that it is necessary to perform a migration process, the migration point, which defines how close the user is to the wireless network boundary (see *Migration policy* later in this section), determines when the migration starts. When the user crosses the migration point, the system is allowed to start the migration. Once these conditions (i.e., where and when to migrate) are satisfied, the system must prepare the data for migration. This process happens in the **Before Migration** phase, which is discussed in more details in *Before migration* at the end of this section. When the Before Migration phase finishes, the migration system has the necessary information to start the migration. In the **During Migration** phase, the system must monitor, manage,

Algorithm 3: Migration overview.

```
1  while User u is within the migration zone do
2      if MigrationDecision() is TRUE then
3          if User u is at the migration point established by the Migration Policy then
4              Before Migration: prepare_migration();
5              while Migration is being performed do
6                  if Cannot migrate then
7                      Fail and leave the service at the source FN;
8              end
9              After migration: reconfigure_network();
10     else
11         Leave the service at the source FN;
12 end
```

and synchronise the process, depending on the type of migration being used (e.g., cold or live). If anything impairs service migration, either the management system tries to solve the problem at runtime, or the migration might be aborted, and the migration process terminates leaving the service on the source FN. On the other hand, if everything runs as expected, the migration process goes to the **After Migration** phase and informs the user's mobile device to close the connection with the old FN and access the new one. Let us now go into more depth by analysing Migration Policy, Migration Strategy, and Before Migration in MobFogSim.

Migration policy

Migration Policy in MobFogSim determines the timings of service migration by defining the concepts of migration zone and migration point in the map. The current model implemented in MobFogSim is based on the migration policy discussed in (Lopes et al., 2017). An illustration of that model can be seen in Figure 6.4, which shows some of the parameters to be monitored.

Users 1 and 2 have different speeds, directions, and geographical positions in the map. These attributes are verified during the Migration Decision process. The system monitors the users and decides whether their services should be migrated or not. The **migration zone** is an area where migration decisions are constantly computed. Looking at Figure 6.4, this is the area limited within the migration point (which is any point along the dashed red line in figure). Once a Migration Decision

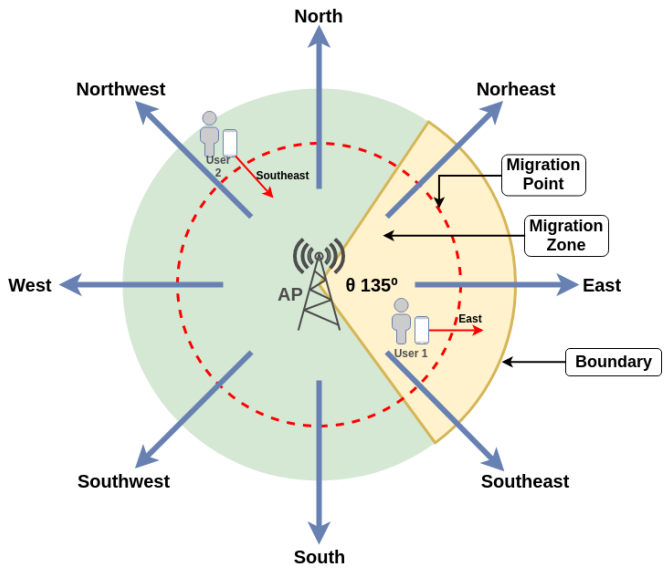


Figure 6.4: Migration model and parameters.

returns TRUE, service migration is actually started only when the user reaches the migration point. The potential destination FNs considered in a migration decision are limited by the migration cone (i.e., the yellow area in Figure 6.4). This cone is extended to the next access points in the map, thus limiting the amount of possible destination FNs for the user's service and also contributing to a speedup in the underlying optimisation process performed by Migration Strategy, which is described next. This cone is defined by:

- the two directions adjacent to the current direction of the user (e.g., the direction of user 1 is East, then the adjacent edges are Northeast and Southeast).
- an angle θ that defines the relative region between the access point and the user (e.g., 135° in Figure 6.4).

This cone is always constructed on the same side and direction of the moving user. Note that user 2 does not have a cone because she/he is already connected to the access point and is moving towards this access point, i.e., her/his direction is Southeast, but her/his position relative to the access point is Northwest. On the other hand, the cone for user 1 is shown because her/his direction is East, and her/his position relative to the access point is also East.

As already outlined, the **migration point** is a point on the map where service migration should be started before the handoff mechanism occurs (i.e., proactive migration). The migration point can be set depending on characteristics of the infrastructure and the wireless connection, e.g., taking into account how wireless handoff policies behave. A migration point can be either static or dynamic. A static migration point is fixed on the map regardless of other parameters. For example, Figure 6.5 shows an example of a static migration point that is defined as when the user has

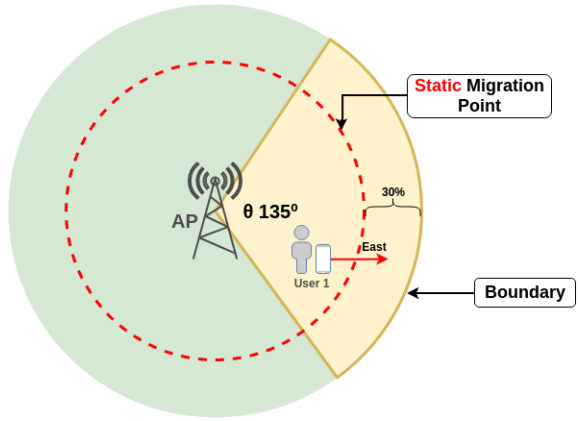


Figure 6.5: Static Migration point.

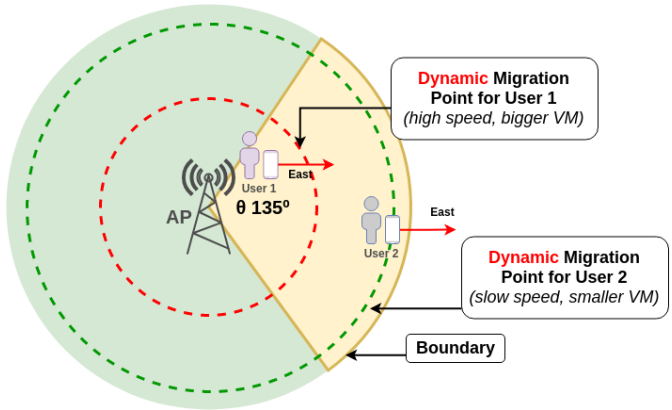


Figure 6.6: Dynamic Migration point.

already travelled 70% of the radius of the coverage area, thus still remaining 30% of the distance to travel before the expected handoff occurrence point (boundary). Instead, a dynamic migration point can take into consideration other parameters, as for example the size of the data being migrated as well as the user's speed. Figure 6.6 shows a scenario with two examples that combine data size and user's speed. User 1 has a larger volume of data to migrate and has a higher speed; user 2 has a smaller volume of data and a slower speed. Hence, with a dynamic migration point, service migration for user 2 starts later than that for user 1.

Migration strategy

In the first part of Algorithm 3, Migration Decision decides whether the user's service should be migrated or not. The Migration Decision flow is illustrated in Figure 6.7. First, the algorithm verifies if the user is moving, and if so, the migration decision process will start by discovering the relative user's location in relation to the access point, aiming at verifying if the user is in the migration zone. If the user is in the migration zone, the algorithm chooses a new FN to receive the user's service. This choice depends on Migration Strategy.

Three different strategies are currently available in MobFogSim:

- **the lowest distance between the user and the access point -** this strategy chooses the FN connected to the access point that is geographically closest to the user;
- **the lowest distance between the user and the FN -** this strategy chooses the FN that is geographically closest to the user;
- **the lowest latency -** this strategy chooses the FN with the lowest latency to the user (i.e., the topologically closest).

The first two strategies are calculated based on the geographical distance between the user and the access point/FN. We highlight that the FC paradigm suggests to provide computing and network resources as close as possible to the users. However, many aspects may impact the quality of the connection between the users and their applications placed in the Fog. The geographically closest FN or access point may not necessarily offer the lowest latency to the user. By implementing these strategies, MobFogSim allows to study this correlation between geographical and topological distance. The third strategy, instead, takes into consideration the end-to-end latency between the user and the destination FN. This value is calculated as the sum between the latency from the user to the access point and that from the access point to the destination FN. We highlight that researchers/developers can extend these strategies and implement their own algorithms, for instance by choosing the destination FN that offers the lowest migration time or energy consumption, or by including load balancing mechanisms.

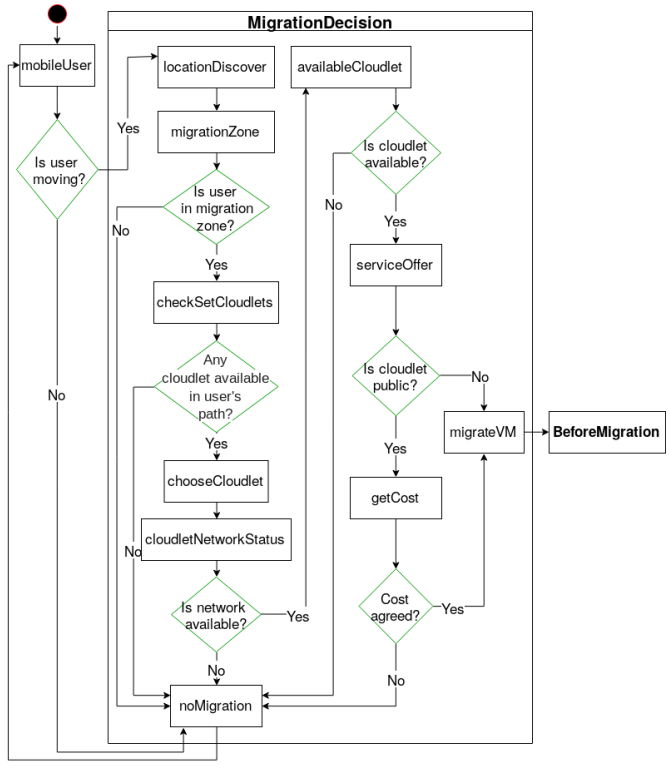


Figure 6.7: Migration Decision component.

Before migration

Once the Migration Decision component has decided that it is time to migrate the service, the Before Migration component intervenes. Figure 6.8 illustrates the steps performed by this component. The first two steps of Before Migration are *dataPrepare* and *replicaVM*. They are respectively used to define the way through which data are prepared for migration and the way used to actually migrate them. After this, the third step opens the connection between the source and the destination FN. Once the connection is established, data transfer starts through the network. Note that data preparation and transmission are performed according to a certain migration technique. As discussed in Chapter 4, there exist four migration techniques at the moment of writing, i.e., cold, pre-copy, post-copy, and hybrid migrations. As of now, MobFogSim models two of these techniques: cold and post-copy migrations. However, we highlight that developers are welcome to implement the remaining techniques and include them in the simulator.

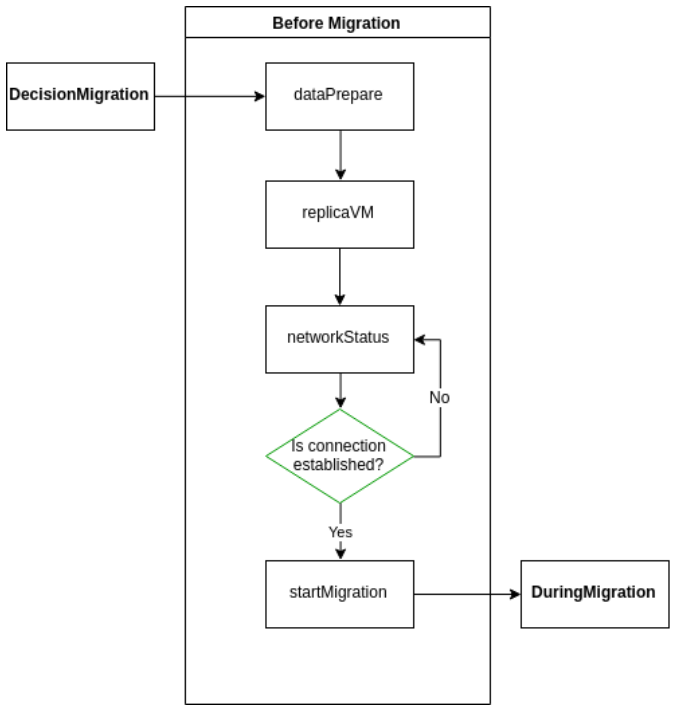


Figure 6.8: Before Migration component.

6.3 MobFogSim implementation

In this section, we discuss the most noteworthy aspects relative to the implementation of MobFogSim. Firstly, we report the main Java classes that make MobFogSim an extension of iFogSim with support to device mobility and service migration. Then, we focus on the simulation events and their flow within the migration and handoff procedure. Finally, we illustrate how MobFogSim supports realistic user's mobility patterns.

