ALMA MATER STUDIORUM, UNIVERSITÀ DI BOLOGNA

DOTTORATO DI RICERCA IN
COMPUTER SCIENCE AND ENGINEERING

CICLO XXXII

SETTORE CONCORSUALE DI AFFERENZA: 09/H1
SETTORE SCIENTIFICO DISCIPLINARE: ING-INF/05

Edge Computing for Extreme Reliability and Scalability

PRESENTATA DA:
DOMENICO SCOTECE

COORDINATORE DOTTORATO:
CHIAR.MO PROF. ING.
DAVIDE SANGIORGI

RELATORE
CHIAR.MO PROF. ING.
PAOLO BELLAVISTA

Esame finale anno 2020

Abstract

The massive number of Internet of Things (IoT) devices and their continuous data collection will lead to a rapid increase in the scale of collected data. Processing all these collected data at the central cloud server is inefficient, and even is unfeasible or unnecessary. Hence, the task of processing the data is pushed to the network edges introducing the concept of Edge Computing. Processing the information closer to the source of data (e.g., on gateways and on edge micro-servers) not only reduces the huge workload of central cloud, also decreases the latency for real-time applications by avoiding the unreliable and unpredictable network latency to communicate with the central cloud.

State-of-the-art solutions in Edge Computing have produced the Multi-access Edge Computing (MEC) and the Fog Computing paradigms. These are enabling the opportunity to have middleboxes either statically or dynamically deployed at network edges, acting as local proxies with virtualized resources for supporting and enhancing service provisioning in edge localities. This makes possible, among the other advantages, better scalability and better reactivity, anytime local control decisions and actuation operations are applicable.

This thesis thoroughly investigates the wide literature concerning the MEC and Fog Computing, provides distinguishing properties of the MEC and Fog paradigms and tries to overcome the main open issues in the field. As a complementing concept to the cloud, MEC and Fog Computing have been identified as promising solutions to manage IoT devices. Fog and MEC are not a mature paradigm yet, and it has a myriad of open issues and very challenging future developments that can help the IoT management. These open issues and challenges are drawing the attention of the research communities, those one that have most relevance are:

- edge heterogeneity;
- handoff management;
- services provisioning;
- computation offloading.

In the literature, few other works have investigated the idea to bring the best of MEC and Fog Computing by integrating them in a unique architecture. It is clear that a kind of standardization or homogenization is needed in order to be able to use the MEC and the Fog paradigms with any type of IoT object. The IoT paradigm is very mobile, the nodes have a high dynamicity and they suddenly join or leave the network. Here, emerges the need for properly managing

the objects network, the node connection loss, new smart node discovery, and the device data migration from a network to another.

Under this foreword, this thesis presents a new architectural model based on the introduction of fully-converged 5G-Enabled Edge (5GEE) architecture, as a core element deployed at the network edge. Our 5GEE enables the combination and joint exploitation of the MEC and Fog capabilities and is able to manage the complexity of IoT devices. The project intends to solve the currently most challenging issues in the field of IoT management in edge-enabled scenarios by proposing novel 5GEE functionalities that help to overcome the most relevant open challenges as stated before. This thesis, first, presents functionality for handoff management. Due to the high mobility nature of IoT and mobile devices, efficient techniques to migrate data from a network to another are needed to preserve a high value of QoS and QoE. More precisely, this thesis presents differentiated approaches for handoff management by leveraging Docker Containers tool as edge-services virtualized technology. These mechanisms leverage services characteristics to enable both application-agnostic and application-aware service migration.

This thesis also overcome the challenge of computation offloading in the edge-enabled scenarios by proposing the Mobile Edge File System (MEFS) a novel application-level distributed filesystem to efficiently support tasks offloading between a mobile node and edge nodes. MEFS runs concurrently on mobile devices, edge devices, and the cloud. Furthermore, this research project has had to face two challenges in the MEFS design such as the Application Portability (handoff management) and Resilience (fault-tolerance management).

In addition, a new approach to services discovery and provisioning at the edge of the network will be presented. This research leverages the flexible nature of 5GEE architecture to provide service discovery functionalities at different layers.

Finally, to evaluate the proposed techniques and mechanisms, some applications are profiled on the Edge Computing platform to measure their required parameters including execution time and resources utilization. These parameters are then used to obtain the network overhead communication between devices and edge nodes as well as battery lifetime and service quality of devices.

Publications

The work did during these three years of my Ph.D., has resulted in 9 publications works:

1. P. Bellavista, L. Foschini and D. Scotece, “Converging Mobile Edge Computing, Fog Computing, and IoT Quality Requirements,” 2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud), Prague, 2017, pp. 313-320.
2. C. Giannelli, P. Bellavista and D. Scotece, “Software Defined Networking for Quality-aware Management of Multi-hop Spontaneous Networks,” 2018 International Conference on Computing, Networking and Communications (ICNC), Maui, HI, 2018, pp. 561-566.
3. P. Bellavista, L. Foschini, D. Scotece, K. Karypidou and P. Chatzimisios, “DRIVE: Discovery seRvice for fully-Integrated 5G enVironmEnt in the IoT,” 2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), Barcelona, 2018, pp. 1-6.
4. P. Bellavista et al., “Design Guidelines for Big Data Gathering in Industry 4.0 Environments,” 2019 IEEE 20th International Symposium on “A World of Wireless, Mobile and Multimedia Networks” (WoWMoM), Washington, DC, USA, 2019, pp. 1-6.
5. D. Scotece, N. R. Paiker, L. Foschini, P. Bellavista, X. Ding and C. Borcea, “MEFS: Mobile Edge File System for Edge-Assisted Mobile Apps,” 2019 IEEE 20th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM), Washington, DC, USA, 2019, pp. 1-9.
6. P. Bellavista, A. Corradi, L. Foschini and D. Scotece, “Differentiated Service/Data Migration for Edge Services Leveraging Container Characteristics,” in IEEE Access, vol. 7, pp. 139746-139758, 2019.
7. P. Bellavista, P. Chatzimisios, L. Foschini, M. Paradisioti and D. Scotece, “A Support Infrastructure for Machine Learning at the Edge in Smart City Surveillance,” 2019 IEEE Symposium on Computers and Communications (ISCC), Barcelona, Spain, 2019, pp. 1189-1194.
8. P. Bellavista, R. Della Penna, L. Foschini, D. Scotece, “Machine Learning for Predictive Diagnostics at the Edge: an IIoT Practical Example,” (ICC 2020) (Accepted).

9. P. Bellavista, A. Corradi, L. Foschini, D. Scotece, “How to Support MEC Service/Data Handoff with Differentiated Granularity: Tradeoffs, Lessons Learnt, and Experimental Validation,” (IEEE Communication Magazine) (Under review).

Contents

List of Tables

List of Figures

1 INTRODUCTION

1.1 Motivation

During the past decade, the Internet of Things (IoT) has revolutionized the ubiquitous computing with a multitude of applications built around the various type of "things". It has resulted in a massive increase in data traffic due to the frequent exchange of information between an enormous number of IoT devices and the cloud. Indeed, Cisco has predicted that about 50 billion devices will be connected to the Internet by 2020 [1]. Figure 1.1 shows the estimated number of IoT devices by the year 2020 with the penetration rate of connected objects. IoT devices cover different sectors including consumer sector, cross-industry, and Industry 4.0. Consumer sector includes the devices which are used by the end users which include tracking and fitness bands, healthcare devices, etc. The cross-industry sector includes the usual and general devices that are being used in different industries including smart home, smart parking, smart city, etc. The last sector, Industry 4.0, includes special devices and infrastructures used in factories to increase the efficiency of assembly lines, quality assurances, etc. There are other forecasts announced by other companies including Ericsson, International Data Corporation (IDC), and Gartner. Even though the predicted number are different, but they all anticipate a massive number of connected IoT devices.

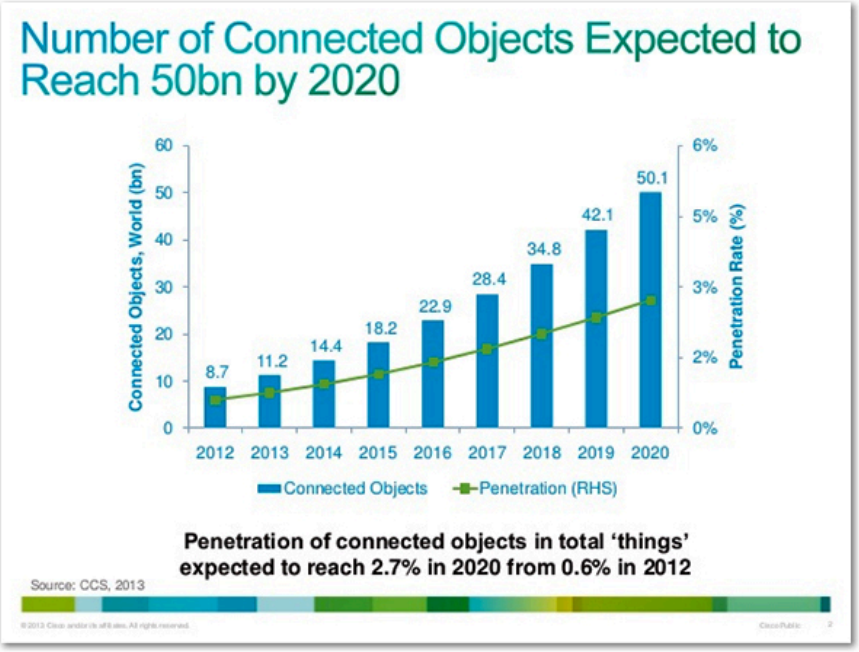


Figure 1.1 Estimated number of IoT devices by the year 2020

Figure 1.2 shows the components of IoT ecosystem, including embedded devices, analytics, networks, etc. [2]. IoT devices are interacting with the physical world using sensors and/or actuators to monitor and/or control the desired parameters. The gateway interfaces the IoT local networks with the Internet. The gateway bridges the networks, aggregates the collected data and even offers processing services. Cloud servers provide analysis and storage services.

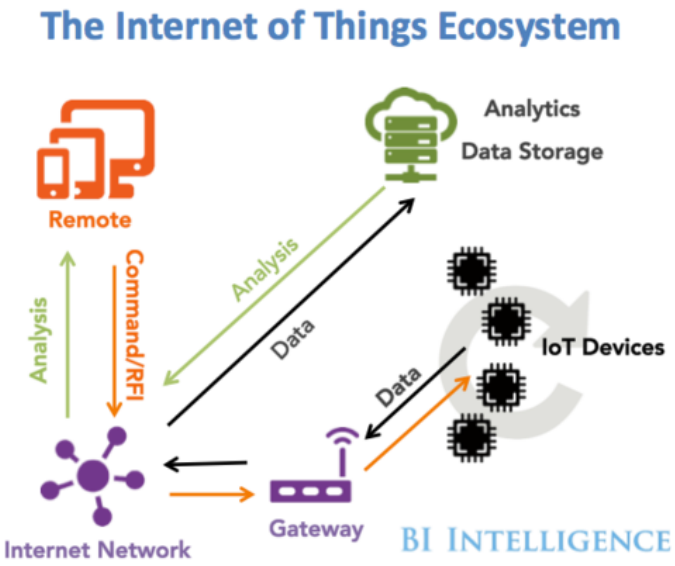


Figure 1.2 Components of the IoT ecosystem

Figure 1.3 shows the hierarchical layers of computation in an IoT system. As we move to the higher layers (from devices layer to the cloud layer), the processing capabilities increases. However, the latency would increase due to major factors: 1) network delay, and 2) more workload on the server. Therefore, the predictability of the real-time properties would decrease.

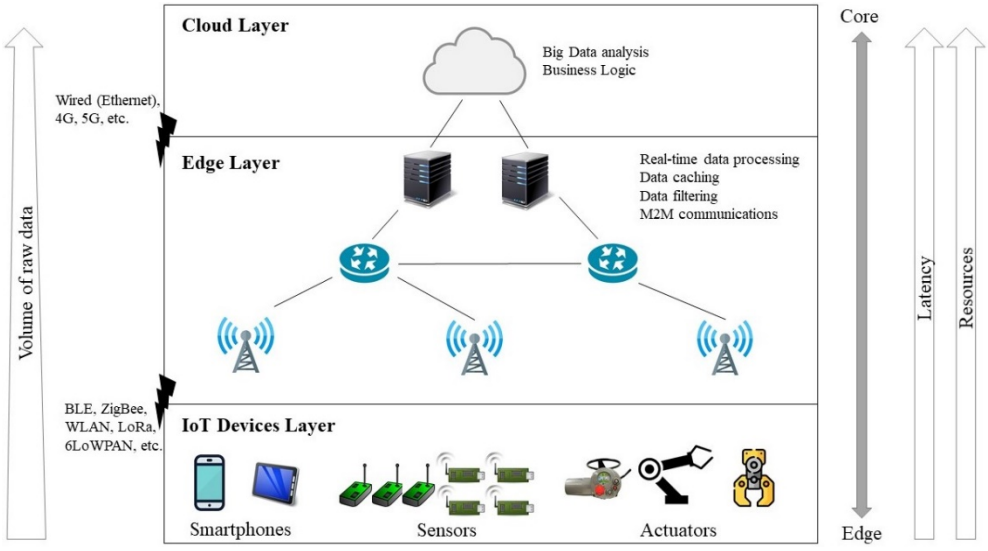


Figure 1.3 Computation layers in IoT systems and their properties

Since the majority of IoT systems are today cloud-centric, a large number of IoT devices, as well as their long-term and continuous data collection, will lead to a hard to manage the amount of acquired data which brings new challenges into the cloud computing world. Moreover, most IoT applications are not anymore compatible with the assumptions of cloud-centric architecture [3]:

1. there is sufficient bandwidth to push data to the cloud;
2. the connectivity is not an issue. A device will always be connected to the cloud;
3. the latency induced by cloud-centralized analytics and control is compatible with the dynamic of the IoT system;
4. the connectivity cost is negligible;
5. industrial companies are comfortable in exposing their data to the cloud (privacy).

One of the most common challenges in IoT is to process and analyze a huge amount of data from heterogeneous devices. This challenge has two aspects: 1) the large volume of data (which is known as Big Data), and 2) diverse application and ecosystem requirements of IoT devices. Handling all these collected data with the central cloud is inefficient, and even sometimes is unfeasible [4], because of:

- unreliable and high latency of the network caused by the high workload to transmit,
- most of the IoT applications require mobility support and geo-distribution in addition to location-awareness.

Edge Computing (EC) is a promising solution to address these issues [5]. In EC the task of data processing is pushed to the edge of the network (comprising gateways and edge devices) close to where data is collected or produced. This approach reduces the application latency (and increase the reliability of the application) and reduces the load on the IoT network by distributing the computation. Typically, an EC platform brings huge benefits on IoT management including proximity, lower latency, and location awareness [6]. Nevertheless, EC is still in its infancy and a complete framework (or infrastructure) to ensure the mentioned benefits is not yet available. Such frameworks will need to satisfy requirements, such as application deployment to process requests in real-time at the edge nodes. Moreover, it brings new challenges and problems that are needed to be addressed including application portability, high performance, and resilience [6].

1.2 Thesis Contribution

The aim of this thesis is to study and investigate the challenges and opportunities of the Edge

Computing in order to improve the management of IoT devices and the efficiency of IoT applications. In particular, the contributions of this thesis are as follows:

- This thesis presents a novel and integrated architectural model for the design of new 5G enabled-edge supports, capable of synergically leveraging Multi-access Edge Computing (MEC) and Fog Computing capabilities together. Since MEC and Fog share many similarities, they adopt the same distributed architecture skeleton that consists of *cloud layer*, *edge layer*, and *end devices layer*, in this work we decide to smartly combine functionalities by minimizing overlapping features and by exploiting synergies. In this perspective, the work proposes a new architectural model based on the introduction of a fully-converged 5G-Enabled Edge (5GEE), as a core element deployed at the edge layer in the proximity of the mobile node.
- This thesis faces out the problem of service migration by proposing the Mobile Edge Service Handoff (MESH) an Edge Computing platform architecture that supports service migration with different options of granularity (either entire service/data migration, or proactive application-aware data migration). In particular, this work presents application-specific optimizations in order to decrease the total migration time by leveraging Docker Containers technology. The proposed platform also addresses the problem of edge nodes heterogeneity by defining an affinity relationship between the service and edge nodes.
- This thesis proposes the Mobile Edge File System (MEFS), an application-level distributed file system designed to support mobile-edge-cloud tasks offloading. MEFS efficiently guarantees consistency among the mobile, edge, and cloud entities and supports application handoff through live migration as end devices move between edge nodes. This work also produces a prototype based on Android and a set of extensive experiments with a test app and real mobile user trace, to validate the functionality and performance of MEFS.
- This thesis introduces the DRIVE a framework for service discovery in a 5GEE environment. The work guarantees dynamic distribution and management of services by leveraging Docker Containers technology. Moreover, we distributed edge services at three different layers of communication such as Application, Service, and Communication layer in order to guarantee the maximum interoperability between components.
- This thesis presents an infrastructure to support distributed Machine Learning (ML) by

enabling edge devices to collaboratively learn a shared model while keeping local knowledge stored at the edge of the network. Moreover, the infrastructure gives the possibility of improving the model through the cloud that acts as a supervisor of the system and contains the global knowledge of the entire system. This work also produces and analyzes two case studies, namely video streaming processing for face recognition and predictive diagnostics for IIoT, deployed in a collaborative edge platform.