MobFogSim as an extension of iFogSim

Figure 6.9 shows the main Java classes present in the simulator. The leftmost part includes PowerDatacenter, which is the class from CloudSim that is important for the creation of the relevant entities in iFogSim and MobFogSim. The classes from iFogSim reside in the middle of the picture. The rightmost part, instead, shows the main classes introduced with MobFogSim. These allow to model device mobility, network handoff, and service migration according to the description from the previous section. In what follows, we describe such classes:

- **Coordinate -** this class acts as a Cartesian Plan map (X, Y) to have all entities

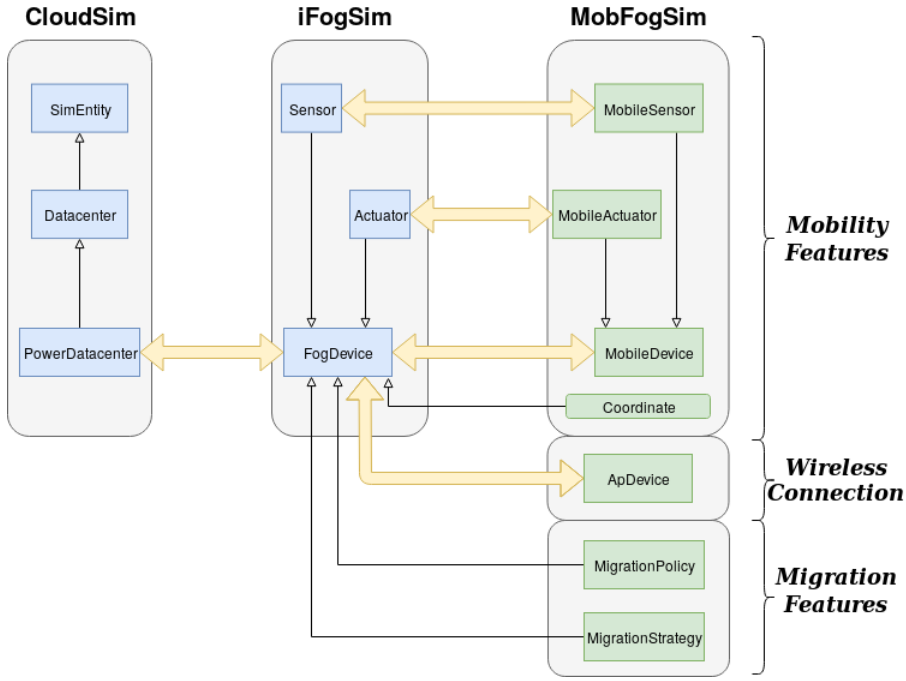


Figure 6.9: Overview of MobFogSim as an extension of iFogSim and CloudSim.

position during the simulation. These entities can be the FNs, the wireless access points, and the users' (IoT) devices. The developer can configure the map boundaries;

- **ApDevice -** this class extends FogDevice and has the access point responsibility in a wireless network. This class manages the handoff mechanisms and connections/disconnections of end devices;
- **MobileDevice -** this class also extends FogDevice. Its main goal is to allow a separation between FNs and IoT devices, since iFogSim implements any device (FNs and IoT devices) with the same features. With this separation, it is possible to have specific features for different devices;
- **MobileSensor -** this class extends the Sensor class. In iFogSim, to build a scenario with multiple sensors, the developer needs to instantiate several Sensor objects. In MobFogSim, MobileSensor already has a set of sensors, and the developer can instantiate only one object and add, when necessary, more sensors into the same hardware. A MobileSensor is associated to a MobileDevice;
- **MobileActuator -** this class is at the same level of abstraction of MobileSensor and extends the Actuator class from iFogSim;
- **MigrationStrategy -** this class implements the migration strategy to be applied in the simulation;
- **MigrationPolicy -** this class implements the migration policy to be applied in the simulation.

These classes implement the migration model presented in Section 6.1, thus creating a mobile simulation environment in MobFogSim that includes the decision-making algorithms described earlier in this chapter, the events related to the migration, and the localisation system.

Implementation of events in MobFogSim

To summarise the core of MobFogSim, Figure 6.10 shows part of the main events generated in a simulation. On the left-hand side, AppExample is a class where a researcher/developer builds all the configurations for the simulation. In this class, one can perform steps to, for example: (i) create all the FogDevices and their features; (ii) create all the MobileDevices, MobileSensors, and MobileActuators; (iii) create the broker configuration; (iv) create the application; (v) create the network; and (vi) schedule all the initial simulation events. After all the settings are in place, the MobileController class controls and schedules all the events in the simulator that are run by the different types of devices (i.e., FogDevice, ApDevice, and MobileDevice).

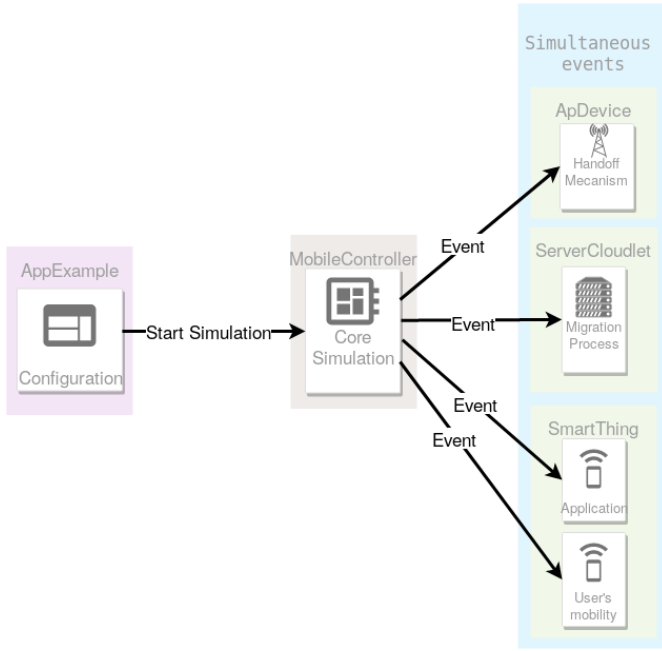


Figure 6.10: The main events generated in a simulation.

FogDevice is responsible for executing all the events related to migration (e.g., verify if the Fog service should migrate according to some migration policy). ApDevice is responsible for executing all the events associated with the handoff mechanism (e.g., make the disconnection from the source access point and make the connection to the destination one). MobileDevice is responsible for executing the user's end application and for implementing mobility (e.g., start processing tuples provided by the sensors; move the user's device according to the new geographical position). An essential aspect of the simulation events is that each device can schedule events at the same simulation time, since those devices are independent entities.

For the sake of comprehensiveness, in Figure 6.11, we illustrate the flow of events that occur in MobFogSim during a migration and handoff procedure, along with the classes involved. The purpose is to show how the sequence of events E1-E8 from Section 6.1 is implemented in the simulator. The scenario considered in Figure 6.11 is such that: (i) service migration is proactive; (ii) it is based on the post-copy migration technique; and (iii) uses "lowest latency" as migration strategy. However, we highlight that the events would be the same also under different scenarios, though with a different flow and/or different classes involved. Note that elements in the sequence diagram are coloured differently according to their nature. Specifically, purple boxes represent Java classes. Blue and green arrows represent events relative to migration and handoff, respectively. The brown arrow represents an event

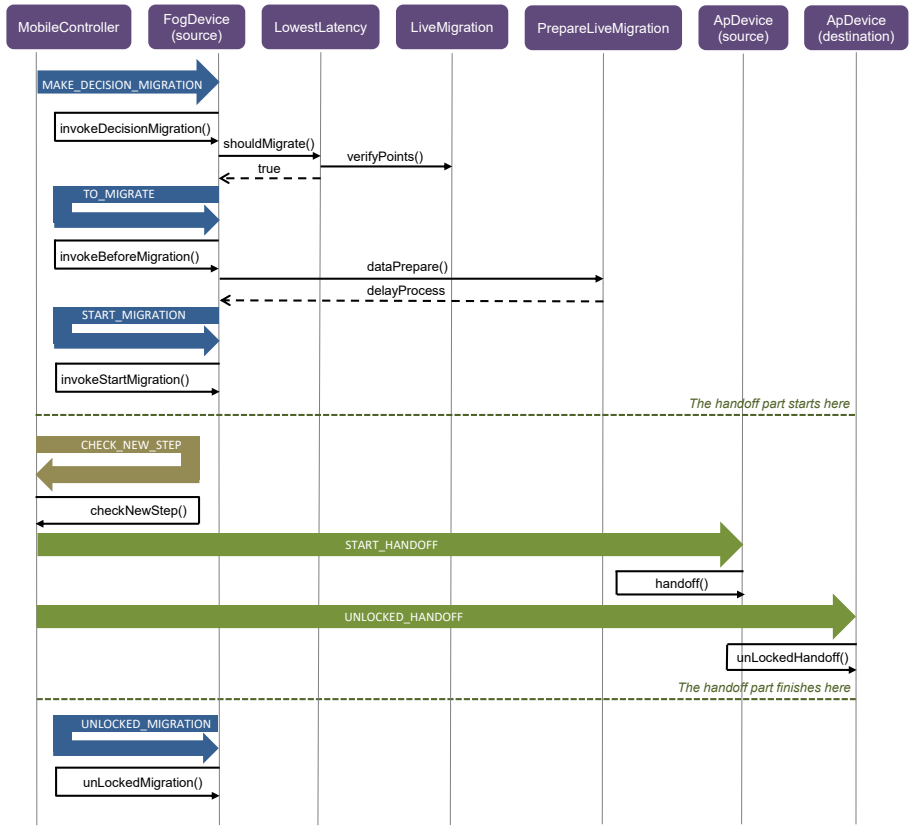


Figure 6.11: Migration and handoff process as implemented in MobFogSim.

in MobFogSim that is neither explicitly associated to migration nor handoff. Finally, black arrows are invocation of methods (and their eventual return values).

By looking at Figure 6.11, it is possible to note that the first event is (periodically) scheduled by MobileController to the source FogDevice. This event is called **MAKE_DECISION_MIGRATION**, and it corresponds to E1 from Section 6.1 if the decision is to migrate the service. In order to make this decision, FogDevice executes the *invokeDecisionMigration*() method, which in turn invokes the *shouldMigrate*() method of the LowestLatency class. The latter triggers the *verifyPoints*() method of LiveMigration, which inspects the relative position of the mobile device with respect to the current access point to find out if the user is in the migration zone and in the migration point. Moreover, verifyPoints() calculates the migration time of the service. With the information set by verifyPoints(), shouldMigrate() is able to decide that it is time to migrate the service (i.e., it returns *true* to invokeDecisionMi-

gration()) and also selects the destination FN based on the migration strategy. Once the decision to migrate is made, the source FogDevice sends a **TO_MIGRATE** event (i.e., E2 from Section 6.1) to itself, which results in the invocation of *invokeBeforeMigration*(). This method triggers the *dataPrepare*() method of the PrepareLiveMigration class, which calculates the time required to checkpoint the state of the service as well as that to open the connection towards the destination FN. Both these times are returned, as delayProcess, to the source FogDevice class. Service migration is then started by the source FogDevice by issuing the **START_MIGRATION** event (i.e., E3 from Section 6.1), with the consequent invocation of *invokeStartMigration*. This method disassociates the service from the source FN and associates it to the destination one. Then (and while the service is being migrated), the handoff part of the procedure is carried out. Going into detail, MobileController (periodically) issues the CHECK_NEW_STEP event to itself and executes the checkNewStep() method. This method verifies that the mobile device is in the handoff occurrence point (i.e., at the boundary between the coverage areas of two access points) and hence calculates the destination access point and the handoff time. Next, MobileController sends the **START_HANDOFF** event (i.e., E4 from Section 6.1) to the source access point, which invokes the *handoff*() method to disconnect the mobile device from the source access point and associate it to the destination one. Then, MobileController sends the **UNLOCKED_HANDOFF** event (i.e., E5 from Section 6.1) to the destination access point, which triggers the *unLockedHandoff*() method to set the end of the connection handoff process. Finally, the migration process concludes similarly to the handoff one, namely through the invocation of the **UNLOCKED_MIGRATION** event (i.e., E6 from Section 6.1) and of the *unLockedMigration*() method. Thus, the whole process is finished, and the user, who is now connected to the destination access point, can access the service on the destination FN (i.e., E7/E8 from Section 6.1).

Support to realistic user's mobility

The modifications made to iFogSim, which resulted in MobFogSim, introduced support to mobile devices. It is worth noting that our simulator supports realistic users' mobility patterns. This is made possible thanks to the integration of MobFogSim with the mobility tool **Simulation of Urban Mobility** (SUMO) (Behrisch et al., 2011). Figure 6.12 presents an example of the data flow used in a simulation. In this example, SUMO (b) interprets a source mobility database saved in, for example, .XML format (a). Many realistic mobility databases are available in .XML format, such as that from the project Luxembourg SUMO Traffic (LuST), which presents realistic data from vehicles in Luxembourg (Codeca et al., 2015). The result of SUMO is then saved in .csv format (c).

Each .csv file represents the mobility of a vehicle in the simulation. Each line in

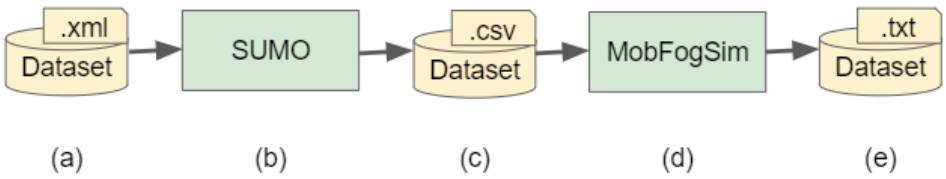


Figure 6.12: Example of the data flow in a simulation.

this file contains data from the vehicle at one point in the simulation. The data are: (i) position x and y on the map; (ii) speed in metres per second; (iii) the direction in radiants; and (iv) the simulation time in which these data were collected. This new database is hence used as a basis to define the users' mobility in MobFogSim. The simulator interprets this database and makes some modifications to adapt these data to its mobility model. Among these changes, there are, for example, the conversion of the vehicle speed from metres per second to kilometres per hour as well as the conversion of the direction from radiants to the eight main cardinal points, which are the basis of the mobility model in MobFogSim. After the simulation in MobFogSim (d), a new database is built (e), which presents the results of user's behaviour for local resource management. Among these results, there are: (i) the average and the maximum latency presented by the application along the user's path; (ii) the migrations performed; (iii) the packages requested and attended; and (iv) the number of handoffs.

6.4 MobFogSim calibration

We now report the experiments that we carried out over the real testbed from Chapter 4 in order to calibrate MobFogSim, i.e., to appropriately configure its input values for simulation. The objective is to replicate in the simulator the same conditions of the testbed. Once the testbed setup is reproduced in MobFogSim, indeed, we can validate the latter by comparing its simulation results of container migration with the results presented in Chapter 4. The actual validation of the simulator is reported in the next section.

Calibration of maximum MIPS rating

There exist multiple ways to estimate a computer speed; one of these is to measure speed in Million Instructions Per Second (MIPS). MobFogSim, as iFogSim, takes MIPS ratings as inputs to indicate the maximum computation speed of devices in an IoT-Fog environment and to define the execution speed of tasks. The purpose of this first group of calibration experiments is to obtain the maximum MIPS ratings of the devices in the real testbed.

As reported in Section 4.3, the end device in the testbed is an ASUS Zenbook UX331UN notebook whose specifications are reported in Table 6.1. To calculate the maximum MIPS rating of this device, we leveraged the *cpumaxmp64* executable, which is one of the Roy Longbottom's Linux MultiThreading benchmarks (Longbottom, 2010). This benchmark performs 64bit integer add instructions over 64bit registers via assembly language. It is possible to specify the number of threads, between 1 and 64, as a command-line parameter. Each thread executes independent code. The assembly code loops execute two billion add instructions each. We carried out experiments with 1, 2, 4, 8, and 16 threads, running the benchmark five times for each number of threads. The blue bar charts in Figure 6.13 represent the obtained results for the notebook, with a 95% confidence level. As shown, MIPS rating increases with the number of threads, reaching an average value of 46 533.80 MIPS with 8 threads. With 16 threads, however, MIPS rating slightly decreases to 45 441.20 MIPS. We were expecting this outcome, as the considered notebook features an Intel i7-8550U CPU, which is a Quad-Core processor with Hyper-Threading technology[54]. Hyper-Threading is the Intel implementation of Simultaneous MultiThreading (SMT), which makes a physical core appear to the OS as two logical processors (Marr et al., 2002). Therefore, this notebook has eight logical processors, which explains why performance with 8 threads is the highest.

We then calculated the maximum MIPS rating of a Raspberry Pi 3 Model B, whose specifications are detailed in Table 6.1. As described in Section 4.3, in the real testbed, both the source and the destination FNs are Raspberry Pis. This time, we exploited the *MP-DHRYPi64* executable, which belongs to the Roy Longbottom's Raspberry Pi benchmark collection (Longbottom, 2013b). Each run of MP-DHRYPi64 executes 1, 2, 4, and 8 threads, with each thread executing a copy of the Dhrystone benchmark. Dedicated data arrays are used for each thread, but there are numerous other variables that are shared. The Dhrystone benchmark provides a measure of integer performance and has been the key standard benchmark since

Table 6.1: Device specifications.

Device	CPU	RAM	Storage	Architecture	OS	Kernel	Technology
ASUS Zenbook UX331UN	Quad-Core 1.8 GHz (Turbo at 4.0 GHz)	16 GB	512 GB	x86_64	Ubuntu 18.04.1	Linux 4.15.0	SMT
Raspberry Pi 3 Model B	Quad-Core 1.2 GHz	1 GB	16 GB	aarch64	Debian 9.5	Linux 4.14.73	SMP

[54]See `https://www.intel.com/content/www/us/en/architecture-and-technology/hyper-threading/hyper-threading-technology.html`. Last accessed: 5 April 2019.

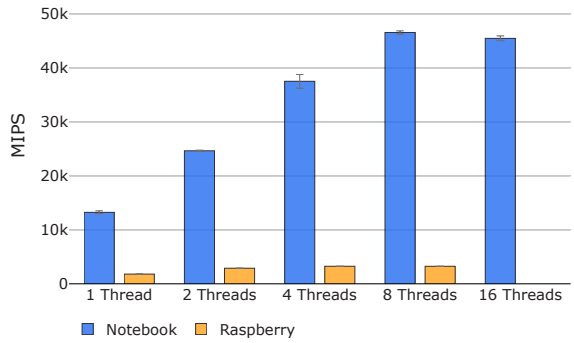


Figure 6.13: Maximum MIPS ratings.

1984. Its code, which is written in C, includes simple integer arithmetic, string operations, logic decisions, and memory accesses. Between 21% and 65% of the overall execution time is spent on string operations (i.e., string assignments and comparisons) (York, 2002). Speed was originally measured in Dhrystones per second. This was later changed to VAX MIPS by dividing Dhrystones per second by 1757, which is the number of Dhrystones per second of the first 1 MIPS minicomputer, namely the DEC VAX 11/780 (Longbottom, 2013a). We run MP-DHRYPi64 five times. Results for the Raspberry Pi are the orange bar charts in Figure 6.13 and are shown with a 95% confidence level. As for the notebook, MIPS rating increases with the number of threads. However, differently from the notebook, the maximum MIPS rating of the Raspberry Pi is reached with 4 threads rather than 8. Average computation speed is 2873.50 MIPS with two threads, 3234.33 MIPS with 4 threads, and 3231.95 MIPS with 8 threads. This is because the Raspberry Pi 3 Model B has a Broadcom BCM2837 CPU, which is a Quad-Core processor with Symmetric Multi-Processing (SMP) technology rather than SMT. As a result, the OS in a Raspberry Pi sees only four logical/physical processors, and this is why the highest performance is reached with 4 threads.

Calibration of latency and throughput

The objective of this second set of calibration experiments is to characterise the RTTs and the throughputs among the devices in the testbed. As described in Section 4.3, container migration experiments over the testbed were performed with two different couples of RTT and throughput values between the FNs. The first couple was defined as *LTE use case* and consisted of a RTT of 122.95 ± 5.57 ms and a throughput of 11.34 ± 2.31 Mbps. The second couple was named *Ethernet use case* and was characterised by a RTT of 6.94 ± 0.61 ms and a throughput of 72.41 ± 3.87 Mbps. Each couple hence identifies a specific network condition between the FNs. For simplic-

ity, in this chapter, we refer to the LTE and Ethernet use case couples as **network condition A** and **network condition B**, respectively. Besides, note that legend entries in all the following figures with results report the one-way latency rather than the RTT between FNs. This is because MobFogSim takes one-way latencies as input.

As shown in Figure 4.6 from Section 4.3, the notebook in the testbed is connected through Wi-Fi to the source FN, which hence also behaves as a Wi-Fi access point. RTTs and throughputs over Wi-Fi were measured using the same methodology exploited to calculate values for network configuration between FNs. Namely, RTTs were measured through the *ping* command in Linux over 10 runs, with 20 measurements per run. We then considered the average RTT per run to obtain 9.55 ± 0.88 ms at a 95% confidence level. In order to get the throughput from the notebook to the Raspberry Pi, we performed 10 measurements using the *iperf3* tool, sending 50 MB each time. The resulting throughput value is 13.32 ± 0.97 Mbps, with a 95% confidence level. In a similar way, we calculated the throughput from the Raspberry Pi to the notebook, obtaining 13.05 ± 1.40 Mbps.

Calibration of the application parameters

The purpose of this final set of calibration experiments is to characterise the application that was used in Chapter 4 to evaluate container migration techniques. By characterisation of the application, we mean the measurement of all those values that MobFogSim requires as inputs to detail the application. We highlight that, in Chapter 4, the server could modify memory pages either at 10 kBps or at 500 kBps. The purpose for this was to evaluate the effect of different page dirtying rates on container migration techniques. However, MobFogSim currently implements migration according to the cold and post-copy techniques. As shown in Chapter 4, both these techniques are not influenced by the page dirtying rate of the service. Therefore, in this chapter, we consider 500 kBps as the only page dirtying rate featured by the CoAP server.

The first parameter that we measured to characterise the application were the MIPS ratings with which both the client and the server execute. To this purpose, we run both the client and the server specifying the *stat* command of the *Linux perf*[55] utility (version 4.14.87 on the Raspberry Pi and 4.15.18 on the notebook). Linux perf is a user-space application that, making use of the *perf_events* interface of the Linux kernel, accesses all the CPU internal counters for performance monitoring. We run the application five times and, based on the performance metrics from Linux perf, we calculated MIPS ratings as follows:

$$MIPS = \frac{f \cdot ipc}{10^6} \tag{6.1}$$

[55] See `http://www.brendangregg.com/perf.html`. Last accessed: 10 April 2019.

where f is the CPU frequency (Hz) with which the task is executed, while ipc (i.e., Instructions Per Cycle) is the average number of instructions that are executed per clock cycle. The resulting task execution speeds are 2901.00 ± 119.26 MIPS and 281 MIPS for the client and the server, respectively. Results are shown with a 95% confidence level. With Linux perf, we also obtained the number of instructions that were executed each time by both the client and the server. The client executes 966.01 ± 13.32 million of instructions, while the server executes 2438.62 ± 22.72 million of instructions, with a 95% confidence level.

We also measured the RAM requirements of both the client and the server. We consider RAM requirements in MobFogSim to be expressed as the maximum Resident Set Size (RSS) of a process during its lifetime, namely the maximum portion of RAM memory occupied by that process. To measure it, we specified the command[56] */usr/bin/time -f "RSS=%M"* when starting both the client and the server from terminal. After five runs, we obtained 49.05 ± 0.14 MB and 128.09 ± 0.16 MB as RAM requirements of the client and the server respectively, with a 95% confidence level. Besides, we leveraged the *du* command in Linux to measure the disk usage of both the client (i.e., the JAR file) and the server (i.e., the OCI bundle for the runC container). Results are of 4 MB for the client and 412 MB for the server.

Finally, we also measured the size of each CoAP request (i.e., from the client to the server) and each CoAP response (i.e., from the server to the client). To do so, we launched Wireshark (version 2.6.6) on the notebook and found that the CoAP request was 87 B while the CoAP response was 54 B.

6.5 MobFogSim validation

In this section, we validate our simulator by comparing its results of container migration with those discussed in Section 4.4. Since MobFogSim currently models cold and post-copy migrations, we consider only results of these migration techniques in this section. The calibration experiments described in the previous section allowed us to select realistic values for the input parameters in MobFogSim, thus to replicate the network and service conditions of the real testbed during simulation. For convenience, Table 6.2 reports these parameters in alphabetical order along with their values. Names in brackets are the actual variable names used in MobFogSim to indicate those parameters. We run 30 simulations for each combination of migration technique (i.e., cold, post-copy) with network condition (i.e., A, B). Simulation results are analysed and compared with those from the real testbed in terms of: (i) total migration time; (ii) downtime; and (iii) volume of transferred data. Results are presented with a 95% confidence interval.

In addition to the input parameters provided by the experiments on the testbed, some complementary input values were assumed as part of the simulated environ-

[56] See `http://man7.org/linux/man-pages/man1/time.1.html`. Last accessed: 10 April 2019.

Table 6.2: Input parameters and their values in MobFogSim based on the testbed experiments.

Parameter	Value
Client execution speed (*mips*)	2901 MIPS
CoAP request size (*tupleNwLength*)	87 B
CoAP response size (*tupleNwLength*)	54 B
Disk usage of client (*size*)	4 MB
Disk usage of server (*size*)	412 MB
Maximum speed of notebook (*mips*)	46 534 MIPS
Maximum speed of Raspberry Pi (*mips*)	3234 MIPS
Number of instructions executed by client (*tupleCpuLength*)	966 million
Number of instructions executed by server (*tupleCpuLength*)	2439 million
One-way latency between notebook and Raspberry Pi (*UplinkLatency*)	4.78 ms
One-way latency under condition A between Raspberry Pis (*lat*)	61.48 ms
One-way latency under condition B between Raspberry Pis (*lat*)	3.47 ms
RAM requirement of client (*ram*)	49 MB
RAM requirement of server (*ram*)	128 MB
Server execution speed (*mips*)	281 MIPS
Throughput from notebook to Raspberry Pi (*upBw*)	13 640 kbps
Throughput from Raspberry Pi to notebook (*downBw*)	13 363 kbps
Throughput under condition A between Raspberry Pis (*bw*)	11 612 kbps
Throughput under condition B between Raspberry Pis (*bw*)	74 148 kbps

ment. These simulation settings are described as follows. In the simulated scenario, we assumed a square 10 x 10 km map with 144 uniformly distributed FNs. Each FN is connected to one access point which reaches up to 500 m of signal coverage. Each simulation assumes a uniformly distributed random direction for user's mobility. The user is supposed to cross the map at a constant speed (20 kmph) until she/he reaches the opposite map edge. However, simulations end once the user finishes her/his first container migration process. The migration point policy was defined as a static point. The migration process starts once the user reaches the migration point, which is defined at 40 m from the access point coverage boundary. The destination of the container in the migration process is chosen based on a greedy approach. The migration strategy assumed in this evaluation selects the FN with the Lowest Latency among a set of 10 candidate FNs that are present in the user's path. Aiming to evaluate a more flexible environment, we assumed three different execution state sizes transmitted during migration: 1.2 (which is the actual size from the testbed results), 6.0, and 12.8 MB. Table 6.3 summarises the above settings of the simulated environment.