1.3 Thesis Outline

The reminder of this thesis is structured as follows:

- Chapter 2 provides background and detailed overview of the related work and state-of-the-art. First, an overview of existing systems and open-source projects for edge computing by categorizing them from their design demands and innovations. Then, identifies challenges and open research issues of edge computing systems. Finally, we present the state-of-the-art of service migration, tasks offloading and service discovery in the Edge Computing environment.
- Chapter 3 presents the main contribution of this thesis. It provides essential knowledge on Multi-access Edge Computing (MEC) and Fog Computing. It also presents the differences between the two paradigm and possible synergies as well as. By discussing the key characteristics, main application domains, and major research issues for both models, this chapter proposes a new edge-based architectural model that integrates MEC and Fog into a unique architecture.
- Chapter 4 describes our novel approaches for service migration in the Edge Computing environment. First, a background on service migration and container service migration at the edge has been presented in Section 4.1. Then, Section 4.2 shows our architectural solution and primary handoff guidelines. Section 4.3 explains our novel application-aware strategies. Finally, the experimental results are presented in Section 4.4.
- Chapter 5 presents the project named Mobile Edge File System (MEFS) that is done in collaboration with the New Jersey Institute of Technology in Newark, New Jersey. This chapter first motivates the need to have a filesystem that runs simultaneously at mobile devices, edge devices, and the cloud. Second, describes how we faced out the challenges introduced by the edge platform including state migration and resilience. Finally, it provides a description of our prototype and a set of experimental results.
- Chapter 6 provides some other functionalities for our proposed edge architecture.

Section 6.1 address the problem of service discovery in an EC platform by explaining our proposed solution called DRIVE. Instead, Section 6.2 investigates the idea to distribute Machine Learning in an EC platform.

- Chapter 7 concludes the thesis with a summary of the contributions and an outlook for the future extensions to this work. Section 7.1 address and discuss the achieved results by the works of this thesis. Finally, Section 7.2 address some future research direction.

2 BACKGROUND AND RELATED WORK

This chapter provides an overview and background for the following technical chapters and presents related work. This thesis envisions a dynamic Edge Computing platform that helps IoT devices and applications by offering computation offloading, resources allocation, and service discovery to enable the data processing at the edge of the network. An overview of existing Edge Computing platform including MEC and Fog Computing is presented in Section 2.1. Section 2.2 discusses the challenges and open research issues of Edge Computing and presents the state-of-the-art of service migration, computation offloading, and service discovery in the field of Edge Computing and their related work in the literature. This chapter contains the background that is needed for the proposed resource management techniques in the following technical chapters.

2.1 Edge Computing Systems

In the cloud computing era, the proliferation of the IoT and the popularization of 4G/5G gradually change the users' habit of accessing and processing data and challenge the linearly increasing capability of cloud computing. Edge Computing is a new computing paradigm with data processed at the edge of the network. Promoted by the fast-growing demand and interest in this area, the Edge Computing systems and tools are blooming, even though some of them may not be popularly used right now. Apart from Multi-Access Edge Computing (MEC), there are other Edge Computing systems such as Cloudlet, Mobile Cloud Computing, and Fog Computing. They tend to coexist with MEC in many technical contexts, hence the tendency for a misappropriation of these technologies given that they all have similar origin. However, these technologies are intrinsically different and each of them comes with its unique value proposition to both existing and future mobile networks as summarized in Table 1 [5].

Table 2.1 High level comparison of edge computing paradigms

	MEC	**Fog Computing**	**Cloudlet**	**MCC**
Initial promotion	ETSI (2014)	Cisco (2013)	C. Mellon Uni. (2009)	Aepona (2010)
Objective	Bring cloud computing capabilities closer to end-users			
Infrastructure Owner	Telecom operator	Private entities / individuals		
Node location	Radio Access Network (RAN)	Any strategic location between end-users and the cloud		
SW architecture	Mobile orchestrator based	Fog abstraction layer based	Cloudlet agent based	Service oriented

Service accessibility	Direct access from the closest end-user	Via Internet
Latency and jitter	Low	High
Context awareness	High	Low
Storage capability and computation power	Limited	High
Relevance to IoT	High	Low

Finally, in the next subsection, we review the aforementioned Edge Computing systems presenting architecture innovations, programming models, and applications, respectively.

2.1.1 Cloudlet

In 2009, Carnegie Mellon University (CMU) proposed the concept of Cloudlet [7], and the Open Edge Computing initiative was also evolved from the Cloudlet project [8]. Cloudlet is a trusted, resource-rich computer or cluster of computers that are well-connected to the internet and available to nearby mobile devices. It upgrades the original two-tier architecture "Mobile-Cloud" of cloud computing paradigm to a three-tier architecture "Mobile-Edge-Cloud". Meanwhile, Cloudlet can also serve users like an independent cloud, making it a "small cloud" or "data center" (DC) in a box. Although the Cloudlet project is not proposed and launched in the name of Edge Computing, its architecture and ideas fit those of the Edge Computing and thus can be regarded as an Edge Computing system.

The Cloudlet is in the middle layer of the three-tier Edge Computing architecture and can be implemented on a personal computer, low-cost server, or small cluster. Like Wi-Fi service

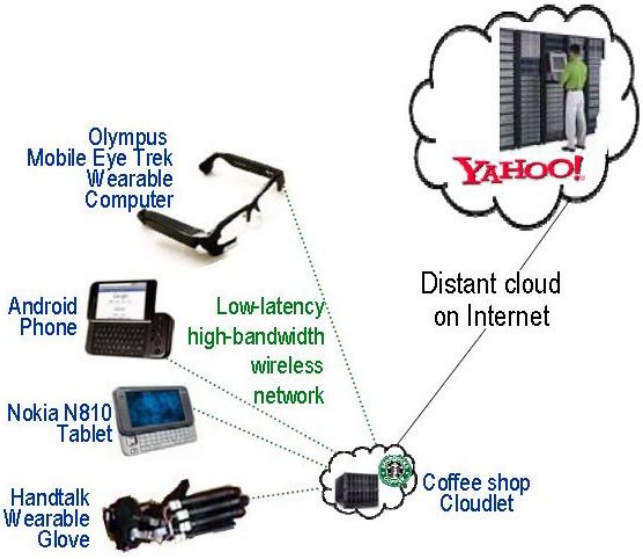


Figure 2.1 The cloudlet concept involves proximate computing infrastructure that can be leveraged by mobile devices

access points, a Cloudlet can be deployed at strategic location such as a restaurant, a café, or a library. Multiple Cloudlets may form a distributed computing platform, which can further extend the available resources for mobile devices. As the Cloudlet is just one hop away from users' mobile devices, it improves the Quality of Service (QoS) with low communication delay and high bandwidth utilization. In detail, Cloudlet has three main features as follows:

1. **Soft State**: Cloudlet can be regarded as a small cloud computing center located at the edge of the network. Therefore, as the server end of applications, the Cloudlet generally has to maintain state information in order to interact with users. However, unlike the cloud, Cloudlet does not maintain long-term state information for interactions, but only temporarily caches some state information. This reduces much of a load of Cloudlet.
2. **Rich Resources**: Cloudlet has sufficient computing resources to enable multiple mobile users to offload computing tasks to it.
3. **Close to Users**: Cloudlets are deployed at those places where both network distance and physical distance are short to the end-user. This guarantees high network bandwidth, short network delay, and low jitter. In addition, the physical proximity ensures that the Cloudlet and the end-user are in the same context (same location), based on which customized services could be provided.

As Figure 2.1 illustrates, Cloudlets are decentralized and widely dispersed Internet infrastructure components whose compute cycles and storage resources can be leveraged by nearby end-users. In addition, Cloudlet supports application mobility, allowing end-users to switch service requests to the nearest cloudlet during the mobile process. Cloudlet supports for application mobility relying on three key steps:

1. *Cloudlet Discovery*: Mobile devices can quickly discover the available Cloudlets around them and chose the most suitable one to offload tasks.
2. *Virtual Machine (VM) Provisioning*: Configuring and deploying the service VM that contains the server code on the cloudlet so that is ready to be used by the client.
3. *VM Handoff*: Migrating the VM running the application to another cloudlet.

The usage of VM in Cloudlet enables clean separation (each end-user is able to run applications in a separate VM). The complex problem of configuring software on the cloudlet to service mobile devices is avoided. Instead, the problem is transformed into a simpler problem of rapidly delivering a precisely preconfigured VM to the cloudlet. A VM cleanly encapsulates and separates a transient guest software environment from the permanent host software environment of the Cloudlet infrastructure. The interface between host and guest is stable and narrow. This ensures the longevity of the cloudlet and increases the chances of compatibility

between cloudlet and mobile device. Nevertheless, the VM approach is weaker than alternatives such as software virtualization or process migration. One VM image is many gigabytes in a size. In a hostile environment, efficient dissemination of VM to the cloudlet is a major challenge. When a cloudlet acquires a copy of a VM, it can treat it as a persistent cache copy and keep it until the space has to be reclaimed [9]. Mobile devices can be connected again in the future with the same cloudlet, with persistent caching of VM's.

2.1.2 Mobile Cloud Computing (MCC)

The term "Mobile Cloud Computing" was introduced not long after the concept of cloud computing [10]. The MCC forum has defined MCC as follows:

"*Mobile cloud computing at its simplest, refers to an infrastructure where both the data storage and data processing happen outside of the mobile device. Mobile cloud applications move the computing power and data storage away from mobile phones and into the cloud, bringing applications and MC to not just smartphone users but a much broader range of mobile subscribers*".

Aepona [11] describes MCC as a new paradigm for mobile applications whereby the data processing and storage are moved from the mobile device to powerful and centralized computing platforms located in clouds. These centralized applications are then accessed over wireless connection based on a native client or web browser on the mobile devices. Alternatively, MCC can be defined as a combination of mobile web and cloud computing [12], which is the most popular tool for mobile users to access applications and services on the Internet. Anyway, MCC aims at providing cloud computing services in a mobile environment and overcomes obstacles for the hosting nodes (e.g., heterogeneity, availability) and performance (e.g., battery autonomy, limited computation capabilities). The mobile devices do not need a powerful configuration (e.g., CPU and RAM) because all the computations can be processed at the cloud.

Figure 2.2 shows the general architecture of MCC, where mobile devices are connected to the mobile networks via base stations (e.g., base transceiver station, access point, or satellite) that establish and control the connections and functional interfaces between the networks and mobile devices. Here, mobile network operators can provide services to mobile users such as authentication, authorization, and accounting. In the cloud, cloud controllers process the requests to provide mobile users with the corresponding cloud services. These services are developed with the concepts of utility computing, virtualization, and service-oriented architecture.

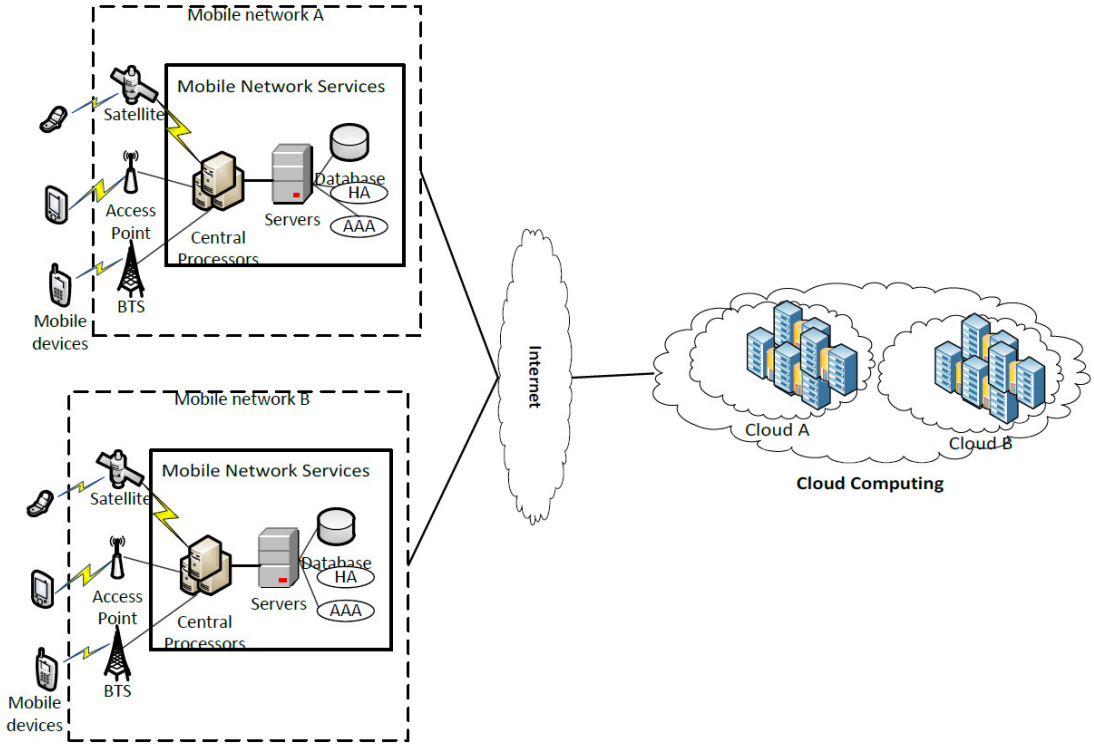


Figure 2.2 Mobile Cloud Computing Architecture

The details of cloud architecture could be different in different contexts. MCC architecture is based on a layered service-oriented cloud computing architecture as specified in [13]. This architecture is commonly used to demonstrate the effectiveness of the cloud computing model in terms of users' QoS. It incorporates Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS), and provides these services like utilities. Figure 2.3 shows the service-oriented cloud computing architecture.

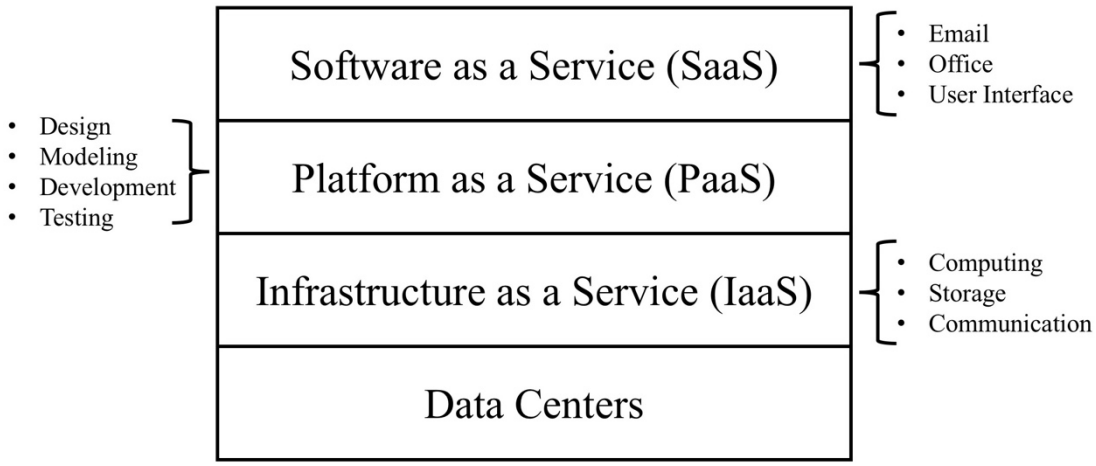


Figure 2.3 Service-oriented Cloud Computing Architecture

- **Data Centers**. This layer provides the hardware facility and infrastructure for clouds. In data center, a number of servers are linked with high-speed networks to provide services for customers.
- **IaaS**. Infrastructure as a Service is built on top of the data center layer. IaaS enables the provision of storage, hardware, servers, and networking components. Users can scale up and down these computing resources on demand dynamically. Examples of this layer include Amazon EC2, Google Compute Engine, and Digital Ocean.
- **PaaS**. Platform as a Service offers an advantage integrated environment for building, testing, and deploying custom applications. Amazon Elastic Beanstalk, Windows Azure, Heroku, and OpenShift are among examples of this layer.
- **SaaS**. Software as a Service supports a software distribution with specific requirements. In this layer, the end-users can access an application and information remotely via the Internet and pay only for that they use. An example of SaaS is Google Maps.

Although the cloud computing architecture can be divided into four layers as shown in Figure 2.3, it does not mean that the top layer must be built on the layer directly below it. For example, data storage service can be considered to be either in IaaS or PaaS.

Cloud Computing is considered to be a promising solution for mobile devices because of many reasons (e.g., communication, portability, and mobility) [14]. The main advantages of MCC are listed in the following:

- *Extending battery lifetime*. Battery is one of the main concerns of mobile devices. Several works are proposed in the literature to enhance battery lifetime, for instance, improve CPU performance, or manage the disk and screen in a smart manner to reduce power consumption. However, these solutions require to modify the architectural design of mobile devices or require new hardware that may increase the cost of mobile devices. MCC proposes a technique named Computation Offloading [15] with the objective to migrate the computation from resource-limited devices (mobile devices) to powerful machine (servers in the cloud). This avoid taking a long application execution time on mobile devices which results in large amount of power consumption.
- *Improving data storage and processing power*. Storage capacity is also a constraint for mobile devices. MCC is developed to enable mobile users to store/access data on the cloud through a wireless connection. There are many examples including Amazon Simple Storage Service [16] which supports file storage service, and Google Photos [17] which enables mobile users to upload images to the cloud immediately after

capturing giving also the possibility to access all images from any devices. Anyway, with the cloud, users can save a considerable amount of energy and storage space on their mobile devices because all images are sent and processed at the cloud.