Figure 6.14 presents the total migration times in MobFogSim (a) against those

Table 6.3: Additional input parameters and their values in MobFogSim.

Parameter	Value
Execution state size	1.2 MB, 6.0 MB, and 12.8 MB
Access point coverage (radius)	500 m
Number of FNs	144
Density of FNs per access points	1:1
Migration strategy	Lowest latency
Migration point policy	Static (40 m)
User's speed	Constant (20 kmph)

from the real testbed (b). Similarly to the testbed results, post-copy migration, in all its variants, presents higher values than cold migration under the correspondent network conditions. As already explained in Chapter 4, this result is due to the fact that post-copy migration transfers memory pages only upon request, and the time for such requests increases the total migration time. Going into detail, under condition A, both the simulated and the testbed post-copy scenarios present a migration time about 30% longer than that of cold migration. However, the simulated environment presents average values that are 25% higher than those in the correspondent testbed scenario. Under condition B, both the simulated cold and post-copy migrations present results that are close to the values from the testbed. In the particular case of post-copy migration, the size of the execution state transmitted in that process does not present a significant impact on the migration time. Increasing the execution state size from 1.2 to 12.8 MB resulted in an increment of about 6% in the migration time under both network conditions A and B. Based on these simulations, in general, the total migration time presented by MobFogSim tends to be consistent with the testbed results for both cold and post-copy migrations.

Another relevant metric for the evaluation of the migration techniques is the

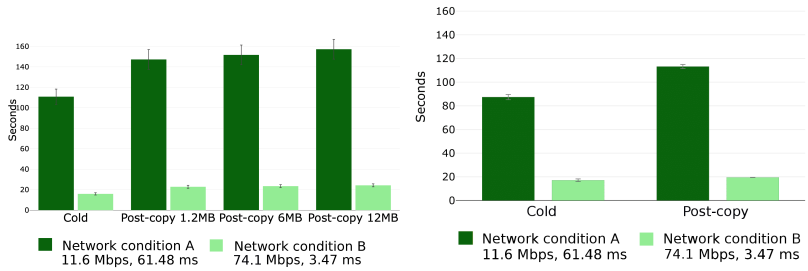
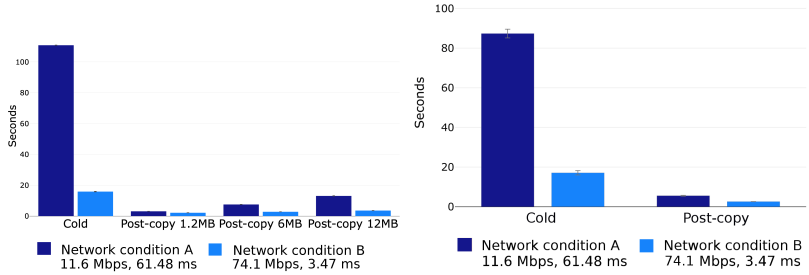


Figure 6.14: Total migration times - MobFogSim (**a**) vs. real testbed (**b**).

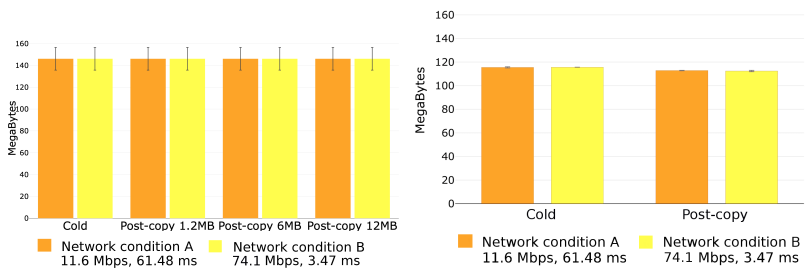


(**a**) (**b**)

Figure 6.15: Downtimes - MobFogSim (**a**) vs. real testbed (**b**).

downtime. Downtimes in MobFogSim and in the real testbed are shown in Figures 6.15(**a**) and 6.15(**b**), respectively. Similarly to the testbed results, the downtime of cold migration is higher than that of post-copy and even coincides with the total migration time, under both conditions A and B. Even though the execution state size does not have a significant impact on the migration time, this parameter strongly influences post-copy migration in terms of downtime. Assuming an execution state size of 12.8 MB, simulations suggest a downtime about 420% higher under condition A and 66% higher under condition B, if compared to migrations with an execution state size of 1.2 MB.

We also present performances in terms of volumes of data transmitted between FNs during migration. Figures 6.16(**a**) and 6.16(**b**) respectively show the amounts of sent data in MobFogSim and in the real testbed. Simulation presents a volume close to 150 MB, which is about 20 MB higher than that in the testbed environment. The volume of data sent both within simulations and over the testbed is stable under both the network conditions. Besides, like in the testbed, both the migration techniques transfer a similar volume of data (nonetheless, total migration time for post-copy is higher due to the time necessary to request memory pages).



(**a**) (**b**)

Figure 6.16: Volumes - MobFogSim (**a**) vs. real testbed (**b**).

Chapter 7

Conclusions

The Fog is a Cloud closer to the ground. As such, FC extends the Cloud towards the network edge, distributing resources and services of computing, storage, and networking anywhere along the Cloud-to-Things continuum. The resulting topological proximity to the end devices is the enabler of key benefits (e.g., low latencies, reduced bandwidth consumption) that are not achievable with Cloud-only environments. Although it is generic enough to satisfy the requirements of the most diverse domains, FC is particularly tailored to the IoT. At the beginning of this thesis, we have provided a survey on the employment of FC for the IoT. This study shows how significant steps forward have been made in the field. For instance, several software and hardware platforms are available for use. Moreover, FC is experiencing significant standardisation efforts and promising collaborations, which are fundamental for a wider and quicker adoption of this paradigm. In June 2018, the IEEE-SA officially adopted the OFRA as the new IEEE 1934^{TM} standard, while ETSI MEC will be a key feature of the next 5G networks. In addition, the recently signed MOU between the ETSI and the OFC is a first step towards further advancements in the field. Yet, as pointed out in this thesis, several open issues and research opportunities still exist to fulfil the full potential of the integration between FC and the IoT.

The focus of this thesis has been on a specific research problem, i.e., that raised by (IoT) device mobility in FC. By moving across the environment, a mobile device may get topologically far from its Fog service. This may have a negative impact on the FC benefits, which are a result of Fog proximity to the end device. To solve this issue and achieve what we refer to as CFC, we have considered Fog service migration across the Fog infrastructure, thus to let the Fog service be always close enough to the served mobile device. In this thesis, we have first analysed the different aspects that characterise this research problem, reporting the efforts from literature to address each of them and outlining the open issues and research directions. We have then proposed the following solutions and achieved the following results.

Firstly, under the assumption of Fog services implemented as application containers, we have set up a small-scale FC testbed and performed experiments over it with the purpose to evaluate and compare the state-of-the-art stateful container migration techniques. Through these experiments, we have achieved a better understanding on how container migration techniques work and differ. Besides, we have shown how no technique performs the best under any condition in a FC environment. We have therefore provided a general insight on which technique might be the most appropriate under which service and network conditions. Pre-copy and post-copy migrations appear to be the most suitable for most scenarios.

As a second contribution, we have proposed a migration platform named CFP that provides the mechanisms required to support device mobility in the Fog. Such a platform leverages containerisation and container migration to this purpose. Specifically, it implements both stateful and stateless container migrations. We have validated our CFP by running experiments over a real FC testbed and by comparing stateful and stateless migration scenarios with scenarios where the service was hosted either in the Cloud or in the Fog and was never migrated. Results have demonstrated how the proposed platform improves performances, in terms of RTTs and outage times, with respect to the Cloud and to standard FC (i.e., FC with no service migration). Furthermore, we have shown how stateless migration presents better performances than stateful migration (even though it does not transfer any state at destination).

Finally, we have presented MobFogSim, an extension of iFogSim that models device mobility and service migration in FC. We have first provided details on the migration and handoff model considered in MobFogSim as well as on the design and implementation of the simulator. The results obtained from container migration experiments over the real testbed have allowed a better modelling of service migration in MobFogSim. Moreover, we have first calibrated the simulator with input values that replicated in it the same network and service conditions of the testbed. Then, we have run simulations of container migration and compared simulation results with those obtained over the real testbed in the first of the three main contributions of this thesis. Through such a comparison, which has shown similar trends in the two types of results, we have checked the validity of MobFogSim.

Appendix A

Publications

Journal papers

1. **C. Puliafito**, C. Vallati, E. Mingozzi, G. Merlino, F. Longo and A. Puliafito, "Container Migration in the Fog: A Performance Evaluation", *MDPI Sensors*, volume 19, issue 7, article number 1488, March 2019, doi: 10.3390/s19071488. **Candidate's contributions**: implemented containerisation and container migration (together with G. Merlino); devised the methodology; set up the testbed; performed the experiments; analysed the results; reviewed the literature; wrote the paper.

2. **C. Puliafito**, E. Mingozzi, F. Longo, A. Puliafito and O. Rana, "Fog Computing for the Internet of Things: A Survey", *ACM Transactions on Internet Technology*, volume 19, issue 2, April 2019, doi: 10.1145/3301443. **Candidate's contributions**: devised the methodology; reviewed the literature; wrote the paper.

3. **C. Puliafito**, D. M. Gonçalves, M. M. Lopes, L. L. Martins, E. Madeira, E. Mingozzi, O. Rana and L. F. Bittencourt, "MobFogSim: Simulation of Mobility and Migration for Fog Computing", *Elsevier Simulation Modelling Practice and Theory*, in press, December 2019, doi: 10.1016/j.simpat.2019.102062. **Candidate's contributions**: devised part of the methodology; set up the testbed; performed part of the experiments; analysed part of the data; reviewed the literature; wrote part of the paper.

4. D. Bruneo, S. Distefano, M. Giacobbe, A. Longo Minnolo, F. Longo, G. Merlino, D. Mulfari, A. Panarello, G. Patanè, A. Puliafito, **C. Puliafito** and N. Tapas, "An IoT service ecosystem for Smart Cities: The #SmartME project", *Elsevier Internet of Things*, volume 5, pages 12-33, March 2019, doi: 10.1016/j.iot.2018.11.004. **Candidate's contributions**: implemented part of the solution; reviewed part of the literature; wrote part of the paper.

Peer reviewed conference papers

5. **C. Puliafito**, E. Mingozzi, C. Vallati, F. Longo and G. Merlino, "Companion Fog Computing: Supporting Things Mobility through Container Migration at the Edge", *4th IEEE International Conference on Smart Computing* (*SMARTCOMP 2018*), June 18-20, 2018, Taormina, Italy, doi: 10.1109/SMARTCOMP.2018.00079. **Candidate's contributions**: defined the model; designed the solution; implemented the current solution; devised the methodology; set up the testbed; performed the experiments; analysed the results; reviewed the literature; wrote the paper.

6. D. Bruneo, S. Chillari, S. Distefano, M. Giacobbe, A. Longo Minnolo, F. Longo, G. Merlino, D. Mulfari, A. Panarello, G. Patanè, A. Puliafito, **C. Puliafito**, M. Scarpa, N. Tapas and G. Visalli, "Building a Smart City Service Platform in Messina with the #SmartME Project", *32nd International Conference on Advanced Information Networking and Applications Workshops* (*WAINA 2018*), May 16-18, 2018, Krakow, Poland, doi: 10.1109/WAINA.2018.00109. **Candidate's contributions**: implemented part of the solution; reviewed part of the literature; wrote part of the paper.

7. M. Giacobbe, R. Di Pietro, **C. Puliafito** and M. Scarpa, "J2CBROKER: A Service Broker Simulation Tool For Cooperative Clouds", *10th EAI International Conference on Performance Evaluation Methodologies and Tools* (*VALUETOOLS 2016*), October 26-28, 2016, Taormina, Italy, doi: 10.4108/eai.25-10-2016.2266614. **Candidate's contributions**: reviewed part of the literature; wrote part of the paper.

8. N. Kushwaha, O. P. Vyas, **C. Puliafito** and R. Vyas, "Towards an Ontology development for automated applications in Smart City environment of SmartME Project", *10th EAI International Conference on Performance Evaluation Methodologies and Tools* (*VALUETOOLS 2016*), October 26-28, 2016, Taormina, Italy, doi: 10.4108/eai.25-10-2016.2266527. **Candidate's contributions**: reviewed part of the literature; wrote part of the paper.

Workshop papers

9. **C. Puliafito**, E. Mingozzi, C. Vallati, F. Longo and G. Merlino, "Virtualization and Migration at the Network Edge: An Overview", *4th IEEE International Workshop on Sensors and Smart Cities* (*SSC 2018*) *co-located with the 4th IEEE International Conference on Smart Computing* (*SMARTCOMP 2018*), June 18, 2018, Taormina, Italy, doi: 10.1109/SMARTCOMP.2018.00031. **Candidate's contributions**: reviewed the literature; wrote the paper.

10. **C. Puliafito**, E. Mingozzi and G. Anastasi, "Fog Computing for the Internet of Mobile Things: Issues and Challenges", *3rd IEEE International Workshop on Sen-*

sors and Smart Cities (*SSC 2017*) *co-located with the 3rd IEEE International Conference on Smart Computing* (*SMARTCOMP 2017*), May 29-31, 2017, Hong Kong, China, doi: 10.1109/SMARTCOMP.2017.7947010. **Candidate's contributions**: reviewed the literature; wrote the paper.

11. M. Giacobbe, **C. Puliafito** and M. Scarpa, "The Big Bucket: An IoT Cloud Solution For Smart Waste Management in Smart Cities", *4th International Workshop on CLoud for IoT* (*CLIoT 2016*), September 5, 2016, Vienna, Austria, doi: 10.1007/978-3-319-72125-5_4. **Candidate's contributions**: implemented part of the solution; reviewed part of the literature; wrote part of the paper.

Papers under review

12. **C. Puliafito**, C. Vallati, E. Mingozzi, G. Merlino and F. Longo, "Design and Evaluation of a Fog Platform Supporting Device Mobility Through Container Migration", submitted to *Elsevier Pervasive and Mobile Computing*. **Candidate's contributions**: defined the model; designed the solution; implemented the current solution; devised the methodology; set up the testbed; performed the experiments; analysed the results; reviewed the literature; wrote the paper.

13. **C. Puliafito**, A. Virdis and E. Mingozzi, "The Impact of Container Migration on Fog Services as Perceived by Mobile Things", submitted to *6th IEEE International Conference on Smart Computing* (*SMARTCOMP 2020*), June 22-25, 2020, Bologna, Italy. **Candidate's contributions**: defined the reference architecture and the procedures; implemented the system; devised the methodology; set up the testbed; performed the experiments; reviewed the literature; wrote the paper.

Others

14. R. Di Pietro, M. Giacobbe, **C. Puliafito** and M. Scarpa, "J2CBROKER as a Service: A Service Broker Simulation Tool Integrated in OpenStack Environment", chapter in *Systems Modeling: Methodologies and Tools*, pages 269-285, October 2018, doi: 10.1007/978-3-319-92378-9_17. **Candidate's contributions**: reviewed part of the literature; wrote part of the paper.

15. **C. Puliafito**, C. Vallati, E. Mingozzi, G. Merlino and F. Longo, "Evaluation of a Fog Computing Platform Supporting Device Mobility in Smart Cities", *5th Italian Conference on ICT for Smart Cities and Communities* (*I-CiTies 2019*), September 18-20, 2019, Pisa, Italy. **Candidate's contributions**: defined the model; designed the solution; implemented the current solution; devised the methodology; set up the testbed; performed the experiments; analysed the results; reviewed the literature; wrote the paper.

16. **C. Puliafito**, E. Mingozzi, C. Vallati, F. Longo and G. Merlino, "Supporting Things Mobility in Fog-enabled Smart Cities", *4th Italian Conference on ICT for Smart Cities and Communities* (*I-CiTies 2018*), September 19-21, 2018, L'Aquila, Italy. **Candidate's contributions**: defined the model; designed the solution; reviewed the literature; wrote the paper.

17. **C. Puliafito**, E. Mingozzi and G. Anastasi, "Companion Fog Computing for Mobile Things in a Smart City", *3rd Italian Conference on ICT for Smart Cities and Communities* (*I-CiTies 2017*), September 27-29, 2017, Bari, Italy. **Candidate's contributions**: reviewed the literature; wrote the paper.

18. M. Giacobbe, **C. Puliafito** and M. Scarpa, "Smart Waste and Recycling Systems: The Big Bucket Solution", *2nd Italian Conference on ICT for Smart Cities and Communities* (*I-CiTies 2016*), September 29-30, 2016, Benevento, Italy. **Candidate's contributions**: implemented part of the solution; reviewed part of the literature; wrote part of the paper.

Bibliography

A. Varkockova (2018). What's the difference between runC, containerd, Docker? Well, I asked myself the exact same question... Last accessed: 06 March 2019.

Aazam, M., St-Hilaire, M., Lung, C., Lambadaris, I., and Huh, E. (2018). IoT resource estimation challenges and modeling in Fog. In *Fog Computing in the Internet of Things: Intelligence at the Edge*, pages 17–31. Springer International Publishing.

Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M., and Steggles, P. (1999). Towards a better understanding of context and context-awareness. In *International Symposium on Handeld and Ubiquitous Computing* (*HUC*), pages 304–307.

Ahmad, M., Amin, M. B., Hussain, S., Kang, B. H., Cheong, T., and Lee, S. (2016). Health Fog: a novel framework for health and wellness applications. *Springer Journal of Supercomputing*, 72(10):3677–3695.

Ai, Y., Peng, M., and Zhang, K. (2018). Edge Cloud Computing technologies for Internet of Things: a primer. *Digital Communications and Networks*, 4(2):77–86.

Akamai (2016). Akamai's state of the Internet - Q3 2016 report. Technical Report 3. Last accessed: 21 February 2019.

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., and Ayyash, M. (2015). Internet of Things: a survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, 17(4):2347–2376.

Ali, S. and Ghazal, M. (2017). Real-time Heart Attack Mobile Detection Service (RHAMDS): an IoT use case for Software Defined Networks. In *30th IEEE Canadian Conference on Electrical and Computer Engineering* (*CCECE*), pages 1–6.

Amazon (2017). AWS Network Latency Map. `https://datapath.io/resources/blog/aws-network-latency-map/`. Last accessed: 10 October 2019.

Amazon (2018). AWS Greengrass. `https://aws.amazon.com/greengrass/`. Last accessed: 10 October 2019.

Amiot, E. (2015). The Internet of Things: disrupting traditional business models. Technical report, Oliver Wyman. Last accessed: 10 October 2019.

Association, I. S. (2018). 1934 - IEEE approved draft standard for adoption of OpenFog Reference Architecture for Fog Computing. `https://standards.ieee.org/standard/1934-2018.html`. Last accessed: 10 October 2019.

Asvija, B., Eswari, R., and Bijoy, M. B. (2019). Security in hardware assisted virtualization for Cloud Computing - State of the art issues and challenges. *Computer Networks*, 151:68–92.

Atlam, H. F., Walters, R. J., and Wills, G. B. (2018). Fog Computing and the Internet of Things: a review. *Journal of Big Data and Cognitive Computing*, 10(2).

Atzori, L., Iera, A., and Morabito, G. (2010). The Internet of Things: a survey. *Computer Networks*, 54(15):2787–2805.

Baktir, A. C., Ozgovde, A., and Ersoy, C. (2017). How can Edge Computing benefit from Software-Defined Networking: a survey, use cases, and future directions. *IEEE Communications Surveys & Tutorials*, 19(4):2359–2391.

Bao, W., Yuan, D., Yang, Z., Wang, S., Li, W., Zhou, B. B., and Zomaya, A. Y. (2017). Follow Me Fog: toward seamless handover timing schemes in a Fog Computing environment. *IEEE Communications Magazine*, 55(11):72–78.

Bao, W., Yuan, D., Yang, Z., Wang, S., Zhou, B., Adams, S., and Zomaya, A. (2018). sFog: Seamless Fog Computing environment for mobile IoT applications. In *ACM 21st International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems* (*MSWIM*), pages 127–136.

Behrisch, M., Bieker, L., Erdmann, J., and Krajzewicz, D. (2011). SUMO - Simulation of Urban Mobility: An overview. In *3rd International Conference on Advances in System Simulation* (*SIMUL*).

Beligianni, F., Alamaniotis, M., Fevgas, A., Tsompanopoulou, P., Bozanis, P., and Tsoukalas, L. H. (2016). An Internet of Things architecture for preserving privacy of energy consumption. In *Mediterranean Conference on Power Generation, Transmission, Distribution and Energy Conversion* (*MedPower*), pages 1–7.

Bellavista, P. and Zanni, A. (2017). Feasibility of Fog Computing deployment based on Docker containerization over RaspberryPi. In *18th International Conference on Distributed Computing and Networking* (*ICDCN*).

Bellavista, P., Zanni, A., and Solimando, M. (2017). A migration-enhanced Edge Computing support for mobile devices in hostile environments. In *13th International Wireless Communications and Mobile Computing Conference* (*IWCMC*), pages 957–962.

Bittencourt, L. F., Lopes, M. M., Petri, I., and Rana, O. F. (2015). Towards virtual machine migration in Fog Computing. In *10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, pages 1–8.

Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. (2012). Fog computing and its role in the Internet of Things. In *1st Workshop on Mobile Cloud Computing (MCC)*, pages 13–16.

Bormann, C. (2014). CoAP - RFC 7252 Constrained Application Protocol. Last accessed: 24 April 2019.

Botta, A., de Donato, W., Persico, V., and Pescapè, A. (2016). Integration of cloud computing and Internet of Things: a survey. *Future Generation Computer Systems*, 56:684–700.

Brasser, F., Rasmussen, K. B., Sadeghi, A. R., and Tsudik, G. (2016). Remote attestation for low-end embedded devices: the prover's perspective. In *53rd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6.

Breivold, H. P. and Sandstrom, K. (2015). Internet of Things for industrial automation - challenges and technical solutions. In *IEEE International Conference on Data Science and Data Intensive Systems (DSDIS)*, pages 532–539.

Brennand, C. A. R. L., da Cunha, F. D., Maia, G., Cerqueira, E., Loureiro, A. A. F., and Villas, L. A. (2016). FOX: a traffic management system of computer-based vehicles FOG. In *21st IEEE Symposium on Computers and Communications (ISCC)*, pages 982–987.

Brito, M. S. D., Hoque, S., Magedanz, T., Steinke, R., Willner, A., Nehls, D., Keils, O., and Schreiner, F. (2017). A service orchestration architecture for Fog-enabled infrastructures. In *2nd International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 127–132.

Brogi, A. and Forti, S. (2017a). QoS-aware deployment of IoT applications through the Fog. *IEEE Internet of Things Journal*, 4(5):1185–1192.

Brogi, A. and Forti, S. (2017b). QoS-aware deployment of IoT applications through the Fog. *IEEE Internet of Things Journal*, 4(5):1185–1192.

Brogi, A., Forti, S., and Ibrahim, A. (2017). How to best deploy your Fog applications, probably. In *IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, pages 105–114.

Bruneo, D., Distefano, S., Longo, F., Merlino, G., Puliafito, A., D'Amico, V., Sapienza, M., and Torrisi, G. (2016). Stack4Things as a Fog Computing platform for Smart City applications. In *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 848–853.

Bruschi, R., Lago, P., Lamanna, G., Lombardo, C., and Mangialardi, S. (2016). OpenVolcano: an open-source software platform for Fog Computing. In *28th International Teletraffic Congress* (*ITC 28*), pages 22–27.

Brzoza-Woch, R., Konieczny, M., Nawrocki, P., Szydlo, T., and Zielinski, K. (2016). Embedded systems in the application of Fog Computing - levee monitoring use case. In *11th IEEE Symposium on Industrial Embedded Systems* (*SIES*), pages 1–6.

Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., and Buyya, R. (2011). CloudSim: A toolkit for modeling and simulation of Cloud Computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, 41(1):23–50.

Cao, Y., Chen, S., Hou, P., and Brown, D. (2015). FAST: a Fog Computing assisted distributed analytics system to monitor fall for stroke mitigation. In *IEEE International Conference on Networking, Architecture and Storage* (*NAS*), pages 2–11.

Celesti, A., Fazio, M., Longo, F., Merlino, G., and Puliafito, A. (2017). Secure registration and remote attestation of IoT devices joining the Cloud: the Stack4Things case of study. In *Security and privacy in Cyber-Physical Systems*, chapter 7, pages 137–156. Wiley-Blackwell.

Chen, M., Mao, S., and Liu, Y. (2014). Big Data: a survey. *Mobile Networks and Applications*, 19(2):171–209.

Chen, N., Chen, Y., You, Y., Ling, H., Liang, P., and Zimmermann, R. (2016). Dynamic urban surveillance video stream processing using Fog Computing. In *2nd IEEE International Conference on Multimedia Big Data* (*BigMM*), pages 105–112.

Chen, Y., Leong, H. V., Xu, M., Cao, J., Chan, K. C. C., and Chan, A. T. S. (2006). Innetwork data processing for Wireless Sensor Networks. In *7th International Conference on Mobile Data Management* (*MDM*), pages 26–26.

Cheng, B., Solmaz, G., Cirillo, F., Kovacs, E., Terasawa, K., and Kitazawa, A. (2018). FogFlow: easy programming of IoT services over Cloud and edges for Smart Cities. *IEEE Internet of Things Journal*, 5(2):696–707.

Chiang, M. and Zhang, T. (2016). Fog and IoT: an overview of research opportunities. *IEEE Internet of Things Journal*, 3(6):854–864.

Ciftcioglu, E. N., Chan, K. S., Urgaonkar, R., Wang, S., and He, T. (2015). Security-aware service migration for tactical mobile micro-Clouds. In *IEEE Military Communications Conference* (*MILCOM*), pages 1058–1063.

Cisco (2015). Fog Computing and the Internet of Things: extend the Cloud to where the things are. Technical report. Last accessed: 10 October 2019.

Cisco (2018a). Cisco 800 Series Industrial Integrated Services Routers. https://www.cisco.com/c/en/us/products/routers/800-series-industrial-routers/index.html. Last accessed: 10 October 2019.

Cisco (2018b). Cisco IOx. https://www.cisco.com/c/en/us/products/cloud-systems-management/iox/index.html. Last accessed: 10 October 2019.