- *Improving reliability*. Storing data or running applications at the cloud is an effective way to improve reliability because the data and application are stored and backed up at the cloud. This, of course, reduce the chance of data and application lost on the mobile devices. In addition, MCC can be designed as a comprehensive data security model for both service providers and users. Also, the cloud can remotely provide to mobile users some security services such as virus scanning, malicious code detection, and authentication.

In spite of CC offers enormous advantages to mobile devices, MCC has to face many technical challenges. Some of these have available solutions and others are not addressed yet. The main open issues of MCC are listed in the following:

- *Network latency and low bandwidth*. Bandwidth is one of the big issues in MCC because the radio resources for wireless networks is much scarce as compared with the traditional wired network. In addition, mobile devices have to communicate with the cloud at any time they want to execute a service. This could result in a bottleneck of the MCC systems.
- *Availability*. Service availability becomes a more important issue in MCC than in the CC with a wired network. Mobile users may not be able to connect to the cloud to obtain a service due to several reasons such as traffic congestion, network failure, and the out-of-service.
- *Heterogeneity*. MCC will be used in the highly heterogeneous networks in terms of wireless network interfaces. Different mobile nodes access to the cloud through different radio access technologies such as GPRS, LTE, WiMAX, and WLAN. As a result, an issue of how to handle wireless connectivity while satisfying MCC's requirements arises.

2.1.3 Multi-access Edge Computing (MEC)

Challenges faced in the MCC such as the long propagation distance from the end-user to the remote cloud center, result in excessively long latency for mobile applications. For the Internet of Things, reliability, mobility, and security are some of the top requirements. Authors in [18] assert that MCC is inadequate for a wide-range of emerging mobile applications that are

latency-critical. Therefore, with an increase in IoT devices and 5G communications, there has been a shift from a centralized approach such as MCC to a decentralized approach as Multi-access Edge Computing (MEC). MEC is cloud-computing capabilities at the edge of the network, within the Radio Access Network (RAN) and in close proximity to mobile devices. In particular, MEC is an architectural model and specification proposal (i.e., by European Telecommunications Standards Institute – ETSI) that aims at evolving the traditional two-layers cloud-device integration model, where mobile nodes directly communicate with a central cloud through the Internet, with the introduction of a third intermediate middleware layer that executes at so-called network edges. This promotes a new three-layers device-edge-cloud hierarchical architecture, which is recognized as very promising for several application domains [6]. In particular, the new MEC model allows moving and hosting computing and storage resources at network edges close to the targeted mobile devices, thus overcoming the typical limitations of direct cloud-device interaction, discussed in the previous subsection. As illustrated in Figure 2.4, heterogeneous devices are connected to MEC servers that are in closer proximity which offer services and applications.

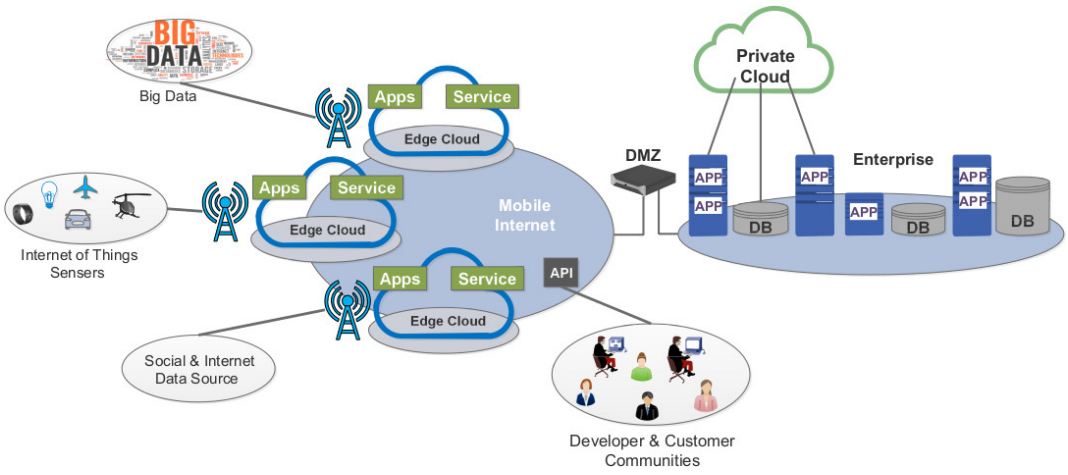


Figure 2.4 Multi-access Edge Computing High Level Architecture

According to the white paper published by ETSI, MEC can be characterized by [6]:

- **On-Premises**: One edge node is local, meaning that it can run isolated from the rest of the network while having access to local resources. This becomes particularly important for Machine-to-Machine scenarios, for example when dealing with security or safety systems that need high levels of resilience.

- **Proximity**: Edge Computing infrastructure is particularly useful to capture key information for analytics tasks and it may also have direct access to the devices which can easily be leveraged by business-specific applications.
- **Lower Latency**: As Edge services run close to the end devices it considerably reduces latency. This can be utilized to react faster, to improve Quality of Experience (QoE) and Quality of Services (QoS), or to minimize congestion of the network.
- **Location Awareness**: One edge node is part of a wireless network, whether it is Wi-Fi or Cellular, local service can leverage low-level signaling information to determine location information of each connected devices.
- **Network context information**: Real-time network data including radio conditions, network statistics, etc., can be used by applications and services to offer context-related services.

MEC facilitates enhancements to the existing applications and offers tremendous potential for developing a wide range of new and innovative applications by enabling authorized third parties to make use of local services and caching capabilities at the edge of the network. MEC can be seen as a natural evolution of legacy mobile base stations, enabling new business opportunities. The MEC white paper provided the main MEC use cases divided into three different application groups: Network-Centric Applications, Enterprise and Vertical Applications, and Efficient Delivery of Local Content [6].

- *Distributed Content and DNS Caching*. The main idea is to store popular content and data at the base station (MEC server). This allows reducing backhaul capacity requirements by up to 35%. Moreover, DNS Caching allows improving the QoE by reducing web pages download time by 20%. (See Figure 2.5a).
- *Ran-aware & Application-aware Content Optimization*. The application exposes accurate cell and subscriber radio interface information including cell load and link quality to the content optimizer enabling dynamic content optimization, improving QoE, and network efficiency. Also, application-aware cell performance optimization for each device in real-time can improve network efficiency and customer experience. (See Figure 2.5b).
- *Active Device Location Tracking*. The infrastructure allows the application to get a mobile device location in real-time and in a passive way (no GPS). It enables location-based services for enterprises and consumers. Relevant in a “smart city” environment. (See Figure 2.6a).

- *Intelligent Video Analytics*. The video management application transcodes, and stores captured video stream from cameras at the base station. The video analytics application processes the video data to detect and notify specific events. The application sends low bandwidth video metadata to the central operations and management server for database searches. Applications may range from safety, public security to smart cities. (See Figure 2.6b).
- *Augmented Reality Content Delivery*. An Augmented Reality application overlays augmented reality content onto objects viewed on the device camera. Applications on the MEC server can provide local object tracking and local AR content caching. This solution minimizes round trip time and maximizes throughput for better QoE. (See Figure 2.7).

As stated before, MEC may become an enabler for real-time context-aware applications combining MEC and RAN. Recent implementations of ETSI MEC framework have been focused on the integration of LTE-A, MEC, Network Function Virtualization (NFV), and Software Defined Networking (SDN) [19]. SDN is emerging as a natural solution for next-

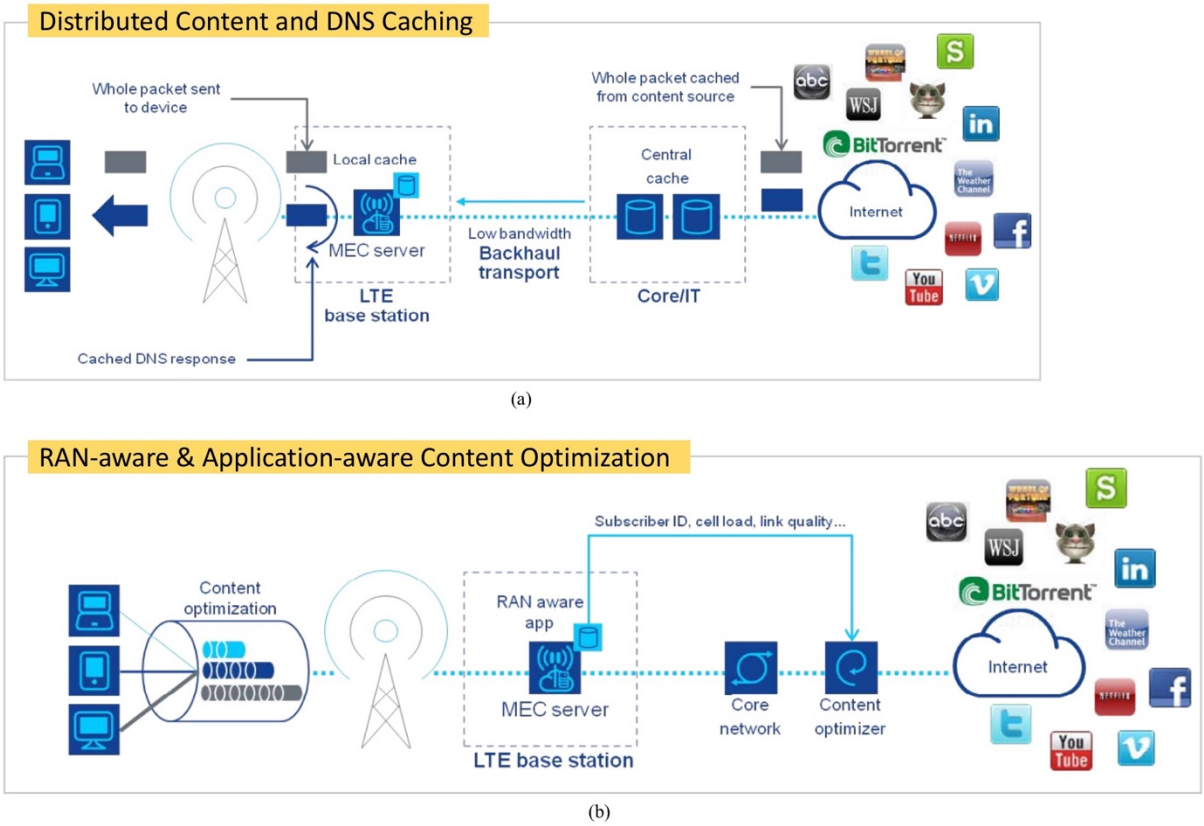
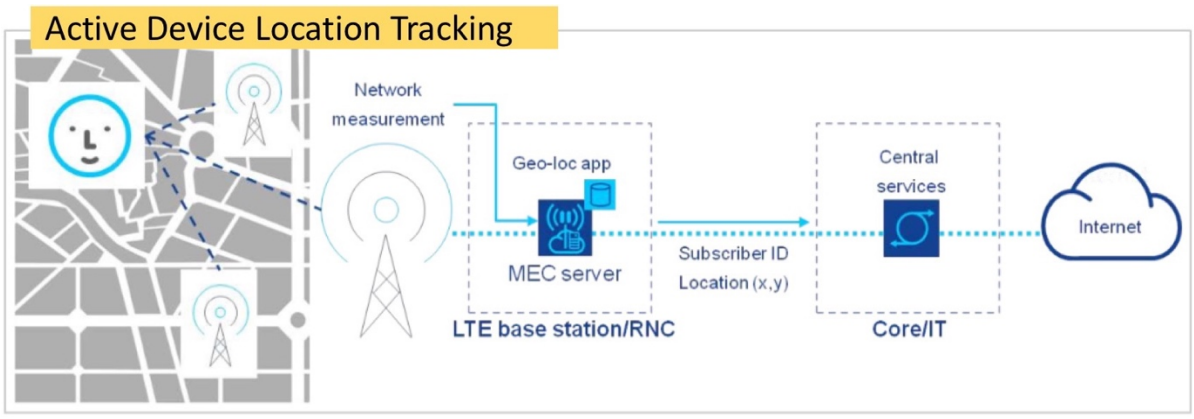
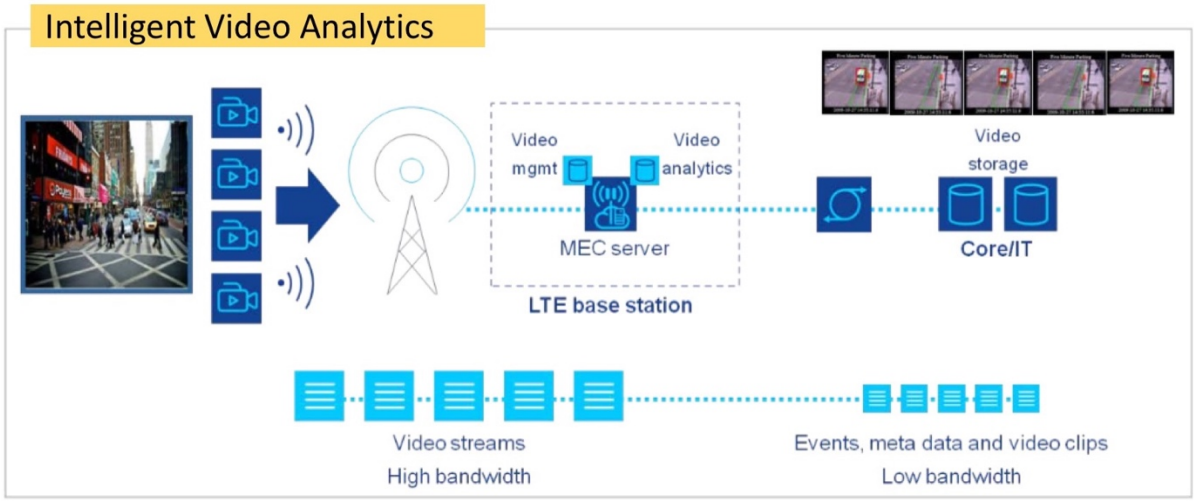


Figure 2.5 Use cases: Network-Centric Applications



(a)



(b)

Figure 2.6 Use cases: Enterprise and Vertical Applications

generation cellular networks as it enables further network function virtualization opportunities and network programmability [20]. Instead, NFV is an ETSI standard which focuses on transforming Network Functions into Virtual Network Functions. In MEC, SDN and NFV will

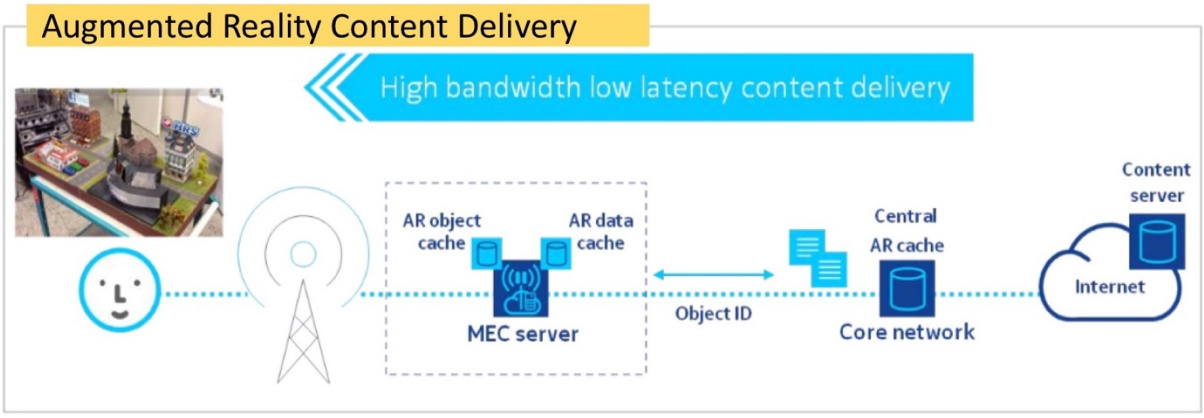


Figure 2.7 Use cases: Efficient Delivery of Local Content

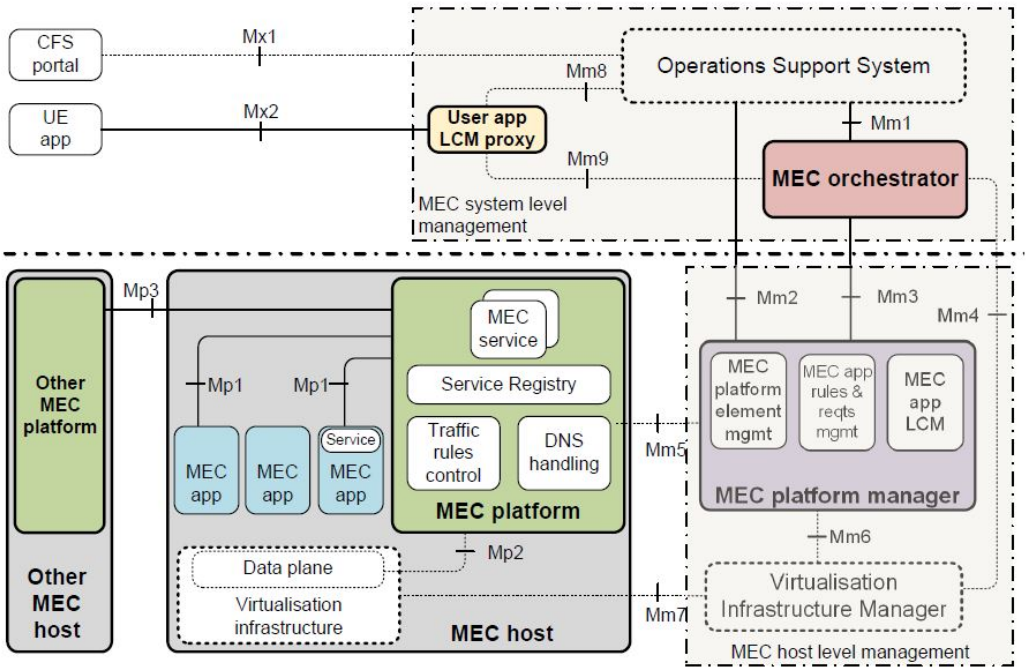


Figure 2.8 The architecture of the MEC system

allow extreme flexibility, when it comes to the specification of extended logics of micro-service architectures at the network edge. The MEC function chain will be managed by the Virtual Network Functions Manager (VNFM) responsible for the installation of Virtual Network Functions (VNFs) and the SDN controller. As illustrated in Figure 2.8, the MEC system consists of the (upper) MEC system level and the (lower) MEC host level. The Custom Facing Service (CFS) for third parties and User Equipment (UE) application portals are entry points towards the MEC system. Therefore, the MEC system allows third parties to install MEC Apps on the MEC host. The MEC App receives traffic directly from the data plane from nearby eNBs by an appropriate traffic configuration. The entire MEC system is divided into separated inter-connected entities, which communicate through reference points defined between them (Mm1-9, Mp1-3, and Mx1-2). The MEC host provides a MEC platform and a Virtualization infrastructure, which run and control MEC apps. From the perspective of MEC apps, the MEC platform uses the Mp1-2 reference points to provide:

- service discovery, registration, and communication, i.e., offering and consuming services (Mp1);
- data plane into the virtualized infrastructure of ME Apps (Mp2).

A user requests a new App through the portal, both CFS and UE app. First, the request arrives at the Operations Support System (OSS) that communicates with the Mobile Edge Orchestrator to manage the lifecycle of Apps. The orchestrator uses the MEC Platform Manager and VIM

to appropriately configure the MEC Platform and Virtualization Infrastructure on the MEC host respectively. On the way from the CFS portal, the lifecycle management of Apps on MEC host is controlled by the Mx1 – Mm1 – Mm3 – Mm6 – Mm7 reference points, while the traffic rules providing the data plane to MEC Apps are provided by the Mx1 – Mm1– Mm3 – Mm5 – Mp2 reference points. For more detail, please consult [21]. Furthermore, the MEC system provides the use of standard APIs and SDKs as key pillars for developing MEC applications within the ecosystem described above. Nevertheless, according to the white paper, there are several technical challenges that still to be solved. The most important challenges are described in the following [6]:

- **Network integration**. The introduction of the MEC servers within the network base station should be completely transparent. This means that the existing 3GPP specifications should not be affected by the presence of the MEC server as well as the MEC applications being hosted on it.
- **Application portability**. The MEC system needs to provide a mechanism for the rapid transfer of MEC applications, which may occur on the fly, between MEC servers. This as a consequence, for instance, of users' mobility (handoff procedure) or resources reallocation. For this reason, the platform-management framework needs to be consistent with the different solutions to ensure that. Moreover, tools and mechanisms used to package, deploy and manage applications also need to be consistent across platforms and vendors.
- **Security**. The MEC platform needs to simultaneously fulfill the 3GPP security requirements while providing a secure sandbox for MEC applications. To ensure this, the MEC platform has to isolate the applications as much as possible from the burden of having to relate to all the implications of 3GPP security (e.g., use Virtual Machine technologies).
- **Performance**. The MEC platform and MEC applications hosted on it need to be dimensioned and should have enough capacity to process the user traffic that is handled by the 3GPP network element. The MEC applications shall be transparent to the UE and, at the same time, shall provide improved QoE. Recent
- **Resilience**. The MEC platform is designed to host applications that process user requests; these hosted applications need to be robust and resilient. To protect against any anomalies, the MEC platform needs to provide a fault tolerance mechanism to ensure that MEC applications operate without problems. If a fault is detected, the MEC

platform should migrate the user traffic towards a new place to prevent a service disruption.

In conclusion, MEC transforms the base station into intelligent service hubs that are capable of delivering highly personalized services directly at the edge of the network while providing the best possible performance in the mobile network. Moreover, the MEC initiative aims to benefit a number of entities within the value chain, including mobile operators, application developers, Over the Top (OTT) players, etc.; all of these parties are interested in delivering services based on Multi-access Edge Computing concepts.

2.1.4 Fog Computing

In parallel with the assertion of MEC, with a stronger emphasis on the (sensing) devices and communication-enabled things, hence stemming from the Internet of Things (IoT) scenario, Cisco has initially proposed the Fog Computing (or simply Fog for the sake of briefness) paradigm, and we have recently witnessed some related standardization efforts inside the Open Fog Consortium [22]. The proposed model shares with the MEC the idea of interposing an intermediate layer, deployed at the edge of the network, between final devices and the central cloud. The idea of Fog is to support the Cloud Computing to overcome most of IoT applications needs, low latency, geo-distribution, location-awareness, and mobility support in order to efficiently collect and promptly process the IoT data. Cloud computing suffers from substantial yet unsolved challenges such as large end-to-end delay, traffic congestion, processing of massive amount of data, and communication cost. Some of these issues are caused mainly due to the large physical distance between the cloud service provider's Data Centers (DCs). To overcome these issues, Fog is defined as an extension to the Cloud Computing that brings Cloud capabilities close to the edge, hence IoT sources, as well as the end-users. Cisco defined the concept of Fog as a bridge between the IoT devices and large-scale Cloud Computing and storage services [23]. The term "fog" is used simply because "*fog is a cloud close to ground*" [4]. Fog is a highly virtualized platform that provides computing, storage, and networking services between the end-users and DCs of the traditional Cloud Computing. Moreover, it has a distributed architecture targeting application and services with widely spread deployment analogous to the IoT. Figure 2.9 presents the idealized information and computing architecture supporting the IoT applications and illustrates the role of Fog [4].

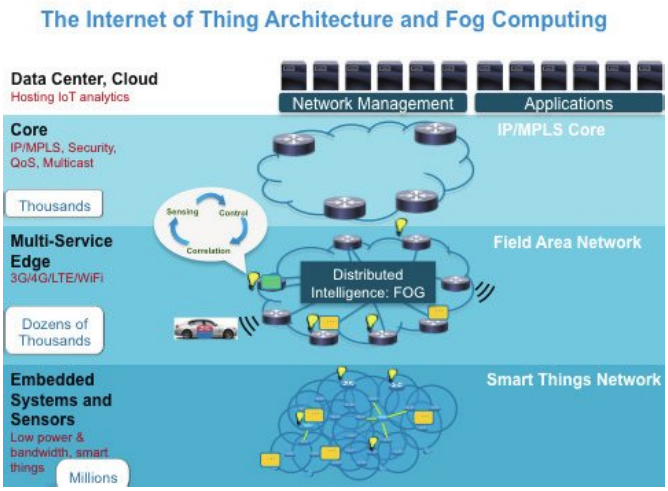


Figure 2.9 The Internet of Thing Architecture and Fog Computing

Fog is positioned as an intermediate layer between Cloud infrastructure and IoT devices. Thus, Fog nodes bridge application objects running in the Cloud and the edge. The major benefit is the support of IoT environments with computing resources, communications protocol, location awareness, mobility support, low latency, geo-distribution, and enhanced QoE. The Open Fog Consortium [22] defines the main capabilities of Fog as SCALE, an acronym stands for Security, Cognition, Agility, Latency, and Efficiency. The most widely accepted and recognized characteristics of Fog are [4]:

- Edge Location, Low Latency and Location Awareness. Fog supports endpoints with rich services at the edge of the network, including applications with low latency requirements (e.g., gaming, video streaming, augmented reality).
- Geographical distribution. As opposed to the Cloud, the services and applications targeted the Fog demand widely distributed deployments. The Fog will play an active role in delivering high-quality streaming to moving devices through access points geographical distributed.
- Support for Mobility. Mobility support is a key requirement for many real-time IoT systems as missed or delayed data during mobility can lead to severe consequences. In order to support mobility, an IoT system needs to be equipped with a handoff mechanism which is responsible for de-registering a sensor node from a source fog node and registering it to a new fog node seamlessly.
- Capacity of Processing High Number of Nodes, as a consequence of the wide geo-distribution, as evidenced in sensor networks in general, and the Smart Grid in particular.

- Predominance of Wireless Access.
- Real-Time Interactions. Important Fog applications involve real-time interactions rather than batch processing.
- Heterogeneity. Fog nodes come in different form factors and will be deployed in a wide variety of environments.

According to Cisco [24], the Fog paradigm provides an ideal place to analyze most data near the devices that produce and act on that data instantaneously. The devices that are within the Fog environment are known as fog devices or fog nodes. Fog nodes can be resource-poor devices such as set-top-boxes, access points, routers, switches, base stations, and end devices, or resource-rich machines such as Cloudlet and IOx [25]. IOx is a Fog device product from Cisco, whose architecture is depicted in Figure 2.10, works by delivering an application enablement framework that brings the Fog concept to life by allowing the delivery of distributed computing capabilities and enabling the creation of an intermediate layer between the "things" and the cloud.

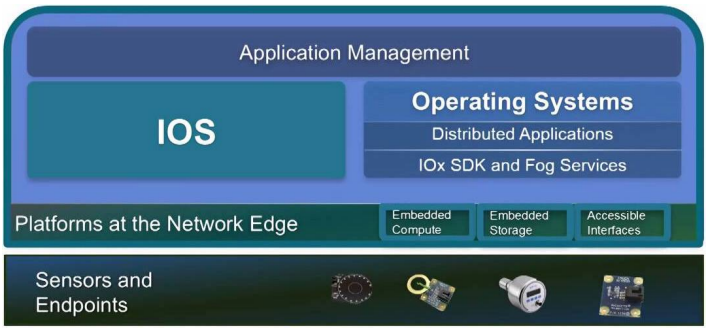


Figure 2.10 IOx Architecture

Cisco Systems is not the only one to have provided a definition of Fog Computing. Vaquero and Rodero-Merino has defined Fog as "*a scenario where a huge number of heterogeneous (wireless and sometimes autonomous) ubiquitous and decentralised devices communicate and potentially cooperate among them and with the network to perform storage and processing tasks without the intervention of third-parties. These tasks can be for supporting basic network functions or new services and applications that run in as and boxed environment. Users leasing part of their devices to host these services get incentives for doing so*" [26]. Several consortium and important company have addressed the emerging Fog topic. For instance, IBM has defined Fog as "*the term fog computing or edge computing means that rather than hosting and working from a centralized cloud, fog systems operate on network ends*" [27]. Whereas, the Open Fog Consortium has stated that "*fog is a system-level horizontal architecture that distributes*

resources and services of computing, storage, control and networking anywhere along the continuum from cloud to Things" [22]. According to Yi et al. in [28], the definition given by Vaquero and Rodero-Merino is debatable and a definition that can distinguish clearly between Fog computing and other related computing paradigms is still required. The definition given by IBM represents Edge and Fog computing as the same computing paradigm. While, from the Shi et al. point of view, in [29], Fog Computing focuses more on the infrastructure side while MEC focuses more on the side of the things. In conclusion, the definitions given by Cisco, Open Fog Consortium, IBM, and many others, see the Fog as a paradigm main focused on supporting IoT and end-user devices. This thesis carefully addresses the problem of defining boundaries between Fog and MEC concepts, indeed, the next chapter is completely dedicated to discussing similarities and differences between them.

2.2 Related Work

After having introduced the context in which this research project has been placed and the goals it aimed to achieve, this section provides a large overview of the state of the art and similar researches.

2.2.1 Multi-Access Edge Computing Systems

The rapid increase in demand for mobile devices within the realms of real-time mobile applications, augmented reality, and mobile gaming, and Industry 4.0 (just to cite a few) motivate the need for real-time mobile cloud applications. Necessarily, these real-time applications require low latencies to provide seamless end-user interaction imposing very strict QoS requirements. For instance, cloud-based multimedia real-time applications require end-to-end latencies below 60ms and much lower values within specific contexts such as the industrial one [30, 31]. The only way to comply with those requirements is moving cloud computing to the edge of the network, in other words, shifting from the MCC to the MEC model. Accordingly, in the last years various models and solutions have been proposed by academia and industry, such as Follow-Me Cloud [32] and Micro Datacenter (MDC) [33, 34]. The authors in [32], define and describe a new architectural model for federated cloud architecture which allows mobile users to always be connected via the optimal gateways, while cloud-based services follow them. The paper provides a first sketch of the system architecture based on the MEC design (three-layers architecture), while most efforts are focused on schemes for ensuring services continuity. Analytic models based on Markov Decision Process are proposed in this

respect. In conclusion, even if the paper proposes sophisticated algorithms to migrate services in order to follow users on the move, it does not address all challenges described by the MEC. Other works, instead, concern the integration of multiple implementations of Edge Computing including the work presents by Jararwehet al. [35] which presents a study on the integration of MEC and Cloudlets. This work produces a hybrid Mobile Cloud Computing framework where Cloudlet controllers are responsible for keeping smooth communication between things inside the same location, while MEC controllers are responsible for all other decisions that require a high level of control. This Cloudlet/MEC layering system allows to achieve many advantages such as i) reducing the MEC controller overhead, ii) taking real-time decisions, and iii) avoiding the single point of failure. This is one of the first works that try to combine different Edge Computing systems. Despite this, this work did not address the critical challenges of integrating two different systems including finding a suitable combination of functionalities between the two systems. On the contrary, the major contribution of this thesis is to smartly combine the functionalities of the MEC and Fog models in order to explore the mutual advantages in the joint synergic exploitation of these two models. Only a few numbers of researches on the field of Edge Computing have been started by companies. Microsoft, in [33], discussed the emergence of micro datacenter (MDC) for mobile edge computing. In the article, the authors present a set of cases study where they analyzed the latency issues introduced by the cloud. To overcome this, they proposed a framework based on Microsoft Azure, able to provide computation, storage, and network functionalities at the edge of the network within micro datacenters. Finally, they showed all the benefits of using MDCs in terms of bandwidth saved and reducing latency time. Differently, IBM, in [34], started to produce the hardware supports for future micro datacenters. In particular, they propose DOME [36] a small micro server board that contains all essential functions of a similar cloud server with a major focus on energy-efficiency. For further information please refer to [36]. Despite some recent companies involvement, there is still not a suitable architecture today that can be used for off-the-shelf.

2.2.2 Fog Computing Systems

As it is already stated, the concept of Fog Computing was introduced in 2012 by Cisco. The Fog paradigm entails moving intelligence down to the local area network (LAN) level and data is processed at an IoT gateway. The main aim of Fog is to extend services and functionalities offered by the cloud at the edge of the network. With exciting benefits of minimizing network congestion, minimizing end-to-end latency, tackling connectivity bottlenecks, improving

security and privacy, and enhancing scalability, the Fog is seen as the way forward. Furthermore, there are claims within the academia and the industry of the vast business opportunities that could be derived with the advent of the Fog. According to Mouradian et al., [37], recent seminal researches on Fog Computing are focused on two major categories: proposed architectures for Fog systems and proposed algorithms for Fog systems. Within the first category, the work proposed by Omoniwa et al. [38] proposes a Fog Computing distributed architecture (FECIoT) that enhances service provisioning from the cloud to things. In particular, the authors proposed an architectural framework to support IoT devices where fog/edge devices may be linked to form a mesh to provide load balancing, resilience, fault tolerance, data sharing, and reduction of the IoT-Cloud communication. Architecturally, this demands that fog/edge devices have the ability to communicate both vertically and horizontally within the IoT ecosystem. The FECIoT inherits the basic IoT architecture and delivers all IoT requirements in a more efficient way by leveraging on the distributed Fog Computing. Moreover, the paper has discussed three different architecture such as three-layers, four-layers, and five-layers architecture. The three-layers architecture, starting from at the top consists of the application, network, and devices layer. Instead, for four-layers architecture, the authors have added a service layer (SoA) between the application layer and the network layer for integrating third-party services. Finally, the five-layers architecture includes a business layer on top of the entire architecture which allows support for business and profit models. In conclusion, the work discusses why the FECIoT architecture should be deployed to fill possible technological gaps with a view to enhancing new business opportunities. Another interesting related work was proposed by Corsaro et al. [39], that have proposed a Fog Computing architecture, named fogØ5, with the aim of reducing heterogeneity and resource constraints problems of Fog environments. In particular, fogØ5 defines a set of abstractions to unify compute, storage, and networking across cloud, edge, and things. To ensure this, fogØ5 leverages OpenStack and NFV technologies and uses VNF Management and Orchestration (MANO) [40] as an orchestration of Virtual Network Functions (VNFs). In conclusion, fogØ5 addresses the major Fog Computing requirements and also provides the entire code as an open-source project. On the contrary, several research initiatives have focused on proposing algorithms for Fog systems. Oueis et al. in [41], have presented a task scheduling algorithm for Fog environments in order to manage tasks execution. More precisely, the work proposed an algorithm designed for load balancing for Fog that specifies metrics according to specific applications and network requirements. In [42], the authors present a service-oriented resource management model for Fog environments, which can help in efficient, effective, and fair

management of resources for the IoTs. Li et al. in [43], introduced a coding framework that allows to redistribute tasks and/or inject redundant ones in the Fog environments. The framework operates by considering the tradeoff between communication load and computation latency. In particular, they proposed two distinct coding schemes with different aims; the first coding scheme aims at minimizing bandwidth usage, while the second coding scheme targets the minimization of latency.