Cisco (2018c). Compute modules for the Cisco 1000 Series Connected Grid Routers. https://www.cisco.com/c/en/us/products/collateral/routers/1000-series-connected-grid-routers/datasheet-c78-739683.html. Last accessed: 10 October 2019.

Codeca, L., Frank, R., and Engel, T. (2015). Luxembourg SUMO Traffic (LuST) scenario: 24 hours of mobility for vehicular networking research. In *Vehicular Networking Conference (VNC), 2015 IEEE*, pages 1–8.

Consortium, O. (2017). OpenFog Reference Architecture for Fog Computing. https://www.iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf. Last accessed: 10 October 2019.

Consortium, O. (2018a). OpenFog Consortium - member companies. https://www.iiconsortium.org/members.htm. Last accessed: 10 October 2019.

Consortium, O. (2018b). Top 10 myths of fog computing. https://www.openfogconsortium.org/top-10-myths-of-fog-computing/. Last accessed: 03 July 2018.

D. J. Walsh (2018). A history of low-level Linux container runtimes. Last accessed: 06 March 2019.

Dastjerdi, A. V., Gupta, H., Calheiros, R. N., Ghosh, S. K., and Buyya, R. (2016). Fog Computing: principles, architectures, and applications.

Dautov, R., Distefano, S., Merlino, G., Bruneo, D., Longo, F., and Puliafito, A. (2017). Towards a global intelligent surveillance system. In *11th International Conference on Distributed Smart Cameras (ICDSC)*, pages 119–124.

de Brito, M., Hoque, S., Steinke, R., Willner, A., and Magedanz, T. (2017). Application of the Fog Computing paradigm to Smart Factories and Cyber-Physical Systems. *Transactions on Emerging Telecommunications Technologies*, 29(4):1–14.

Delicato, F. C., Pires, P. F., and Batista, T. (2017). The resource management challenge in IoT. In *Resource Management for Internet of Things*, pages 7–18. Springer International Publishing.

Dell Technologies (2018). Gateways & embedded computing. http://www.dell.com/en-us/work/shop/cty/sc/gateways-embedded-pcs?stp_redir=false&~ck=mn. Last accessed: 10 October 2019.

Desai, A., Oza, R., Sharma, P., and Patel, B. (2013). Hypervisor: A survey on concepts and taxonomy. *International Journal of Innovative Technology and Exploring Engineering*, 2(3):222–225.

Dhingra, N. (2014). Challenges, limitation and security issues on Mobile Computing. *International Journal of Current Engineering and Technology*, 4(5):3459–3462.

Dimitrakopoulos, G. and Demestichas, P. (2010). Systems based on cognitive networking principles and management functionality. *IEEE Vehicular Technology Magazine*, 5(1):77–84.

Docker (2017a). About storage drivers. Last accessed: 07 March 2019.

Docker (2017b). Docker overview. Last accessed: 06 March 2019.

Drath, R. and Horch, A. (2014). Industrie 4.0: hit or hype? [Industry Forum]. *IEEE Industrial Electronics Magazine*, 8(2):56–58.

Dupont, C., Giaffreda, R., and Capra, L. (2017). Edge Computing in IoT context: horizontal and vertical Linux Container migration. In *Global Internet of Things Summit (GIoTS)*, pages 1–4.

Dutta, J. and Roy, S. (2017). IoT-Fog-Cloud based architecture for Smart City: prototype of a Smart Building. In *7th International Conference on Cloud Computing, Data Science & Engineering (CONFLUENCE)*, pages 237–242.

Eivy, A. (2017). Be wary of the economics of "Serverless" Cloud Computing. *IEEE Cloud Computing*, 4(2):6–12.

Elmisery, A. M., Rho, S., and Botvich, D. (2016). A Fog based middleware for automated compliance with OECD privacy principles in Internet of Healthcare Things. *IEEE Access*, 4:8418–8441.

Ericsson (2017). From healthcare to homecare - The critical role of 5G in healthcare transformation. Technical report. Last accessed: 21 February 2019.

ETSI (2016). Mobile Edge Computing (MEC): Technical requirements. Technical report. ETSI GS MEC 002 V1.1.1, last accessed: 21 February 2019.

ETSI (2017a). ETSI and OpenFog Consortium collaborate on Fog and Edge applications. http://www.etsi.org/news-events/news/1216-2017-09-news-etsi-and-openfog-consortium-collaborate-on-fog-and-edge-applications. Last accessed: 10 October 2019.

ETSI (2017b). ETSI Multi-access Edge Computing starts second phase and renews leadership team. http://www.etsi.org/news-events/news/1180-2017-03-news-etsi-multi-access-edge-computing-starts-second-phase-and-renews-leadership-team. Last accessed: 10 October 2019.

ETSI (2018). Multi-access Edge Computing. http://www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing. Last accessed: 10 October 2019.

Fan, C., Wu, Z., Chang, C., and Yuan, S. M. (2016). Web resource cacheable edge device in Fog Computing. In *15th International Symposium on Parallel and Distributed Computing* (*ISPDC*), pages 432–439.

Fang, X., Misra, S., Xue, G., and Yang, D. (2012). Smart Grid - the new and improved power grid: a survey. *IEEE Communications Surveys & Tutorials*, 14(4):944–980.

Farahani, B., Firouzi, F., Chang, V., Badaroglu, M., Constant, N., and Mankodiya, K. (2017). Towards Fog-driven IoT eHealth: promises and challenges of IoT in medicine and healthcare. *Future Generation Computer Systems*, 78(2):659–676.

Farris, I., Taleb, T., Iera, A., and Flinck, H. (2017). Lightweight service replication for ultra-short latency applications in Mobile Edge networks. In *IEEE International Conference on Communications* (*ICC*), pages 1–6.

Fernando, N., Loke, S. W., and Rahayu, W. (2013). Mobile Cloud Computing: a survey. *Future Generation Computer Systems*, 29(1):84–106.

Fernández-Caramés, T. M., Fraga-Lamas, P., Suárez-Albela, M., and Vilar-Montesinos, M. (2018). A Fog Computing and cloudlet based augmented reality system for the Industry 4.0 shipyard. *MDPI Sensors*, 18(6):1–18.

FogHorn (2018). FogHorn Lightning. https://www.foghorn.io/products/. Last accessed: 10 October 2019.

Foundation, L. (2018). EdgeX Foundry. https://www.edgexfoundry.org/. Last accessed: 10 October 2019.

Fratu, O., Pena, C., Craciunescu, R., and Halunga, S. (2015). Fog Computing system for monitoring mild dementia and COPD patients - romanian case study. In *12th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services* (*TELSIKS*), pages 123–128.

Ge, X., Li, Z., and Li, S. (2017). 5G Software Defined Vehicular Networks. *IEEE Communications Magazine*, 55(7):87–93.

GmbH, A. I. S. E. (2018). macchina.io. http://macchina.io. Last accessed: 10 October 2019.

Gonçalves, D., Velasquez, K., Curado, M., Bittencourt, L., and Madeira, E. (2018). Proactive Virtual Machine migration in Fog environments. In *IEEE Symposium on Computers and Communications* (*ISCC*), pages 742–745.

Govindaraj, K. and Artemenko, A. (2018). Container live migration for latency critical industrial applications on Edge Computing. In *IEEE 23rd International Conference on Emerging Technologies and Factory Automation* (*ETFA*), volume 1, pages 83–90.

Gupta, H., Vahid Dastjerdi, A., Ghosh, S. K., and Buyya, R. (2017). iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. *Software: Practice and Experience*, 47(9):1275–1296.

Ha, K., Abe, Y., Eiszler, T., Chen, Z., Hu, W., Amos, B., Upadhyaya, R., Pillai, P., and Satyanarayanan, M. (2017). You can teach elephants to dance: agile VM handoff for Edge Computing. In *2nd ACM/IEEE Symposium on Edge Computing* (*SEC*).

Ha, K. and Satyanarayanan, M. (2015). OpenStack++ for Cloudlet deployment. Technical report, CMU School of Computer Science. CMU-CS-15-123, Last accessed: 10 October 2019.

Habib, I. (2008). Virtualization with KVM. *Linux J.*, 2008(166). Available online: `http://dl.acm.org/citation.cfm?id=1344209.1344217`. Last accessed: 15 January 2019.

Han, W. and Xiao, Y. (2016). Big Data security analytic for Smart Grid with Fog nodes. In *9th International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage* (*SpaCCS*), pages 59–69.

He, X., Ren, Z., Shi, C., and Fang, J. (2016). A novel load balancing strategy of Software-Defined Cloud/Fog networking in the Internet of Vehicles. *China Communications*, 13:140–149.

Hedge, Z. (2017). Fog computing global market will exceed US$18 billion by 2022. `https://www.iot-now.com/2017/10/31/70085-fog-computing-global-market-will-exceed-us18-billion-2022/`. Last accessed: 10 October 2019.

Hoque, S., d. Brito, M. S., Willner, A., Keil, O., and Magedanz, T. (2017). Towards container orchestration in Fog Computing infrastructures. In *41st IEEE International Computer Software and Applications Conference* (*COMPSAC*), pages 294–299.

Horwitz, L. (2017). Edge computing technology key to future enterprise, Gartner says. `https://www.cisco.com/c/en/us/solutions/internet-of-things/edge-computing-technology-gartner.html`. Last accessed: 10 October 2019.

Hou, X., Li, Y., Chen, M., Wu, D., Jin, D., and Chen, S. (2016). Vehicular Fog Computing: a viewpoint of vehicles as the infrastructures. *IEEE Transactions on Vehicular Technology*, 65(6):3860–3873.

Hu, Y. C., Patel, M., Sabella, D., Sprecher, N., and Young, V. (2015). Mobile Edge Computing: a key technology towards 5G. `http://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp11_mec_a_key_technology_towards_5g.pdf`. ETSI White Paper No. 11. Last accessed: 10 October 2019.

Huawei (2017). 5G unlocks a world of opportunities - Top ten 5G use cases. Technical report. Last accessed: 21 February 2019.

IBM (2018). IBM Watson IoT platform. `https://www.ibm.com/cloud/internet-of-things`. Last accessed: 10 October 2019.

IFRC (2016). World disasters report 2016. Technical report. Last accessed: 10 October 2019.

Intel (2018). Intel Edison development platform. `https://www.intel.com/content/dam/support/us/en/documents/edison/sb/edison_pb_331179002.pdf`. Last accessed: 10 October 2019.

Ismail, B. I., Goortani, E. M., Ab Karim, M. B., Tat, W. M., Setapa, S., Luke, J. Y., and Hoe, O. H. (2015). Evaluation of Docker as Edge Computing platform. In *IEEE Conference on Open Systems (ICOS)*, pages 130–135.

Jiang, Y., Huang, Z., and Tsang, D. H. K. (2018). Challenges and solutions in Fog Computing orchestration. *IEEE Network*, 32(3):122–129.

Kai, K., Cong, W., and Tao, L. (2016). Fog Computing for Vehicular Ad-hoc Networks: paradigms, scenarios, and issues. *Journal of China Universities of Posts and Telecommunications*, 23(2):56–65.

Kakakhel, S. R. U., Mukkala, L., Westerlund, T., and Plosila, J. (2018). Virtualization at the network edge: A technology perspective. In *IEEE 3rd International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 87–92.

Kanellos, M. (2016). How to keep the Internet of Things from breaking the Internet. `https://www.forbes.com/sites/michaelkanellos/2016/06/16/how-to-keep-the-internet-of-things-from-breaking-the-internet/#5d210e2e6a7c`. Last accessed: 10 October 2019.

Karim, M. B. A., Ismail, B. I., Tat, W. M., Goortani, E. M., Setapa, S., Luke, J. Y., and Ong, H. (2016). Extending cloud resources to the edge: possible scenarios, challenges, and experiments. In *International Conference on Cloud Computing Research and Innovations (ICCCRI)*, pages 78–85.

Khan, S., Parkinson, S., and Qin, Y. (2017). Fog Computing security: a review of current applications and security solutions. *Journal of Cloud Computing*, 6(1).

Kim, O. T. T., Tri, N. D., Nguyen, V. D., Tran, N. H., and Hong, C. S. (2015). A shared parking model in vehicular network using Fog and Cloud environment. In *17th Asia-Pacific Network Operations and Management Symposium* (*APNOMS*), pages 321–326.

Kraemer, F. A., Braten, A. E., Tamkittikhun, N., and Palma, D. (2017). Fog Computing in healthcare - a review and discussion. *IEEE Access*, 5:9206–9222.

Ksentini, A., Taleb, T., and Chen, M. (2014). A Markov Decision Process-based service migration procedure for Follow Me Cloud. In *IEEE International Conference on Communications* (*ICC*), pages 1350–1354.

Lai, C., Song, D., Hwang, R., and Lai, Y. (2016). A QoS-aware streaming service over Fog Computing infrastructures. In *Digital Media Industry & Academic Forum* (*DMIAF*), pages 94–98.

Lebre, A., Pastor, J., Simonet, A., and Desprez, F. (2017). Revising OpenStack to operate Fog/Edge Computing infrastructures. In *IEEE International Conference on Cloud Engineering* (*IC2E*), pages 138–148.

Lera, I., Guerrero, C., and Juiz, C. (2019). YAFS: A simulator for IoT scenarios in Fog Computing. *IEEE Access*, 7:91745–91758.