As it is already stated, all the approaches listed above addressed some of the challenges of the Fog architecture, while in this research has been addressed the challenge of fusion of the Fog model with the MEC standard.

2.2.3 Service Migration at the Edge

In the past few years, notable efforts of researches have been focused on the benefits and challenges of Edge Computing. One of the uncleared challenges is service migration, which guarantees service continuity as users move across different edge nodes. Ha et al. [44] proposed Cloudlet, as one of the seminal examples of computing at the network, and a mechanism for edge-enabled handoff management based on VM service synthesis and migration to the newly visited edge nodes. This has led to a few important middleware solutions to address service migration in the presence of user mobility. Mobile Micro-Cloud (MMC) [45] started exploring the idea to place micro-clouds closer to end-users. In that work, authors faced out the service migration problem by taking into account the costs associated with running service at the same MMC server and the costs associated with migrating the service to another MMC server. To do this, the authors define an algorithm to predict the future costs for finding the optimal placement of services. Another important work in the same context is Follow-Me Cloud [32] that enables mobile cloud services to follow mobile users alongside datacenters. The framework allows service migration by migrating all or portions of services to the optimal data center. Service migration decision is based on user constraints and network conditions. Some other recent proposals are based specifically on VM migration. Preliminary research efforts focused on the impact on network performance [46]. To overcome this limitation, various VM migration systems exploit the concept of live migration optimized for the edge computing. Live migration is mostly identifying as a technique for VM migration in datacenters at the cloud layer. Indeed, datacenters are assumed to be stable environments with high-bandwidth data paths always available. Most important solutions are based on pre-copy approaches, where VM control is not transferred to the destination until all VM state has been copied. On the contrary, post-copy approaches resume VM at the destination first and then the state is retrieved [47].

Ha et al. [44] highlight the limitations of traditional live VM migration on edge devices and propose live migration in response to client handoff in cloudlets, with less involvement of the hypervisor and by promoting migration to optimal offload sites, adapting to changing network conditions and processing capacity. The same authors also proposed a mechanism called VM handoff that supports agility for cloudlet-based applications [48]. The mechanism preserves the core properties of VM live migration for data center while optimizing for the agile environment of Edge Computing. This approach leverages on pipelined stages that aim at reducing the differences between the VM state at the source and VM state at the destination.
Containers differ from VMs technology since they directly share the hardware and the kernel with their host machines. As a result, containers occupy fewer resources and have lower virtualization overhead than VMs. For this reason, container migration has started to be a very active area that has not been systematically studied in the literature yet. Machen et al. [49] investigate live migration of LXC containers [50] by proposing a three-layer framework with synchronized filesystem methodology for memory state sync. Substantially, that work shows a quantitative view on the difference between LXC containers migration and KVM [51] migration. Live migration of containers become possible since CRIU [52] supports checkpoint/restore functionalities for the most container solutions such as OpenVZ [53], LXC/LXD, and Docker [54]. Several solutions have been explored in the literature that leverage CRIU for migrating stateful containers. OpenVZ supports live migration of containers [55]; however, it exploits Virtuozzo Storage System [56], that is a distributed storage system where all files are shared across the network. In most cases, the network bandwidth of edge servers is limited, and the deployment of a distributed storage could be not possible [57]. Moreover, the implementation is not optimized due to the transfer of root filesystem of the container across edge nodes. IBM proposes Voyager [58], a live container migration service designed in accordance with the Open Container Initiative (OCI) principles: the IBM solution implements a novel filesystem-agnostic and vendor-agnostic migration service with consistency guarantees. Summarizing the related work, although a few solutions have been proposed to contribute to the field of service migration in Edge Computing, there is no ready solution that exploits the characteristics of the application and works within a Fog or MEC architecture. As a consequence, this thesis proposes a solution to make service migration faster and easier than existing proposals.

2.2.4 Computation offloading at the edge

As explained in the previous section, offloading is one of the main features of MCC to improve the battery lifetime for mobile devices and to increase the performance of applications. Computational offloading for MCC is addressed, among the others, in CloneCloud [59], MAUI [60], ThinkAir, [61] and COMET [62], which propose task offloading onto a centralized surrogate in the cloud. CloneCloud migrates threads to application-level VMs. MAUI and ThinkAir adopt the offloading approach with method granularity. COMET provides distributed shared memory to support thread offloading. Even though all of them have shown the benefits of offloading for speedup and energy savings, there are many related issues including application portability and resilience as you're moving from the MCC to the MEC world. As an evolution to the above solutions for MCC, a few frameworks have been already proposed with a MEC or Fog oriented approach. MECO (Mobile-edge computation offloading) [63] is a technique proposed for prolonging the battery life and enhancing the computational capacity of mobile nodes. It aims at minimizing mobile energy consumption by considering the computation overhead and the available resources at MEC. The proposed algorithm calculates an offloading priority for each user. In addition, MECO is focused on model aspects and does not consider MEC challenges such as app portability and resiliency. Another recent work is CloudAware [64] that presents a programming model and a framework that directly fits the common app developer's mindset to design elastic and scalable MEC-based mobile applications with extremely low response time, e.g., multimedia applications. But these applications are typically stateless since the server is agnostic of the client state. Therefore, if the applications need to keep a server-side state, a mobility management technique is needed to manage service/state migration. Unlike these mentioned works, this thesis proposes an application-level distributed filesystem designed to be resilient that enables computation offloading at the edge and supports the session handoff.

2.2.5 Service discovery at the Edge

In the pervasive computing era, service discovery in IoT faces many challenges in terms of management of service information, service selection methods, service caching and power saving, etc. For that purpose, many service discovery protocols have been proposed but the field is still immature. The variants of service discovery in web, wireless ad-hoc and MANETs could be modified to provide an adequate solution for SD in IoT environments. So below we discuss some of the most relevant SD protocols available for both Internet and LAN protocols. Service discovery protocols like Service Location Protocol (SLPv2 [65] is the latest version) provides a scalable framework for service selection in the IP networks. The Universal

Description Discovery and Integration (UDDIv3.0.2 [66] is the current version) defines a standard method to publish and discover network-based software components in the service-oriented infrastructure. UPnP [67] was proposed for use in a small office and home environments and mainly targets device and service discovery by using the IP protocol. The Service Discovery Protocol (SDP) can search and access the services among Bluetooth's devices. Jini [68] provides SD in the form of object-oriented distributed computing technology based on Java. Bonjour [69] runs over the IP protocol and also has the capability of automatically assigning IP addresses to networked devices, even without the help of a DHCP server. The above-mentioned schemas have been conceived for LANs and are focused to work with a specific mechanism or protocol. This may result in a critical open issue for future converging MEC and Fog networks. In fact, one of the main problems of MEC and Fog is that in most cases the software is embedded in the edge node and operates for a specific ecosystem. This restriction can have implications for service discovery functionalities. Moreover, existing protocols are not targeting for constrained devices, such as those used in IoT environments. A common approach is to have an intermediate layer (edge layer) that by supporting multiple network technologies, enables constrained devices to communicate with the edge node. Efforts have been done to adapt these solutions to the world of heterogeneous constrained devices (IoTs). In [70], the authors proposed architecture for integrating the cloud and the IoT. In the work, they introduced the concept of “Smart Gateway” (SG), which acts as an intermediate layer between heterogeneous IoT devices and the cloud. This kind of a gateway, SG, would help in better utilization of network and cloud resources by providing several network technologies and service discovery functionalities. Likewise, Cirani et al., in [71], presented a smart Fog node, denoted “IoT Hub”, placed at the edge of the network, which enhances the capabilities of the network. The proposed IoT Hub provides functionalities such as border router, proxy, cache, and resource directory. Moreover, the IoT Hub uses several communication protocols on different layers in order to enhance the interoperability of the system. These are only some of the proposals for heterogeneous service discovery functionalities at the edge. Unfortunately, the majority of these works only on specific ecosystems either Fog or MEC environments. In contrast, this thesis proposes services functionalities for fully-integrated 5G networks with the goal of creating an edge node suitable for heterogeneous IoT devices and for dynamic 5G networks.

2.2.6 Machine Learning at the Edge

Machine Learning (ML) techniques are already widely used across a variety of domains to extract useful information from large-scale data. More recently, distributed ML solutions have been employed in Edge Computing scenarios known to have a three-layer architecture that includes the cloud, the edge layer, and the device layer. In fact, it is not practical to move all generated data to a centralized cloud in order to run the ML algorithm on it. The network communication would result the major bottleneck for this scenario. A recent approach to overcome this is the geo-distributed ML approach (Gaia) [72] that employs an intelligent communication mechanism over the network to efficiently utilize the scarce bandwidth while retaining the accuracy and correctness guarantees of ML algorithms. Furthermore, Gaia is general enough to be applicable to a wide variety of ML algorithms, without requiring any changes to the algorithms themselves. Finally, Gaia introduces a basic approach to efficiently run ML algorithms on a distributed architecture (edge-enabled architecture is also included) which may become a general guideline for a wide range of ML systems. Motivated by this, many systems start to leverage the Edge infrastructure both for the possibility to have computation close to the data source and for the freshness of the data. For instance, the work described in [73], support strategies for the allocation of computational resources using deep reinforcement learning in Edge Computing networks. Similarly, Chandakkar et al. [74] proposed strategies for re-training a deep neural network (DNN) in an Edge Computing infrastructure. Focusing on Edge Computing in Industrial IoT environments, only recently researchers started to exploit ML or intelligence at the edge of the network for predictive analysis or manufacturing control. In particular, the work by Raileanu [75] proposed an architectural framework for gathering heterogeneous data from the shop floor and aggregating them at the edge of the network. Successively, those data are sent to the cloud control platform that hosts a control system in charge of operation optimization, execution, and monitoring. Another work that leverages Edge Computing in IIoT is presented in [76]. In that work, the authors propose an architecture of edge computing for IoT-based manufacturing. Despite they basically analyze the role of Edge Computing in an IoT-based manufacturing system, the paper explores the idea to run ML at the edge nodes to utilize real-time data to make predictions and to subsequently update the knowledge base. These few proposals not fully exploit the ML functionalities at the edge. Unfortunately, the majority of these works focus only on running ML algorithms at the edge. In contrast, this thesis proposes architecture solutions for fully-integrated 5G networks with the goal of supporting Industrial IoT devices and managing ML algorithms at the edge for monitoring them.

3 5G-ENABLED EDGE (5GEE)

This chapter is the core part of this dissertation, it will present the model of the proposed 5G-Enable Edge architecture by discussing the differences and synergies between the two prominent models in the field of Edge Computing such as MEC and Fog Computing. In particular, this chapter addresses the problem of efficiently combine MEC and Fog in a unique architecture by proposing novel design guidelines for a possible implementation blueprint. Also, this architecture will be used for the rest of this dissertation as the baseline architecture on the following researches. Finally, we will conclude with a discussion of two relevant use cases in order to show the need for 5G-Enable Edge infrastructure and the open technical challenges.

3.1 Fog vs MEC

The main goal of this research is to propose a novel and integrated architectural model for the design of new 5G-enabled supports. The two elements that have been identified as key enabling components in order to fulfill the final goal, are Fog Computing and Multi-access Edge Computing. MEC and Fog focus on (and permit to increase) the quality and performance of several cloud-assisted device services, but we claim that these models still face, each of them separately, some non-negligible incompleteness and weaknesses. For instance, starting with the MEC, the number of employed edge nodes is generally limited, since each node introduces additional costs of operation for supported services, such as deployment, maintenance, and configuration costs for telco operators. Moreover, MEC typically works in infrastructure mode, being unable to easily leverage the resources available in surrounding devices at runtime: once MEC edges are deployed, they are rarely and hardly re-deployed (high cost of re-configurations) in other positions and this might be highly inefficient, e.g., when service load conditions significantly change during provisioning, such as during specific time slots, maybe with daily, weekly, or yearly patterns. On the contrary, Fog being more decentralized architecture (at least from a control/management perspective) makes it more flexible, at the same time it complicates its management and the possibility to leverage the monitored context (e.g., resource usage and availability) typically available in infrastructure-oriented MEC telco environments. In addition, Fog use cases are tailored mainly for resource-poor devices and sensing scenarios, and so SGs are typically unable to host heavy computations, such as in the case of a video surveillance service that monitors an environment with smartphone cameras

capturing photos/videos of the surroundings and that exploits face recognition to trace suspicious users' movements. To overcome such limitations, we claim the need for new solutions able to bring the best of the two MEC and Fog approaches by integrating them into a unique and fully-converged 5G architecture. As stated in the previous chapter, only a very limited number of seminal works have explored the mutual advantages in the joint synergic exploitation of these two classes of solutions. However, before starting to explain the proposal, we have to understand the differences and the similarities between MEC and Fog systems.

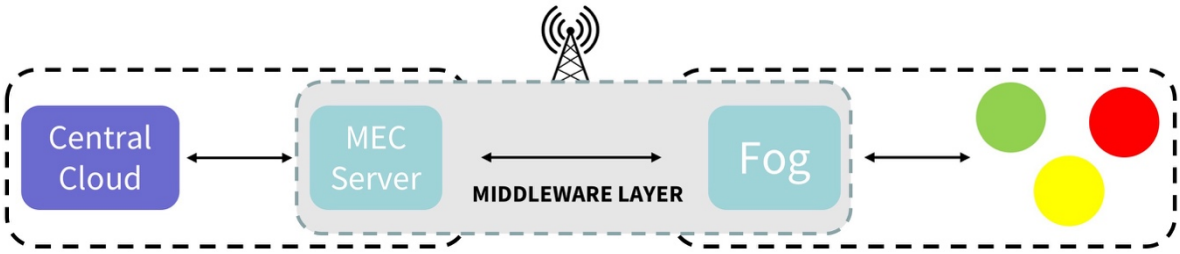


Figure 3.1 Architecture of MEC and Fog Computing

As illustrated in Figure 3.1, the architecture of MEC and Fog share many similarities, indeed they adopt the same distributed architecture skeleton that consists of a *Central Cloud*, a *Middleware* (middle) *layer*, typically deployed at the edge of the network, and *end devices* that might connect either directly through mobile heterogeneous networks, as it is usual for the Fog, or through a wireless access points, such as for cellular mobile nodes connected through a radio access network. The first difference to be highlighted is that the two models reside in two distinct network zone, which means existing within different administrative boundaries. The MEC infrastructure resides at the edge of the operator infrastructure. The operator owns and manages the infrastructure but not the devices. Figure 3.2 illustrates the MEC administrative boundaries. In terms of functionalities, MEC servers are usually implemented by the operator to enrich their network infrastructure with new services.

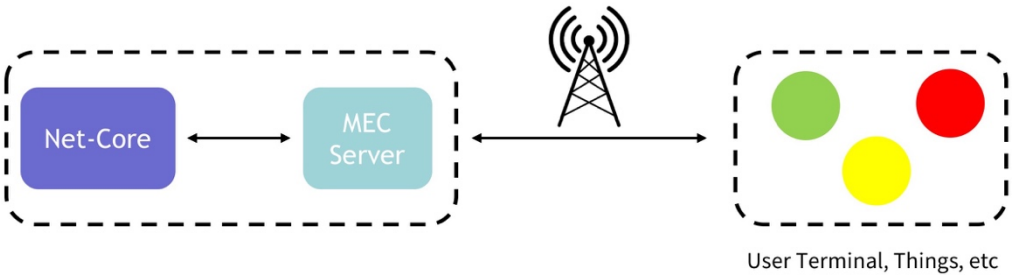


Figure 3.2 MEC Boundaries

On the contrary, the Fog infrastructure resides on-premises and at the edge of end-system infrastructure. The Fog infrastructure, as well as the things, are often owned and managed by the same authority, i.e. smart factory, smart grid, etc. Figure 3.3 illustrates the Fog architecture administrative boundaries. In terms of functionalities, the Fog architecture is more oriented to industrial Smart Gateways (SGs) allowing things to execute services and provides connectivity to the underlying Mobile Heterogeneous Networks (MHNs).

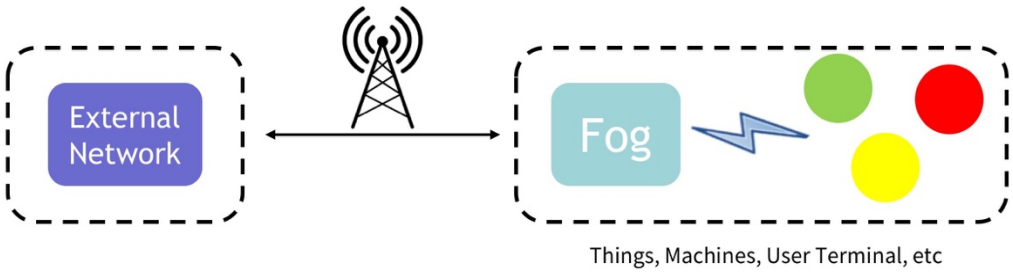


Figure 3.3 Fog architecture Boundaries

In conclusion, this thesis proposes a smart integration of MEC and Fog approaches by leveraging Fog, **on-premises**, and MEC at the **edge of the network**; we claim that this will be the ideal situation for demanding IoT applications. Figure 3.4 shows the resulting architecture.

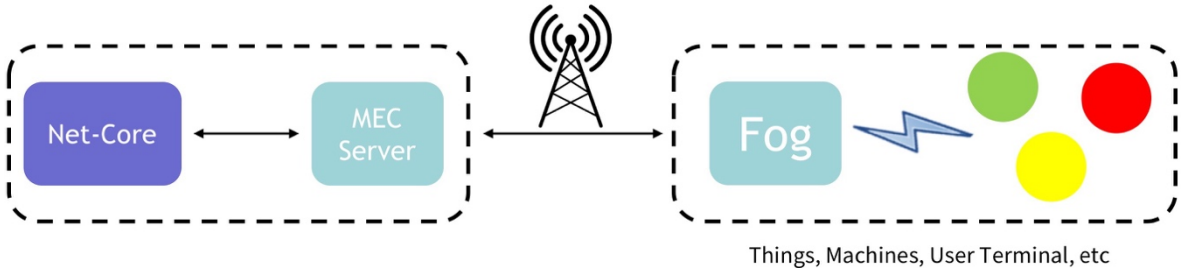


Figure 3.4 MEC and Fog combined architecture

3.1.1 MEC and Fog functionalities

Focusing on the crucial *Middleware layer*, for both architectures, it provides a set of functions close to the edge. Those features are sometimes similar, other times different and complementary, and can be seen as enablers for the development of new 5G and IoT service scenarios. Starting with MEC, at the middleware layer we can find MEC servers, and based on existing literature, we have identified the following main MEC features (we identify each of them with an incremental id MFi, and for each of them we report a brief description of its main pros/cons):

Table 3.1 List of MEC features

MF1	*Execution of resource-intensive applications*. This function enables the execution of telco multimedia and mobile applications close to the edge to grant ultra-low latencies below 1ms (one of the main 5G requirements). The main limitation is that it usually does not support the sharing of resources across different nodes in the same locality.
MF2	*Context data services*. This function, typical of telco infrastructures, allows the MEC node to retrieve context information about the Radio Access Network (RAN) it is logically linked to, such as cell load, user location, and allocated bandwidth in a very efficient way. This function is highly beneficial for the development of new classes of context-aware services.
MF3	*Pre-processing of data*. This can enable, for example, the real-time video analytics scenario detailed later as a relevant use case, where video data is processed by MEC servers and the application can detect and notify only specific events to the central cloud. This can help to guarantee required privacy and to avoid useless transmissions of huge amounts of data across the network via locality-based proximity processing.
MF4	*Caching of multimedia contents*. This function enables contents caching, i.e., one of the basic building blocks to minimize round trip time and to maximize throughput for better quality, for example in multimedia content streaming. In the currently available proposals, this is typically realized by a single MEC node; it is envisioned that it could benefit from leveraging also storage resources of other MEC nodes and mobile devices in the locality.
MF5	*Resource management*. This facility enables the reservation of slices of resources along the whole end-to-end service path including the MEC node. To be efficient, that typically requires also to predict incoming loads and observed patterns with a relatively good approximation. Resource management is actually adopted only for the data plane; to grant high-quality levels, an interesting extension that has just started to be investigated in the related literature is about the introduction of resource slicing also at the management plane.
MF6	*Mobile offloading*. This facility has been originally proposed to operate cloudlets by enabling the remote execution of services (or pieces of service) as mobile node companion applications at MEC nodes. How to automate the decision of which part(s) of the application to offload according to the actual context and the characteristics of the application is still an important research direction. In addition, novel forms of offloading, considering also the opportunity to dynamically migrate functions from MEC nodes to the global cloud start to be investigated, for even greater flexibility at provisioning time.

Instead, the Fog architecture is usually used for helping heterogeneous devices, such as sensors, actuators, and smartphones to overcome cloud computing limitations. In order to guarantee/support differentiated quality levels, we have identified some SG features (we

identify each of them with an incremental id FFi, and for each of them we report a brief description of its main pros/cons):

Table 3.2 List of Fog features

FF1	*Mobile Heterogeneous Network*. This facility enables the automatic building of network topologies with heterogeneous devices that exploit (possibly multiple) heterogeneous forms of wireless connectivity. In addition, SG may allocate them network resources in order to support some forms of quality management over the provided connections.
FF2	*D2D communication support*. This support function allows either to communicate in D2D modality or facilitate and make more efficient the D2D-based interactions of devices in the Fog node locality. For instance, SG may work as a cluster head in a peer-to-peer collaborative network of devices that are directly under its local control.
FF3	*Context data services*. This function is complementary to the context data service function in MEC. In fact, MEC context data service focuses on the context information available at the infrastructure side; instead, in Fog environments, this function helps to maintain and to know who is in the network, but also to characterize the sensed (physical) surrounding environment.
FF4	*Storage and aggregation of data*. This facility allows storing data collected from heterogeneous sensors or devices. Considered the requirements on reliability and durability of data in a given locality, after an agreed period, SG can synchronize the whole (or only the significant part of the) locally aggregated data to the cloud.
FF5	*Discovery/Registry of services*. In Fog environments, in order to support the underlying IoT and D2D models that are typically highly decentralized, it is necessary to include and offer service discovery/registration functions at the SG level, not at the logically centralized cloud as typically offered in 5G telco infrastructures. The final goal is always to keep track of all devices and services available in the surrounding, where the surroundings are generally defined in terms of a single locality (multiple adjacent federated localities can be anyway considered as well as a suitable scope in some application domains).
FF6	*Execution of resource-intensive applications*. Similarly, the fog layer is also useful to support the execution of applications with low latency and high energy consumption requirements. However, in fog, the support solution is typically tailored for a specific class of sensing applications (rather than multimedia service oriented as in MEC).
FF7	*Service Handoff*. This facility allows the transparent service rebinding of end-user devices as they move from one (old) SG to another (new) one, possibly while maintaining continuity of service provisioning. Let us note that this function is not available in MEC nodes because 5G/LTE handoff management is completely on behalf of other inner infrastructure components, thus becoming transparent for the pure MEC functionality.

FF8	*Mobile offloading*. This support function in Fog has the goal of granting low latency and short reaction time; moreover, due to the poorer SG resources, the decision about the splitting of the application between the mobile node and the SG is typically rather static and proactively taken at application deployment time. At the same time, mobile offloading in Fog is typically intertwined with service handoff due to the more dynamic and decentralized nature of Fog environments.

Moreover, behind the proposal, we have the idea of properly splitting support functionalities in order to make them more easily and efficiently usable by final user applications; this is a relatively new idea in MEC–Fog computing integrated scenarios. In particular, we propose to put a service layer on top of all proposed support functionalities; this also means that we need to dynamically manage such services in order to link them with the used and composed functionalities at the lower layer.

3.2 The 5GEE architecture: Model and Design

Figure 3.5 presents the general architecture of our proposal. We aim to merge the MEC and Fog architectures into a single infrastructure with the combined support features. Our architecture consists of multiple 5GEE nodes, which expose and allow us to combine MEC and Fog functionalities. As listed in the previous section, some MEC and Fog features are similar in existing proposals. In this work, we decide to smartly combine those functionalities by minimizing overlapping features and by exploiting synergies. In 5GEE architecture, we include the following features explained in Table 3.3.

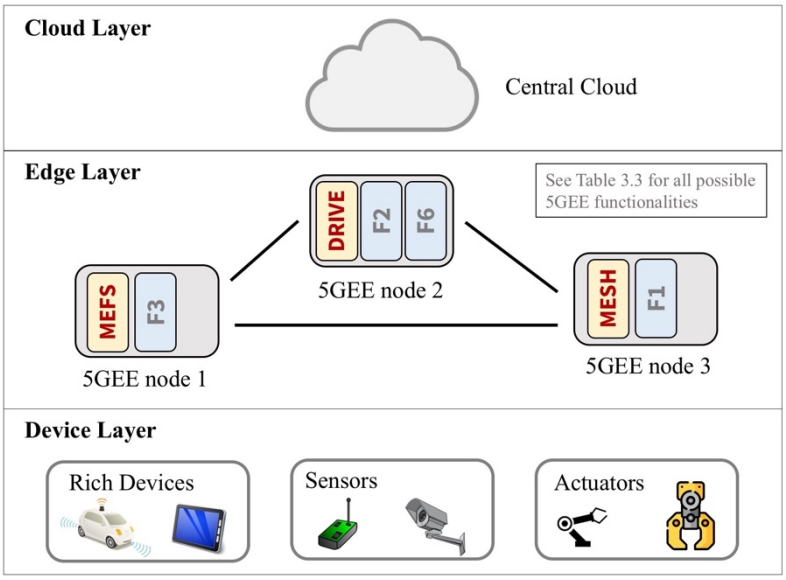


Figure 3.5 General architecture of the proposed 5GEE integration

Table 3.3 5GEE functions and motivations

FUNCTION	MOTIVATION
MF1 and FF6: **F1 – Execution of resource-intensive applications**	Both functionalities help devices to execute resource-intensive applications. Despite MEC limitations, our architecture is able to run third-party applications.
MF2 and FF3: **F2 – Context data service**	MEC has network-based information about context, while fog solutions tend to have better information closer to the edge. We combine these aspects in order to obtain an all-context-aware support service.
MF3 and MF4: **F3 – Pre-processing and caching of multimedia contents**	MEC platforms are more multimedia-oriented than in Fog solutions. Our 5GEE proposal combines all functionalities from the two paradigms in order to help devices to achieve better quality, especially for multimedia services.
MF5: **F4 – Resource management**	This is a central functionality for enabling efficient handoffs, in particular in scenarios of service continuity. Proper management of resource slicing is a needed enabler for allowing runtime service migration between 5GEE nodes.
MF6 and FF8: **F5 – Service offloading**	We combine together these two different functionalities about service offloading. The result is that the derived service offloading feature can move to process either to the 5GEE layer and/or to the global logically centralized cloud.
FF1: **F6 – Mobile Heterogeneous Network**	This feature helps our architecture to communicate with all sets of dynamically discovered devices, which may exploit different wireless connectivity and discovery protocols.
FF2: **F7 – D2D communication support**	D2D communication is recognized as one of the emerging technologies of central interest for increasing the scalability of 5G architectures. In our solution, the integration of D2D mechanisms and protocols is also the key to locality identification and detection, as well as for enabling localized communications with no load over the MEC/Fog-to-cloud trunk.
FF4: **F8 – Data storage and aggregation**	See the FF4 description above (Table 3.2).
FF5: **F9 – Discovery / Registry of service**	The addition of discovery/registry facilities for available resources and services in a locality is of central relevance, even if not typically covered by MEC/Fog solutions at this stage of their implementation evolution. Of course, this kind of support goes into the direction of leveraging locality-based interactions that are autonomous with regards to the central cloud. The integration of existing and heterogeneous discovery protocols is envisioned as appropriate to facilitate the path towards adoption.
FF7: **F10 - Handoff**	This is crucial functionality for our architecture, given our specific interest in the support of mobile IoT applications. In order to guarantee continuity of

	service, the envisioned feature has to exploit predictive mechanisms (also based on profiling), proactive management of involved resources, and live but lightweight migration.

3.2.1 Management & Deployment Issues

By focusing on the management and deployment of the integrated architecture features, we foresee that services and network functionalities will be mainly implemented as software components executing on standard operating systems by leveraging the latest achievements in these fields. As stated in chapter 2, NFV and SDN are basic enablers for the new scenario by simplifying the way network resources and functionalities are deployed and controlled across distributed locations. Moreover, in order to manage services and functionalities, we need a service orchestrator to take over the deployment complexity across the whole infrastructure. Many recent research activities are in this direction: in fact, it starts to be recognized that having an orchestrator allows architectures to overcome deploy challenges. In this work, we proposed to use an open source implementation of the MANO framework [40] named Open Baton [77]. In the very recent period, there was an increasing interest in having Orchestration solutions compliant with the MANO information model in order to reduce the fragmentation between different existing management platforms. The orchestrator selected is Open Baton, supported by the Fraunhofer FOKUS Institute and the Technical University of Berlin, implementing a compliant ETSI NFV MANO Framework using the Information Model specified in the initial phase of the standardization; its architecture is shown in Figure 3.6. MANO, as well as Open Baton, provides the management and the orchestration of all resources including computing, networking, storage, and virtual machine resources. The main focus of MANO is to allow flexible on-boarding and sidestep the chaos that can be associated with the rapid spin-up of network components. MANO is composed by three main functional blocks: i) NFV Orchestrator, responsible for on-boarding of new network services (NS) and virtual network function (NFV) packages; ii) NFV Manager, oversees lifecycle management of NFV instances; iii) Virtualized Infrastructure Manager (VIM), controls and manages the NFV compute,

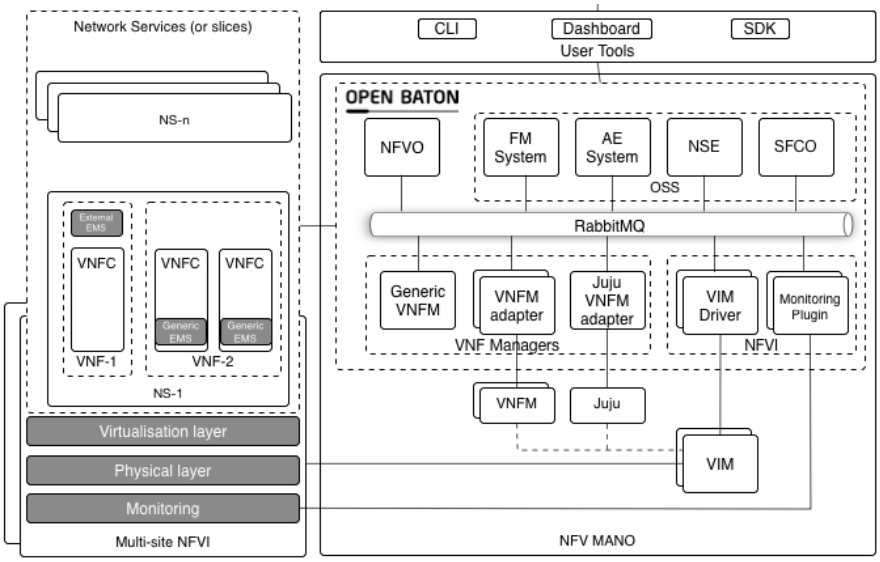


Figure 3.6 Open Baton Framework

storage, and network resources. Finally, the choice of using Open Baton is led by the possibility to use container development tools, i.e. Docker, as the VIM.

3.2.2 Implementation Blueprint

Here, we describe a possible implementation of the 5GEE architecture designed according to a three-layer architecture, based on the extension of emerging MEC - Fog technologies, by integrating proactive and/or reactive container migration. The three considered layers are:

- **Devices layer**: Our devices layer consists of all the endpoints that need to perform high-resource demanding executions of mobile services and do not have enough capabilities to do that. For instance, this includes heavy image and video analysis/processing that can be performed uniquely with computationally intensive techniques that require resources beyond the capability of general-purpose mobile devices, at least taking into consideration possible application-specific requirements on response time. Our solution fits a very wide spectrum of heterogeneous mobile devices, with the only constraint to run Android OS, or basically Linux-based OSs (in the future we can extend the approach to the other relevant OS, such as iOS). Generally, mobile devices often do not have enough capabilities to satisfy strict requirements on response time, in particular, if considering their possible immersion in hostile environments; therefore, it is a must that they have to delegate most analysis functions to the 5GEE layer.
- **Edge layer**: Our edge layer consists of a set of 5GEE nodes geographically distributed. In particular, the 5GEE node consists of two primary components: i) the Docker

Container technology, one of the most promising and complete open-source project to realize services at the edge of the network, that allows moving and orchestrating containers near mobile devices; and ii) a service orchestrator based on Open Baton that provides containers deployment of services and functionalities. The 5GEE node runs a Docker daemon instance, while services and functionalities are implemented as containers in execution on it. We propose this option because containers can relevantly reduce resource computation on the edge because container-oriented solutions start to be supported by Open Baton, and containers are more easily movable across our 5GEE infrastructure. Moreover, 5GEE nodes can be provisioned in either a proactive or a reactive way. Proactive provisioning allows to minimize and automate in an efficient way virtualized function migration, by pre-loading the needed functions in advance on the target 5GEE node that, presumably, will be the next visited by the served user; this is managed before receiving explicit migration requests, thus limiting the costs in terms of unavailability and performance during the procedures of mobile device handoffs and associated container data volume. The central cloud layer (third layer below) triggers virtualized function migration. On the contrary, reactive migration is triggered when a mobile device requests explicitly to move a virtualized function.