Liu, J., Wan, J., Jia, D., Zeng, B., Li, D., Hsu, C., and Chen, H. (2017a). High-efficiency urban-traffic management in context-aware computing and 5G communication. *IEEE Communications Magazine*, 55(1):34–40.

Liu, J., Wan, J., Zeng, B., Wang, Q., Song, H., and Qiu, M. (2017b). A scalable and quick-response Software Defined Vehicular Network assisted by Mobile Edge Computing. *IEEE Communications Magazine*, 55(7):94–100.

Liu, P., Willis, D., and Banerjee, S. (2016a). ParaDrop: enabling lightweight multitenancy at the network extreme edge. In *IEEE/ACM Symposium on Edge Computing* (*SEC*), pages 1–13.

Liu, P., Willis, D., and Banerjee, S. (2016b). ParaDrop: enabling lightweight multitenancy at the network's extreme edge. In *IEEE/ACM Symposium on Edge Computing* (*SEC*), pages 1–13.

LLC, G. (2018). Android kernel configs. Last accessed: 21 February 2019.

Longbottom, R. (2010). Roy Longbottom's PC benchmark collection - Linux Multi-Threading benchmarks. Last accessed: 5 April 2019.

Longbottom, R. (2013a). MP Dhrystone benchmarks - MP-DHRY, MP-DHRYPiA7. Last accessed: 8 April 2019.

Longbottom, R. (2013b). Roy Longbottom's Raspberry Pi, Pi 2 and Pi 3 benchmarks. Last accessed: 8 April 2019.

Longo, F., Bruneo, D., Distefano, S., Merlino, G., and Puliafito, A. (2017). Stack4Things: a Sensing-and-Actuation-as-a-Service framework for IoT and Cloud integration. *Annals of Telecommunications*, 72(1):53–70.

Lopes, M. M., Higashino, W. A., Capretz, M. A., and Bittencourt, L. F. (2017). MyiFogSim: A simulator for Virtual Machine migration in Fog Computing. In *6th International Workshop on Clouds and (eScience) Applications Management (CloudAM). Companion Proceedings of the 10th International Conference on Utility and Cloud Computing*, pages 47–52.

LP, H. P. E. D. (2018a). HPE Edgeline EL1000 and EL4000. `http://download.power-point.net/client/HPEIoTFanTurbineConditionMonitoringSolution/EdgelineProducts.pdf`. Last accessed: 10 October 2019.

LP, H. P. E. D. (2018b). HPE GL20 IoT Gateway. `https://www.hpe.com/us/en/product-catalog/servers/edgeline-systems/pip.hpe-edgeline-el20-intelligent-gateway.1008670391.html`. Last accessed: 10 October 2019.

Ma, L., Yi, S., Carter, N., and Li, Q. (2018). Efficient live migration of edge services leveraging container layered storage. *IEEE Transactions on Mobile Computing*. Early Access.

Mach, P. and Becvar, Z. (2017). Mobile Edge Computing: a survey on architecture and computation offloading. *IEEE Communications Surveys & Tutorials*, 19(3):1628–1656.

Mahmud, R., Kotagiri, R., and Buyya, R. (2017). Fog Computing: a taxonomy, survey and future directions. In *Internet of Everything: Algorithms, Methodologies, Technologies and Perspectives*, pages 103–130. Springer Singapore.

Mahmud, R., Ramamohanarao, K., and Buyya, R. (2018a). Latency-aware application module management for Fog Computing environments. *ACM Transactions on Internet Technology*. Early Access.

Mahmud, R., Srirama, S. N., Ramamohanarao, K., and Buyya, R. (2018b). Quality of Experience (QoE)-aware placement of applications in Fog Computing environments. *Journal of Parallel and Distributed Computing*. Early Access.

Manyika, J., Chui, M., Bisson, P., Woetzel, J., Dobbs, R., Bughin, J., and Aharon, D. (2015). The Internet of Things: mapping the value beyond the hype. Technical report, McKinsey Global Institute. Last accessed: 10 October 2019.

Mao, Y., You, C., Zhang, J., Huang, K., and Letaief, K. B. (2017). A survey on Mobile Edge Computing: the communication perspective. *IEEE Communications Surveys & Tutorials*, 19(4):2322–2358.

Marìn-Tordera, E., Masip-Bruin, X., Garcìa-Alminana, J., Jukan, A., Ren, G.-J., and Zhu, J. (2017). Do we all really know what a Fog node is ? Current trends towards an open definition. *Computer Communications*, 109:117–130.

MarketsandMarkets (2016). Fog Computing market by offering (hardware, software), application (building & home automation, smart energy, smart manufacturing, transportation & logistics, connected health, security & emergencies), and geography - global forecast to 2022. Technical report. Last accessed: 10 October 2019.

Marr, D. T., Binns, F., Hill, D. L., Hinton, G., Koufaty, D. A., Miller, J. A., and Upton, M. (2002). Hyper-Threading technology architecture and microarchitecture. *Intel Technology Journal*, 6(1). Last accessed: 5 April 2019.

Masip-Bruin, X., Marìn-Tordera, E., Gòmez, A., Barbosa, V., and Alonso, A. (2016). Will it be Cloud or will it be Fog ? F2C, a novel flagship computing paradigm for highly demanding services. In *Future Technologies Conference* (*FTC*), pages 1129–1136.

Mayer, R., Gupta, H., Saurez, E., and Ramachandran, U. (2017). The Fog makes sense: enabling social sensing services with limited Internet connectivity. In *2nd International Workshop on Social Sensing* (*SocialSens*), pages 61–66.

Mei, B., Li, R., Cheng, W., Yu, J., and Cheng, X. (2017). Ultraviolet radiation measurement via smart devices. *IEEE Internet of Things Journal*, 4(4):934–944.

Mell, P. and Grance, T. (2011). The NIST definition of Cloud Computing. `https://www.nist.gov/sites/default/files/documents/itl/cloud/cloud-def-v15.pdf`. NIST Special Publication 800-145. Last accessed: 10 October 2019.

Microsoft (2018). Microsoft Azure IoT Edge. `https://azure.microsoft.com/en-us/services/iot-edge/`. Last accessed: 10 October 2019.

Monteiro, A., Dubey, H., Mahler, L., Yang, Q., and Mankodiya, K. (2016). FIT: a Fog Computing device for speech tele-treatments. In *2nd IEEE International Conference on Smart Computing* (*SMARTCOMP*), pages 1–3.

Montero, R. S., Rojas, E., Carrillo, A. A., and Llorente, I. M. (2017). Extending the Cloud to the network edge. *Computer*, 50(4):91–95.

Morabito, R. (2017). Virtualization on Internet of Things edge devices with container technologies: A performance evaluation. *IEEE Access*, 5:8835–8850.

Morabito, R., Cozzolino, V., Ding, A. Y., Beijar, N., and Ott, J. (2018). Consolidate IoT Edge Computing with lightweight virtualization. *IEEE Network*, 32(1):102–111.

Motlagh, N. H., Bagaa, M., and Taleb, T. (2017). UAV-based IoT platform: a crowd surveillance use case. *IEEE Communications Magazine*, 55(2):128–134.

Mouradian, C., Naboulsi, D., Yangui, S., Glitho, R. H., Morrow, M. J., and Polakos, P. A. (2018). A comprehensive survey on Fog Computing: state-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 20(1):416–464.

Mukherjee, M., Matam, R., Shu, L., Maglaras, L., Ferrag, M. A., Choudhury, N., and Kumar, V. (2017). Security and privacy in Fog Computing: challenges. *IEEE Access*, 5:19293–19304.

Nadgowda, S., Suneja, S., Bila, N., and Isci, C. (2017). Voyager: Complete container state migration. In *IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2137–2142.

Nan, Y., Li, W., Bao, W., Delicato, F. C., Pires, P. F., Dou, Y., and Zomaya, A. Y. (2017). Adaptive energy-aware computation offloading for Cloud of Things systems. *IEEE Access*, 5:23947–23957.

Nazmudeen, M. S. H., Wan, A. T., and Buhari, S. M. (2016). Improved throughput for Power Line Communication (PLC) for Smart Meters using Fog Computing based data aggregation approach. In *2nd IEEE International Smart Cities Conference: Improving the Citizens Quality of Life (ISC2)*, pages 1–4.

Nebbiolo (2017). Fog Computing: keystone of Industrial IoT and Industry 4.0. Technical report. Last accessed: 10 October 2019.

Nebbiolo (2018a). Nebbiolo Fog Computing platform. `https://www.nebbiolo.tech/`. Last accessed: 10 October 2019.

Nebbiolo (2018b). Nebbiolo fogNode. `https://www.nebbiolo.tech/wp-content/uploads/fogNode-OVERVIEW-rev3.pdf`. Last accessed: 10 October 2019.

Nebbiolo (2018c). Nebbiolo SDK. `https://docs.nebbiolo.io/latest/sdk-guide/services/installSDK/`. Last accessed: 10 October 2019.

Ni, J., Zhang, K., Lin, X., and Shen, X. S. (2018). Securing Fog Computing for Internet of Things applications: challenges and solutions. *IEEE Communications Surveys & Tutorials*, 20(1):601–628.

Okay, F. Y. and Ozdemir, S. (2016). A Fog Computing based Smart Grid model. In *International Symposium on Networks, Computers and Communications* (*ISNCC*), pages 1–6.

Open Container Initiative (2017). What has Docker done to help create this foundation? Last accessed: 06 March 2019.

OpenEdgeComputing (2018). OpenEdgeComputing - home page. `http://openedgecomputing.org`. Last accessed: 10 October 2019.

OpenStack (2013). OpenStack foundation - Home page. Last accessed: 21 February 2019.

OpenStack (2018a). Fog Edge Massively Distributed Clouds Group of Interest - home page. `https://wiki.openstack.org/wiki/Fog_Edge_Massively_Distributed_Clouds`. Last accessed: 10 October 2019.

OpenStack (2018b). OpenStack Foundation - home page. `https://www.openstack.org/foundation`. Last accessed: 10 October 2019.

OpenVolcano (2018). OpenVolcano - home page. `http://openvolcano.org/`. Last accessed: 10 October 2019.

Pathan, A. K. and Buyya, R. (2007). A taxonomy and survey of Content Delivery Networks. Technical report. Last accessed: 10 October 2019.

Peralta, G., Iglesias-Urkia, M., Barcelo, M., Gomez, R., Moran, A., and Bilbao, J. (2017). Fog Computing based efficient IoT scheme for the Industry 4.0. In *IEEE International Workshop of Electronics, Control, Measurement, Signals and their Application to Mechatronics* (*ECMSM*), pages 1–6.

Perera, C., Qin, Y., Estrella, J. C., Reiff-Marganiec, S., and Vasilakos, A. V. (2017). Fog Computing for sustainable Smart Cities: a survey. *ACM Computing Surveys*, 50(3).

Perera, C., Zaslavsky, A., Christen, P., and Georgakopoulos, D. (2014). Context aware computing for the Internet of Things: a survey. *IEEE Communications Surveys & Tutorials*, 16(1):414–454.

Petri, I., Rana, O. F., Bignell, J., Nepal, S., and Auluck, N. (2017). Incentivising resource sharing in Edge Computing applications. In *International Conference on the Economics of Grids, Clouds, Systems, and Services* (*GECON*, pages 204–215.

Pi, R. (2018). Raspberry Pi 3 Model B+. `https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/`. Last accessed: 10 October 2019.

Plachy, J., Becvar, Z., and Strinati, E. C. (2016). Dynamic resource allocation exploiting mobility prediction in Mobile Edge Computing. In *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications* (*PIMRC*), pages 1–6.

Poggi, A. and Tomaiuolo, M. (2011). Mobile agents: concepts and technologies. In *Handbook of Research on Mobility and Computing: Evolving Technologies and Ubiquitous Impacts. IGI Global*, pages 343–355. IGI Global.

Qayyum, T., Malik, A. W., Khattak, M. A. K., Khalid, O., and Khan, S. U. (2018). FogNetSim++: A toolkit for modelling and simulation of distributed Fog environment. *IEEE Access*, 6:63570–63583.

Qualcomm (2018a). DragonBoard 410c development board. `https://developer.qualcomm.com/hardware/dragonboard-410c`. Last accessed: 10 October 2019.

Qualcomm (2018b). DragonBoard 820c development board. `https://developer.qualcomm.com/hardware/dragonboard-820c`. Last accessed: 10 October 2019.

Rahmani, A. M., Gia, T. N., Negash, B., Anzanpour, A., Azimi, I., Jiang, M., and Liljeberg, P. (2017). Exploiting smart e-Health gateways at the edge of healthcare Internet-of-Things: a Fog Computing approach. *Future Generation Computer Systems*, 78(2):641–658.

Ramalho, F. and Neto, A. (2016). Virtualization at the network edge: a performance comparison. In *17th IEEE International Symposium on A World of Wireless, Mobile and Multimedia Networks* (*WoWMoM*), pages 1–6.

Rauniyar, A., Engelstad, P., Feng, B., and Thanh, D. V. (2016). Crowdsourcing-based disaster management using Fog Computing in Internet of Things paradigm. In *2nd IEEE International Conference on Collaboration and Internet Computing* (*CIC*), pages 490–494.

Rejiba, Z., Masip-Bruin, X., and Marín-Tordera, E. (2019). A survey on mobility-induced service migration in the Fog, Edge, and related computing paradigms. *ACM Comput. Surv.*, 52(5):90:1–90:33.