- **Central Cloud layer**: The cloud layer is used to assist the 5GEE intermediate middleware; in our implementation, this assistance is mainly focused on proactive analysis about users' movements and to guarantee system resilience. In addition, the cloud layer is used during the initial service setup operations and interacts with a service orchestrator in order to load the needed services and functionalities over the interested subset of 5GEE nodes, by need at service provisioning time.

Figure 3.7 shows in detail the implementation of the 5GEE node. All functionalities discussed before are developed as a Docker container and will be managed by the Open Baton orchestrator. The Service Layer allows us to expose APIs both towards the devices layer and within it.

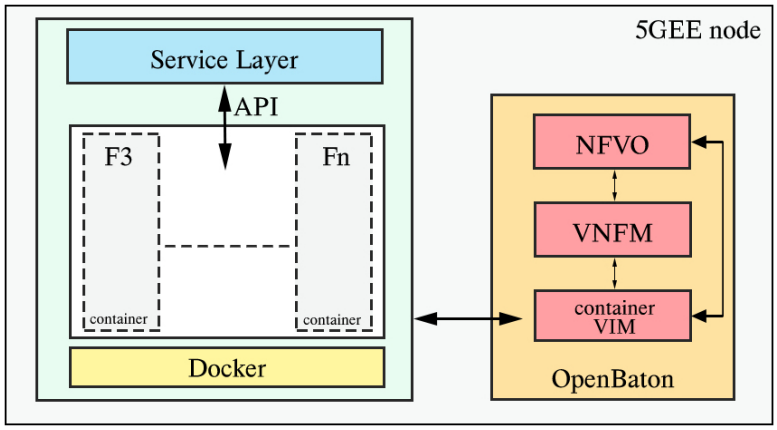


Figure 3.7 Implementation blueprint of the 5GEE node

3.3 5GEE Use Case and Discussion

After sketching the general concept and architecture in the previous section, here we practically show that the previously described features may be useful in two main use cases, and then we report a discussion of open technical challenges and most promising research directions. The two use cases both relate to Mobile Crowd Sensing (MCS).

3.3.1 Use case 1

Alice participates in an MCS project (e.g., the UNIBO crowdsensing project called ParticipAct [78]). The increasing popularity of smartphones, already equipped with multiple sensors from GPS to microphone and camera, paired with the inherent mobility of their owners, enables the ability to acquire local knowledge from the individual's surrounding environment through mobile device sensing properties or even from the individual itself. Participants collect passive and active data that are processed and shared with other participants and with the central cloud. Back to Alice, assume that the connectivity in her city is provided by a new 5GEE node. Normally, the 5GEE node is set to provide telco functionalities such as LTE/5G connectivity and a few additional network support functionalities. On a particular day, when a new MCS campaign starts, the available functionalities over 5GEE nodes in given localities have to be updated in order to support the campaign. This is done by an orchestrator, such as Open Baton, that dynamically loads the needed functionalities as overlays on top of the already installed basis (i.e., base image plus previous possible overlays). Which support features does Alice need? Analyzing the scenario, Alice has to use her smartphone to collect data, process them, and then upload their aggregated summary to the central cloud. So, she would benefit from F8

(store and aggregate data functionality) as well as from F2 (context data service functionality) in order to perform context-aware campaign tasks. These are just the basic functionalities that an MCS campaign needs. But we may also need an F7 feature (D2D communication support) for collaborative crowdsensing, or F10 to effectively support device mobility. That support features help to overcome many cloud computing problems and helps Alice to safeguard power consuming of her smartphone and achieve a better user-perceived end-to-end final quality. Figure 3.8 depicts how the proposed 5GEE architecture works in a crowdsensing context.

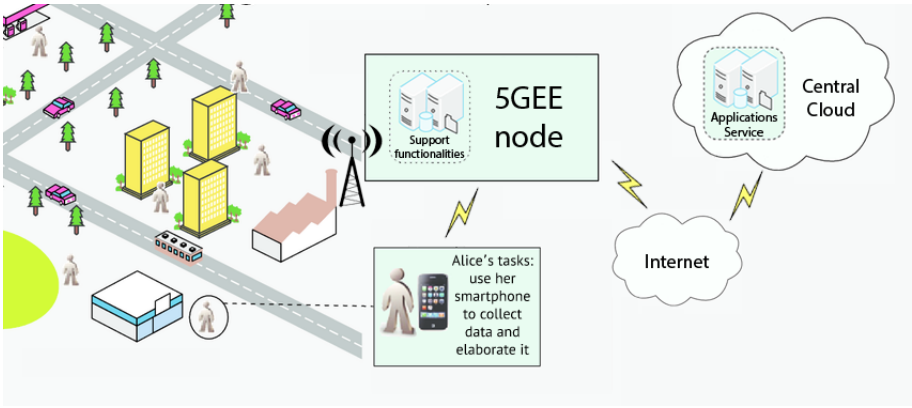


Figure 3.8 5GEE Architecture in an MCS monitoring scenario

However, in a classic Smart City scenario composed by sensors and actuators, we need to maintain a given quality level, for example, to ensure low latency response time for sensor configuration operations and/or for control commands over actuators. The support of quality requirements is with no doubt still an open technical challenge in the field; however, solutions based on container-oriented virtualization can help to reduce latency time: for instance, some few seminal works have started to exploit Docker containers in such IoT scenarios [79].

3.3.2 Use case 2

On the contrary, let us suppose that, in order to complete his MCS task, Bob has to share video streaming to other people to show what happened in his zone (see Figure 3.9). In the last past years, mobile video streaming traffic has increased considerably and represents a relevant bottleneck for mobile traffic data (e.g., YouTube and Netflix services) [80]. To overcome this challenge, edge caching has been identified as a natural solution. Videos may be cached in the 5GEE node, so that demands from users can be accommodated easily without transmission from remote resources, e.g., on the global cloud. This approach contributes to reducing central cloud usage and content access delay. Back to Bob's problem, he wants to share his multimedia

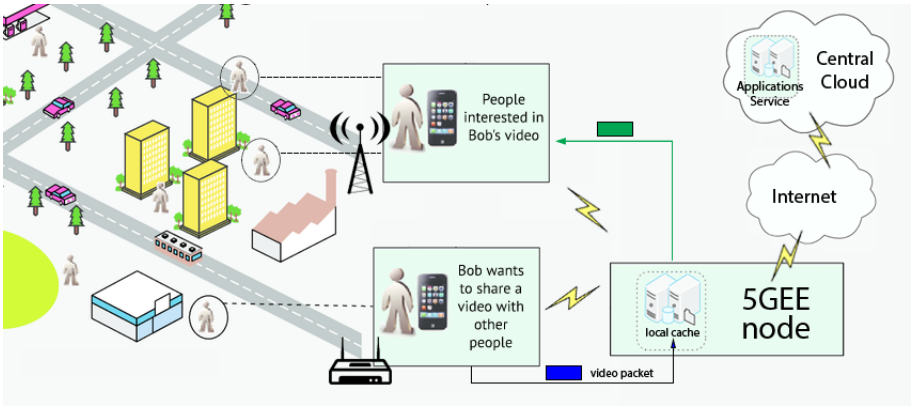


Figure 3.9 5GEE in an MCS content sharing scenario

content with a neighbor node in his locality. As it happens in the previous case, the connectivity in the city is provided by a new 5GEE node that normally has basic telco support features. In this case, which functionalities does Bob need for his active MCS participation? We may probably benefit from i) F3 (pre-processing and caching of multimedia contents) in order to store video contents in a local cache and ii) F1 (execution of resource-intensive applications) to enable the execution of possibly heavy codecs that may be necessary to adapt video contents for a set of device renderers currently present in the locality. Figure 3.9 shows how our 5GEE architecture works in this kind of application scenario.

3.3.3 Discussion

As discussed above, smart and efficient component/service migrations are crucial enablers in such a context; their availability for converging MEC/Fog computing is still a very open and technically challenging point. In particular, traditional virtual machine migration approaches that have been applied to cloud computing (which abound in the related literature) are demonstrating to be unsuitable for the scenarios envisioned in this thesis. Furthermore, the contribution of this dissertation would be to propose components and functionalities for 5GEE architecture including an efficient service migration (handoff) functionality. Therefore, in the following chapter, we will present the MESH (MEC Service Handoff) framework that proposes a novel and smart technique for service migration at the edge. Another important issue that is still not addressed is that each paradigm (MEC and Fog) works within a specific ecosystem. So, new functionalities should work for both systems; it means develop functionalities as more general as possible trying to cover the requirements of the two models. The DRIVE framework goes in this direction, in that work (see chapter 6.1) we will present a service discovery functionality for 5GEE architecture where its architecture is composed by different layers to

allow us to cover the requirements of MEC and Fog. Lastly, this thesis proposes a task offloading system able to work at the edge of the network. Despite a large number of solutions for task offloading proposed in the recent literature, task offloading at the edge is still an open issue due to the mobility and resilience constraints. In chapter 5, we will propose the MEFS task offloading framework that is able to solve mobility and resilience constraints.

In conclusion, we have presented 5GEE, a new architecture model to ease the provisioning and to extend the coverage of traditional edge computing approaches by bringing together the best of MEC and Fog research areas. The cornerstone of our proposal lies in the ability to dynamically orchestrate all functions and needed resources at 5GEE nodes without assuming any predefined configuration. Rather, 5GEE leverages the latest achievements in the two main areas of cloud computing and containerization, while exploiting MANO to automate and fasten the tailoring of 5GEE node installations according to the specific requirements of final users and currently supported applications. As mentioned above, the rest of this dissertation will present new components and functionalities designed for working within the principles of 5GEE architecture. Please note that from now on we refer the term Edge Computing as a general converging MEC and Fog architecture with the same principles discussed for 5GEE architecture.

4 MOBILE EDGE SERVICE HANDOFF (MESH)

In the previous chapter, it has been addressed the integration of MEC and Fog Computing capabilities aimed to propose a new architecture that supports dynamic service provisioning. Due to the users' high mobility, one functionality discussed in the previous chapter, namely Handoff, needs to be tackled. This chapter will address the problem of service migration by presenting differentiated approaches that leverage service characteristics or not. In this way, automatically the system selects the appropriate application-aware approach in order to reduce the total migration time. Also, this chapter will present a specific application-aware algorithm, that selects data to be moved proactively, based on data access frequencies. Finally, we will present a complete set of experiments, for both application-agnostic and application-aware migration.

4.1 Motivation

The main goal of this research is to realize an edge-based framework capable of overcoming one of the hottest still open issues of Edge Computing also known as service migration. The reason for this work is that generally edge nodes have small network coverage and it is very common that a mobile node changes its connection during its path. Due to limited coverage of a single edge node, users' mobility could affect the degradation of network performance reducing the QoS or in some cases causing the interruption of the edge service. Thus, in order to guarantee service continuity, it is necessary to support efficient service/data migration between edge nodes. However, at the current stage, there are several heterogeneous virtualization technologies and migration strategies and some of them might not be practical when used with resource-poor edge devices (e.g. Raspberry Pi). Moreover, future 5G networks will be composed of heterogeneous devices, such as home gateways and MEC micro-servers that do not host the same resources. That makes service migration management a very complex task; for instance, heavy-computation services should be migrated to high powerful micro-servers rather than to poor Fog gateway nodes. Focusing on existing solutions, most seminal efforts have been focused on the concept of Live Migration of Virtual Machines (VMs) [81] to guarantee the lowest possible downtime of the service. Live Migration considers service migration as the stateful migration of services where a service contains internal state data of users. After the completion of the migration, the service resumes exactly where it had stopped before. To ensure this, complex mechanisms have been proposed including the main two

variants called pre-copy and post-copy. Pre-copy pushes most of the data to the destination host before stopping and migrating the VM [81]. Post-copy pulls most of the data from the source host after resuming VM at the destination host [47]. Successively, with the diffusion of container technologies such as Docker, most of the research efforts have been focused on service container migration. This is justified by several experiments conducted to compare the performance of VMs and Containers [82]. Since containers are a more lightweight virtualization option, several companies started using them to develop applications. Today, containers are largely used for edge-based services due to their adaptive characteristics including lightness and portability. Nevertheless, only very few proposals have been started to target the technological advances associated with container migration in place of VM. Some proposals tried to use container virtualization technology as the object of the live migration (container state migration). Other efforts, instead, have focused on service migration by leveraging Docker technology. Therefore, at the current stage, Edge Computing does not have a standardized fast service/data migration support. To tackle this problem, this research focuses on reducing the total time of service migration by leveraging service characteristics. In the following, we show how to design a fast handoff schema that exploits the characteristics of the service. At the heart of our schema, there is Docker which allows the separation between data and service containers. The proposed framework supports differentiated granularity levels for container migration based on characteristics of service/data components between off-the-shelf and currently deployable edge nodes, according to the follow-me model, i.e., in response to user handoff at service provisioning time. In particular, the proposed work has the following primary innovation elements and features, which we claim provide a non-negligible contribution to the advancement of the literature in the field. First, it enables either application-agnostic or application-aware approaches for container migration. In the application-agnostic mode, our framework can perform the migration without having any visibility of the application behavior (named “cold migration”). Dually, in the application-aware case, the framework can fully exploit application-specific knowledge to determine which data should be migrated proactively in order to minimize latency and to optimize the usage of possibly limited resources, e.g., inter-edge bandwidth and edge storage. Second, an efficient way to take advantage of data characteristics has been investigated to reduce application-aware migration times. We have developed a Decision Module that contains a set of mechanisms for carrying out application-aware migration based on data change probability. For instance, we proposed a mechanism that associates with data with lower access frequencies that can be proactively moved, while it postpones the migration of data with higher access frequencies. Moreover, our

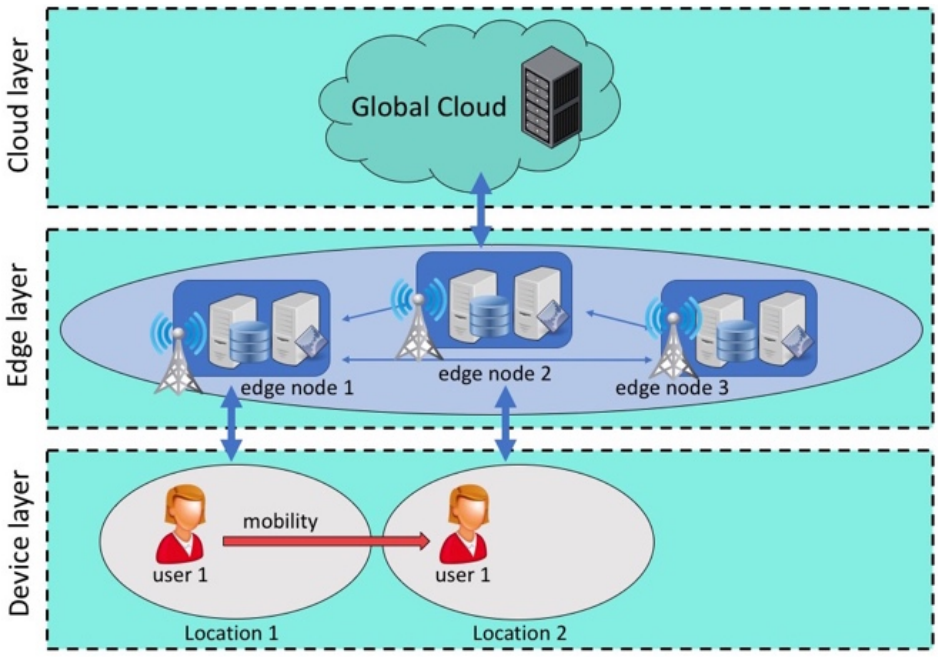


Figure 4.1 Logical vision of an Edge Computing architecture, with lightweight coordination of edge nodes

framework allows developers to dynamically specify the list of primitives to be triggered by our handoff procedure. Third, our framework also guarantees data consistency between edge nodes after the handoff: if some data moved proactively to the new edge node have been changed during the handoff period, our framework automatically reconciles those data. Last, our framework addresses the problems of heterogeneity and energy management at edge nodes by defining an affinity relationship that depends on service characteristics. This solution allows our framework to decide the best target node toward which to perform the handoff in terms of needed resources and energy consumption. We consider as edge node both resource-poor devices, including Raspberry Pi acting as Fog nodes and powerful computers used as MEC node.

4.2 Background and Modeling

To better understand our work, the following section provides all the needed definitions for the involved technologies and methodologies and an overview of our proposed design guidelines.