Research, A. (2015). Data captured by IoT connections to top 1.6 zettabytes in 2020, as analytics evolve from Cloud to Edge. `https://www.abiresearch.com/press/data-captured-by-iot-connections-to-top-16-zettaby/`. Last accessed: 10 October 2019.

Roman, R., Lopez, J., and Mambo, M. (2016). Mobile Edge Computing, Fog et al.: a survey and analysis of security threats and challenges. *Future Generation Computer Systems*, 78(2):680–698.

Sapienza, M., Guardo, E., Cavallo, M., Torre, G. L., Leombruno, G., and Tomarchio, O. (2016). Solving critical events through Mobile Edge Computing: an approach for Smart Cities. In *2nd IEEE International Conference on Smart Computing* (*SMARTCOMP*), pages 1–5.

Sareen, S., Gupta, S. K., and Sood, S. K. (2017). An intelligent and secure system for predicting and preventing Zika virus outbreak using Fog Computing. *Enterprise Information Systems*, 11(9):1436–1456.

Sarkar, S., Chatterjee, S., and Misra, S. (2018). Assessment of the suitability of Fog Computing in the context of Internet of Things. *IEEE Transactions on Cloud Computing*, 6(1):46–59.

Satyanarayanan, M. (2001). Pervasive computing: vision and challenges. *IEEE Personal Communications*, 8(4):10–17.

Satyanarayanan, M. (2017). The emergence of Edge Computing. *Computer*, 50(1):30–39.

Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. (2009). The case for VM-based Cloudlets in Mobile computing. *IEEE Pervasive Computing*, 8(4):14–23.

Satyanarayanan, M., Chen, Z., Ha, K., Hu, W., Richter, W., and Pillai, P. (2014). Cloudlets: at the leading edge of mobile-Cloud convergence. In *6th International Conference on Mobile Computing, Applications and Services* (*MobiCASE*), pages 1–9.

Satyanarayanan, M., Lewis, G., Morris, E., Simanta, S., Boleng, J., and Ha, K. (2013). The role of Cloudlets in hostile environments. *IEEE Pervasive Computing*, 12(4):40–49.

Satyanarayanan, M., Simoens, P., Xiao, Y., Pillai, P., Chen, Z., Ha, K., Hu, W., and Amos, B. (2015). Edge analytics in the Internet of Things. *IEEE Pervasive Computing*, 14(2):24–31.

Saurez, E., Hong, K., Lillethun, D., Ramachandran, U., and Ottenwalder, B. (2016). Incremental deployment and migration of geo-distributed situation awareness applications in the Fog. In *10th ACM International Conference on Distributed and Event-based Systems* (*DEBS*), pages 258–269.

Scarpiniti, M., Baccarelli, E., and Momenzadeh, A. (2019). VirtFogSim: A parallel toolbox for dynamic energy-delay performance testing and optimization of 5G mobile-fog-cloud virtualized platforms. *MDPI Applied Sciences*, 9(6).

Schulz, P., Matthe, M., Klessig, H., Simsek, M., Fettweis, G., Ansari, J., Ashraf, S. A., Almeroth, B., Voigt, J., Riedel, I., Puschmann, A., Mitschele-Thiel, A., Muller, M.,

Elste, T., and Windisch, M. (2017). Latency critical IoT applications in 5G: perspective on the design of radio interface and network architecture. *IEEE Communications Magazine*, 55(2):70–78.

Seitz, A., Johanssen, J. O., Bruegge, B., Loftness, V., Hartkopf, V., and Sturm, M. (2017). A Fog architecture for decentralized decision making in Smart Buildings. In *2nd International Workshop on Science of Smart City Operations and Platforms Engineering (SCOPE)*, pages 34–39.

Shi, W. and Dustdar, S. (2016). The promise of Edge Computing. *Computer*, 49(5):78–81.

Shin, S., Seo, S., Eom, S., Jung, J., and Lee, K. H. (2016). A Pub/Sub-based Fog Computing architecture for Internet-of-Vehicles. In *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 90–93.

Shirazi, S. N., Gouglidis, A., Farshad, A., and Hutchison, D. (2017). The extended Cloud: review and analysis of Mobile Edge Computing and Fog from a security and resilience perspective. *IEEE Journal on Selected Areas in Communications*, 35(11):2586–2595.

Simmhan, Y. (2017). Big Data and Fog Computing.

Skarlat, O., Nardelli, M., Schulte, S., Borkowski, M., and Leitner, P. (2017). Optimized IoT service placement in the Fog. *Service Oriented Computing and Applications*, 11(4):427–443.

Soltesz, S., Pötzl, H., Fiuczynski, M. E., Bavier, A., and Peterson, L. (2007). Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. *ACM SIGOPS Operating Systems Review*, 41(3):275–287.

Sonmez, C., Ozgovde, A., and Ersoy, C. (2018). EdgeCloudSim: An environment for performance evaluation of Edge Computing systems. *Transactions on Emerging Telecommunications Technologies*, 29(11).

Stojmenovic, I., Wen, S., Huang, X., and Luan, H. (2015). An overview of Fog Computing and its security issues. *Concurrency and Computation: Practice and Experience*, 28(10):2991–3005.

Strategy, M. I. . (2017). Cleaning up the industrial IoT Edge. Technical report. Last accessed: 10 October 2019.

Suciu, G., Ularu, E. G., and Craciunescu, R. (2012). Public versus private Cloud adoption - a case study based on open source Cloud platforms. In *20th Telecommunications Forum (TELFOR)*, pages 494–497.

Suto, K., Nishiyama, H., Kato, N., and Huang, C. W. (2015). An energy-efficient and delay-aware wireless computing system for Industrial Wireless Sensor Networks. *IEEE Access*, 3:1026–1035.

Taleb, T. and Ksentini, A. (2013). Follow Me Cloud: interworking distributed Clouds & distributed mobile networks. *IEEE Network*, 27(5):12–19.

Taleb, T., Ksentini, A., and Frangoudis, P. (2016). Follow-Me Cloud: when Cloud services follow mobile users. *IEEE Transactions on Cloud Computing*. Early Access.

Taneja, M. and Davy, A. (2017). Resource aware placement of IoT application modules in Fog-Cloud Computing paradigm. In *IFIP/IEEE Symposium on Integrated Network and Service Management* (*IM*), pages 1222–1228.

Tang, Z., Zhou, X., Zhang, F., Jia, W., and Zhao, W. (2018). Migration modeling and learning algorithms for containers in Fog Computing. *IEEE Transactions on Services Computing*. Early Access.

Tanganelli, G., Vallati, C., and Mingozzi, E. (2017). A Fog-based distributed look-up service for Intelligent Transportation Systems. In *18th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks* (*WoWMoM*), pages 1–6.

Tao, M., Ota, K., and Dong, M. (2017). Foud: integrating Fog and Cloud for 5G-enabled V2G networks. *IEEE Network*, 31(2):8–13.

Technologies, D. (2018a). Dell Edge Device Manager. `http://delliotpartners.com/#!/edgedevicemanager/overview`. Last accessed: 10 October 2019.

Technologies, D. (2018b). Dell Edge Gateway 5000. `http://www.dell.com/en-us/work/shop/gateways-embedded-computing/edge-gateway-5000/spd/dell-edge-gateway-5000/xctoi5000us`. Last accessed: 10 October 2019.

Technologies, D. (2018c). Dell Embedded Box PCs. `http://i.dell.com/sites/doccontent/shared-content/data-sheets/en/Documents/specsheet-dell-embedded-box-PC-3000-5000.pdf`. Last accessed: 10 October 2019.

Tomovic, S., Yoshigoe, K., Maljevic, I., and Radusinovic, I. (2017). Software-Defined Fog network architecture for IoT. *Wireless Personal Communications*, 92(1):181–196.

Truong, N. B., Lee, G. M., and Ghamri-Doudane, Y. (2015). Software Defined Networking-based Vehicular Adhoc Network with Fog Computing. In *IFIP/IEEE International Symposium on Integrated Network Management* (*IM*), pages 1202–1207.

TTTech (2018). MFN 100 Edge Computing device. `https://www.tttech.com/wp-content/uploads/TTTech_MFN-100.pdf`. Last accessed: 10 October 2019.

UC Berkeley (2012). Data classification standard. Last accessed: 21 February 2019.

UC Berkeley (2013). Minimum security standards for electronic information. Last accessed: 21 February 2019.

United Nations, Department of Economic and Social Affairs, Population Division (2014). World urbanization prospects: the 2014 revision, highlights. Technical report. Last accessed: 10 October 2019.

Urgaonkar, R., Wang, S., He, T., Zafer, M., Chan, K., and Leung, K. K. (2015). Dynamic service migration and workload scheduling in Edge-Clouds. *Performance Evaluation*, 91:205–228.

Vallati, C., Virdis, A., Mingozzi, E., and Stea, G. (2016). Mobile-Edge Computing come home - connecting Things in future Smart Homes using LTE device-to-device communications. *IEEE Consumer Electronics Magazine*, 5(4):77–83.

Varghese, B., Wang, N., Nikolopoulos, D. S., and Buyya, R. (2017). Feasibility of Fog Computing.

Wang, N., Varghese, B., Matthaiou, M., and Nikolopoulos, D. S. (2018). ENORM: A framework for Edge NOde Resource Management. *IEEE Transactions on Services Computing*. Early Access.

Wang, S., Urgaonkar, R., Zafer, M., He, T., Chan, K., and Leung, K. K. (2015a). Dynamic service migration in Mobile Edge-Clouds. In *IFIP Networking Conference*, pages 1–9.

Wang, S., Xu, J., Zhang, N., and Liu, Y. (2018). A survey on service migration in Mobile Edge Computing. *IEEE Access*, 6:23511–23528.

Wang, Y., Uehara, T., and Sasaki, R. (2015b). Fog Computing: issues and challenges in security and forensics. In *39th IEEE Conference on Computers, Software and Applications (COMPSAC)*, pages 53–59.

Weinman, J. (2018). The 10 laws of Fogonomics. *IEEE Cloud Computing*, 4(6):8–14.

Wen, Z., Yang, R., Garraghan, P., Lin, T., Xu, J., and Rovatsos, M. (2017). Fog orchestration for Internet of Things services. *IEEE Internet Computing*, 21(2):16–24.

Wu, D., Liu, S., Zhang, L., Terpenny, J., Gao, R. X., Kurfess, T., and Guzzo, J. A. (2017). A Fog Computing-based framework for process monitoring and prognosis in cyber-manufacturing. *Journal of Manufacturing Systems*, 43(1):25–34.

Yaghmaee, M. H., Moghaddassian, M., and Leon-Garcia, A. (2017). Autonomous two-tier Cloud based Demand Side Management approach with microgrid. *IEEE Transactions on Industrial Informatics*, 13(3):1109–1120.

Yan, Y. and Su, W. (2016). A Fog Computing solution for advanced metering infrastructure. In *IEEE/PES Transmission and Distribution Conference and Exposition (T&D)*, pages 1–4.

Yao, H., Bai, C., Zeng, D., Liang, Q., and Fan, Y. (2015). Migrate or not ? Exploring Virtual Machine migration in roadside Cloudlet-based vehicular Cloud. *Concurrency and Computation: Practice and Experience*, 27(18):5780–5792.

Ye, D., Wu, M., Tang, S., and Yu, R. (2016). Scalable Fog Computing with service offloading in bus networks. In *3rd IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)*, pages 247–251.

Yigitoglu, E., Mohamed, M., Liu, L., and Ludwig, H. (2017). Foggy: a framework for continuous automated IoT application deployment in Fog Computing. In *6th IEEE International Conference on AI and Mobile Services (AIMS)*, pages 38–45.

York, R. (2002). Benchmarking in context: Dhrystone. ARM White paper. Last accessed: 29 April 2019.

Yu, W., Liang, F., He, X., Hatcher, W. G., Lu, C., Lin, J., and Yang, X. (2017). A survey on the Edge Computing for the Internet of Things. *IEEE Access*, 6:6900–6919.

Zanni, A., Forsstrom, S., Jennehag, U., and Bellavista, P. (2018). Elastic provisioning of Internet of Things services using Fog Computing: An experience report. In *IEEE 6th International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pages 17–22.

Zao, J. K., Gan, T., You, C., Chung, C., Wang, Y., Mèndez, S. R., Mullen, T., Yu, C., Kothe, C., Hsiao, C., Chu, S., Shieh, C., and Jung, T. (2014). Pervasive brain monitoring and data sharing based on multi-tier distributed computing and linked data technology. *Frontiers in Human Neuroscience*, 8(370).

Zhang, W., Hu, Y., Zhang, Y., and Raychaudhuri, D. (2016). SEGUE: quality of service aware Edge Cloud service migration. In *International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 344–351.

Zhou, J., Cao, Z., Dong, X., and Vasilakos, A. V. (2017). Security and privacy for Cloud-based IoT: challenges. *IEEE Communications Magazine*, 55(1):26–33.

Zhu, C., Tao, J., Pastor, G., Xiao, Y., Ji, Y., Zhou, Q., Li, Y., and Ylä-Jääski, A. (2018). Folo: Latency and quality optimized task allocation in Vehicular Fog Computing. *IEEE Internet of Things Journal*. Early Access.

Zhu, J., Chan, D. S., Prabhu, M. S., Natarajan, P., Hu, H., and Bonomi, F. (2013). Improving Web sites performance using edge servers in Fog Computing architecture. In *7th IEEE International Symposium on Service-Oriented System Engineering (SOSE)*, pages 320–323.