4.2.1 Background and multi-layer Container Migration

The need for service/data migration in the Edge Computing environments can be illustrated with the example shown in Figure 4.1. Here, user1 is initially connected to edge node1 and has some of her service/data components hosted on virtualized resources on it; the single-hop local

connection between user1 and edge node1 favors low latency and better scalability through localized provisioning. However, after some time and during service provisioning, user1 may move to a location that has direct connectivity to edge node2 (node2 locality). In several application cases, it can be beneficial to migrate user1's service/data components from edge node1 to edge node2, in order to continue to serve user1 with virtualized resources in her current locality. Note that edge nodes should also benefit from some forms of lightweight coordination among them, even without passing through the global cloud, based either on their hierarchical organization (typically when scalability and wide-scale deployment are primary objectives) or on peer-to-peer interactions. As already stated in the motivation section (section 4.1), the term handoff in Edge Computing deployment scenarios is associated with the concept of service migration, the related literature typically uses live migration to identify the process of VM migration in datacenters at the cloud layer. The primary goal of live migration is to minimize the downtime of service during which a migrating VM instance is not accessible; in our modeled scenario of Figure 4.1, we originally use the live migration term also to indicate VM/container migration between edge nodes. In fact, live migration typically allows a VM instance to continue execution at source edge also during handoff, by using a distributed file system (accessible to edge nodes) and by transferring the state, possibly modified during handoff, to the destination host in background. As the final step, when the VM instance is suspended at the source, all its remaining modified memory states are sent to the destination, and the execution is transparently resumed. As already stated, the service migration term resembles live migration in Edge Computing architectures, but there are a few not-negligible differences between the two concepts that start to be recognized in the related literature [83]:

- They target different performance metrics. Service migration aims to reduce the total time of the migration process; while live migration aims to reduce the downtime perceived by mobile users.
- Live migration needs to use a substantial amount of shared storage and memory, while in Edge Computing scenarios these resources are limited.
- In live migration, edge nodes should have enough resources to continue their processing work, while service migration cannot depend on the availability of rich resources at the edges.
- In the target edge node, the resources required by the ongoing service may exist; this can avoid unnecessary data transfer during service migration.

Table 4.1 Comparison between the terms service migration and live migration, as currently used in the existing literature

	Service Migration	**Live Migration**
Performance metrics	Total time of migration	Downtime time
Shared resources	NO	YES
Resources guarantee	NO	YES
Resource reuse	YES	NO

Finally, according to Y.C. Tay et al [84], service migration performs better in terms of requested time and resources if compared to live migration in case of stateless workloads. In that paper, in fact, the authors have distinguished among stateless and stateful VMs/containers: in this context, stateless means that any computation done on the old edge does not need to be executed again in the target edge node; instead, stateful is used to indicate that the computation that was aborted on the old edge node needs to be re-executed at the new edge. Our work is focused on service migration aspects, which can involve service components, data ones, or both. Furthermore, we distinguish between so-called layered services and monolithic services. Layered services consist of diverse layers, such as service and data parts, that could be managed as different separated blocks. Differently, monolithic services have to be considered by our framework as a single block, which internally includes service components, data components, and all associated resources. To better understand layered services, let us describe a simple example: a simple Python web application represents the service part, while a Redis database in which data are stored represents the data part. In this way, the service part and the data part could be managed as distinct blocks. On the contrary, monolithic services can be put into execution within dedicated and self-contained VMs as proposed in some recent literature: for instance, [7] have proposed a mechanism for edge-enabled handoff management based on VMs synthesis and migration to the closest edge nodes. However, the usage of VMs introduces non-negligible latency and overhead due to VM size and complexity. The exploitation of container-based virtualization techniques could reduce the above weaknesses, by pushing towards the opportunity of considering more layered services that can be de-composed in micro-services (to be containerized one by one). This research presents a novel approach for multi-layer container-based service migration by leveraging service characteristics. To achieve this, edge-enabled services in our proposal are built as Docker Containers composed by a *service layer* (acting as the ‘‘business logic’’ part of the service) and a *data layer* (representing the state stored and managed through the service layer) that should be managed as separate containers. The Data Container is used to persist data and could be managed by either a DBMS or a NoSQL

manager. In particular, we followed a general approachable to work also with no-database-based storage including general filesystems. In addition, our proposal is able to support two kinds of service migrations: *application-agnostic* and *application-aware*. Application-agnostic handoff enables the migration of the entire Data Container, as the data backup, without requiring any previous knowledge of the specific data software layer technology. Application-aware, instead, leverages service characteristics to extract and proactively transfer part of data to the target edge node in order to reduce service interruption. However, the latter mode requires partial visibility of some characteristics of the implementation of the Data Container, and for this reason, it is not usable for any kind of application. As said before, this work is based on Docker technology and at the current stage Docker technology does not provide any official migration tool, but, recently, few developers have constructed tools for specific versions of Docker. For instance, the work [57] supports Docker containers migration of docker version 1.9.0, and Boucher [85] extends the previous work to support docker-1.10 migration. However, both methods simply transfer all the files located under the mount point of the container root file system. More recently, also Docker provided some mechanisms not directly related to the migration but useful for this purpose. For instance, the *docker export* command enables users to create a compressed file from the container filesystem as a ‘‘tar’’ file. This compressed file can be copied over the network to the target edge node via file transfer and then imported into a new container via *docker import* command. The new container created in the target edge node can be accessed using the *docker run* command. One drawback of the docker export tool is that it doesn’t copy environment variables and underlying data volume which contains the container data. Another method based on Docker commands to move the container to another host is container image migration. For the container that has to be moved, its corresponding Docker image is saved into a compressed file by using *docker commit* command. Then the compressed file is moved to the target edge node and a new container is created with *docker run* command. Using this method, the data volume will not be migrated, but it preserves the data of the application created inside the container. Last, Docker provides a mechanism to save an image to a tarball which preserves the history, layers, and entry points via the *docker save* command; at the same time, it provides the equivalent command to load the image in the new host: *docker load*. Unfortunately, the aforementioned methods completely ignore the composition of the service while this work leverages it for smarter migration management. To verify this, we conducted preliminary experiments to migrate containers over different network connections. The experiments use one simple container such as Busybox and one application based on OpenCV for face recognition, to conduct edge task offloading.

Busybox is a software suite that provides several Unix utilities in a single executable file. It has a tiny file system inside the container. On the contrary, the face recognition application is an application that dispatches video streaming from mobile devices to the edge server, which executes the face recognition tasks, and sends back a specific frame with the name of the person. This container hosts a large filesystem to store all the images (i.e., more than 1 GB). Table 4.2 reports obtained preliminary results that show that migration can be done within 2 seconds for Busybox, and within 223 seconds for face recognition application. The network between these two hosts is a Wi-Fi connection with 40 MB/s bandwidth and further tested container migration over a 10 MB/s Wi-Fi network.

Table 4.2 Docker containers migration time (bandwidth 40mb/s, latency 0ms, and delay 0ms)

Application	**Downtime**	**Total Time**	**Total Size**
BusyBox	1.85 s	2.16 s	1,4 MB
Face Recognition	205.61 s	222.89 s	~ 1 GB

As previously stated, poor performance is caused by transferring large files comprising the complete file system, for instance, the total migration time of face recognition application (Table 4.3). This performs worse than the state-of-the-art VM migration solution. Migration of VMs cloud avoids transferring a portion of the filesystem by sharing the base VM images [7], which will finish migration within a few minutes. Therefore, we require a new tool to efficiently migrate Docker Containers, avoiding transmission of the entire container. This new tool should leverage the characteristics and composition of edge-enable services to transfer proactively part of the service data.

Table 4.3 Docker containers migration time (bandwidth 10mb/s, latency 1.5ms, and delay 40ms)

Application	**Downtime**	**Total Time**	**Total Size**
BusyBox	4.78 s	12.11 s	1,4 MB
Face Recognition	>1200 s	>1500 s	~ 1 GB

4.2.2 Design Guidelines for application-aware handoff

The goal of this work is to enable proactive, transparent migration of edge-enabled services (typically represented by both the data part and the service part). The idea of this method is based on the observation that not all data or records are used all the time. In fact, based on a

recent study [86], it was found that most data or records are stored, but rarely or never accessed after a certain time frame. Therefore, data or records can be categorized according to their access frequencies: least accessed data (cold data) and most accessed data (hot data). In our solution, we define a probability of data migration according to the data access frequencies, which means cold data are more inclined to be part of the proactive migration process. To do this, we first need to introduce an operation meter that, for each data or records chunk, calculates the total number of operations did until a certain time. The operation meter is defined as:

$$O_k = I_k(t) + U_k(t) \tag{1}$$

where O_k denotes the total number of operations did on particular data or record chunk (k) which is defined as the sum of the number of *insert* operations (I_k) and the number of *update* operations (U_k). Hence, by repeating the ahead formula (1) for all data it is possible to obtain the value of access frequencies defined as:

$$f_k = \frac{O_k}{\sum_i^n O_i} \tag{2}$$

where f_k represents the access frequencies of data or record chunk defined as the relationship between operations did on chunk k and the number of total operations did in all data. Finally, we define the migration probability assigned to each data chunk k as:

$$P(x) = \frac{1}{f} \tag{3}$$

As expressed in (3), the migration probability is defined as the inverse of the access frequencies. This means that data accessed often have a low value of migration probability, while data rarely accessed have a high value of migration probability. Thus, the goal of this work is to get the maximum benefit from the data characteristics in order to reduce the overhead and the service interruption during the handoff procedure. This requires us to calculate the access frequencies, as well as the migration probability before the handoff happens. Let us note that the decision on when, where and whether to perform the migration depends on many aspects, such as user mobility, user historical paths, resource availability at the edge nodes, and

so on. The Prediction Module (see next Section) guarantees the right execution of our service migration. This module is composed by two components: monitoring and trigger. The monitoring component monitors users' locations in order to predict their movement. Several monitoring strategies have been proposed in the literature, and the design of the Prediction Module allows it to work with any strategy. The trigger component is in charge of determining the appropriate time to initiate the handoff (both long-and short-term). This research identified two distinct handoff triggering strategies: a coarse-grained model (long-term), and a fine-grained model (short-term).

- *Coarse-Grained Model*. Typically, services running at the edge server have a limited period of validity, ranging from few minutes to few hours. The goal of this model is to predict well in advance the user movement in order to calculate data to be moved from one edge node to another. Most long-term handoff triggering algorithms proposed in the literature have taken into account both QoS of application and users' mobility traces. However, if users' historical path is available, the system can proactively predict the handoff timing and can early move service and data from one edge to another.
- *Fine-Grained Model*. The goal of this model is to establish with high accuracy when the handoff happens. In general, short-term handoff triggering algorithms are based on monitoring wireless indicators, such as Received Signal Strength Indications (RSSI) [87]. Otherwise, there are other works that take into account the QoS of application such as TCP throughput [88].

The proposed framework works with both long- and short-term handoff prediction. When a long-term strategy predicts the handoff, the Prediction Module notifies our Decision Module that starts to calculate the migration probability for each data chunk. In order to choose which data move proactively, we defined a probability threshold, statically or dynamically determined, in which our framework migrates all data chunks that have a probability value greater than the threshold. The appropriate value of the threshold is chosen based on the variability characteristics of the data. For instance, if the data have a high value of variability it would be better to use a high value of probability threshold (e.g., around 0.9-0.95). That's because, using a low value of probability threshold may cause problems in the reconciliation phase, in the sense that early migrated data chunks may result changed after the handoff procedure is completed. Instead, if the data have a low value of variability could be convenient to use a low value of probability threshold. Let us note that the correct value of the probability threshold may be decided dynamically -on the fly- in relation to the data characteristics. Once

defined a proper value of the probability threshold, when the Prediction Module predicts the handoff the Decision Module starts to migrate all data chunks that satisfy the threshold condition. Finally, when the handoff occurs the framework migrates all remaining data chunks (data reconciliation phase – step 9'' Figure 4.5) and then checks the data integrity. On the contrary, when a short-term strategy predicts the handoff, as specified before, fine-grained models detect the handoff more precisely than coarse-grained models. Thus, the system has better accuracy but less time to operate. For this reason, could be not possible to send all selected data chunks from one edge to another before the handoff happens; to avoid this, we have adopted a strategy named ''sequential execution'' that starts to sequentially migrate data chunks with the highest value of the migration probability until the handoff happens. Finally, when the handoff happens our framework migrates all remaining data chunks. Regardless of the model, once the handoff terminates the framework has to guarantee data consistency. To ensure this, our framework checks if data chunks sent proactively have changed during the handoff. If some data chunks differ, the framework reconciles them. To correctly check if all data chunks sent proactively are consistent after the handoff, our framework sends hash values of each data chunk, before and after the handoff, and checks if these hash values correspond. If the hash value of some data chunks does not correspond, we must resend those chunks.

4.2.3 Heterogeneity and Energy aspects

As stated in Chapter 3, converging MEC and Fog models have to address the heterogeneity issue of end-devices. In the same way, systems running on those models have to keep track of the heterogeneity of edge nodes. In general, edge nodes are heterogeneous devices spanning from powerful microdata servers (typical for MEC infrastructure) to general-purpose resource-poor gateways (typical for Fog Computing infrastructure). Therefore, if more than one edge node is available in a locality, would it be convenient to select the best target edge node according to both resource needs and energy consumption considerations. We proposed to set an affinity relationship between the running service and the target edge node. The affinity relationship guides the system to take a decision towards which node to execute the handoff. In the edge computing environment, we can have many types of affinity, among them: Communication Affinity (CA), Resources Affinity (RA), and Energy Affinity (EA). CA depends on communication technologies between the mobile node and the target edge node. RA is derived from the resources needed for executing the service at the edge node. EA is induced by the minimum resources needed to run the service and resource availability at the edge node. Regardless of various affinity types, our work denotes the affinity of edge services

as a key factor for the allocation of edge services at the target edge node and takes Energy Affinity as an example. Let us consider a basic scenario that comprises a large number of edge nodes available in an edge environment. The edges are distributed in several localities and are different from each other (some MEC-based other Fog-based). Each edge node has a resource capacity to run a specific service. We aim at minimizing the number of resources needed to run the service in order to save energy in an edge infrastructure. In this scenario, we describe the Energy Affinity parameter as follows. Given the resources consumption of the running service at the old edge node (in terms of CPU and RAM consumption), the optimal allocation of the service is finding by comparing with resources available at the edge nodes. Therefore, the best association is when EA is about 1.

4.3 MESH Framework Architecture

This section presents our system architecture of the proposed framework for edge-enabled service handoff. MESH uses open-source widespread technologies and is open to the community for further improvement and testing. As well, we built MESH handoff protocols on top of the ETSI MEC specifications. Figure 4.2 shows the MESH framework architecture that is organized into two layers: the edge and the device layer. MESH consists of a set of components that are deployed at the two layers and enable the handoff processes.

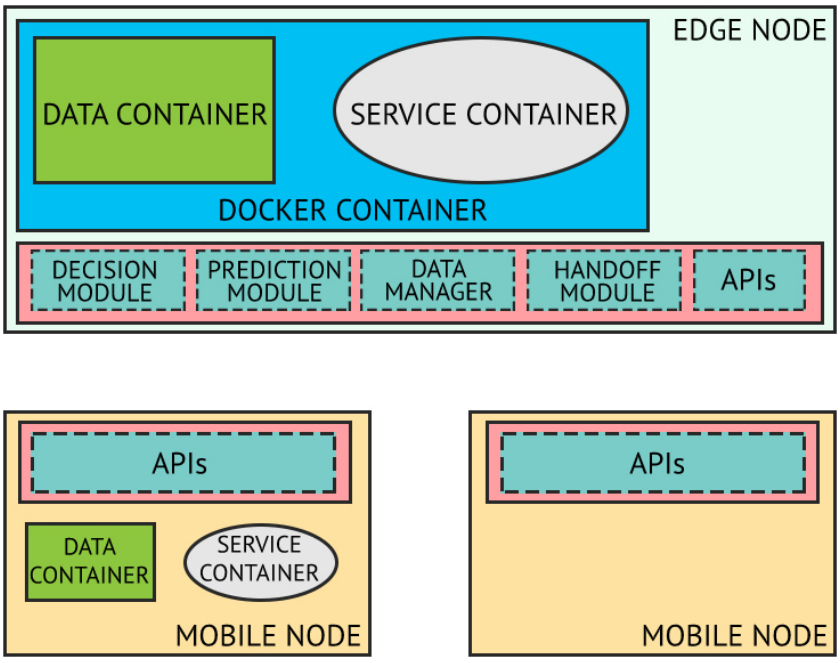


Figure 4.2 Overall logical architecture of MESH

- **APIs**. This component offers a set of common APIs that enable the interactions of our handoff protocol between all involved distributed entities at the two layers. It exposes several methods and features related to the handoff procedure, such as *handoff request*, *start*, *stop*, and so on.
- **Prediction Module**. This module is in charge of determining the appropriate time to initiate the handoff by monitoring users' locations in order to predict their movement. Several monitoring strategies have been proposed in the literature, and this work proposed a solution that enables the Prediction Module to work with any strategies. Particularly, this module collects information about users and calculates several metrics to trigger the Decision Module and to start the handoff process. We claim the importance of distinguishing two kinds of mobility prediction: a coarse-grained historical-based mobility prediction and a fine-grained RSSI-based mobility prediction. The first one is based on the history of user movements: edge nodes, possibly coordinating also with the mobile node to gather mobility traces (such as GPS positions) and the global cloud layer (to process those traces), track user movement to enable long-term predictions of user mobility habits, such as during working days, during weekend, and so forth. Moreover, when it is enabled allows the system to execute proactively long-running operations such as migration of (static) service parts towards the target edge. The second one, namely, fine-grained mobility prediction, evaluates handoff decision by using the value of edge-to-mobile RSSI by employing the monitored RSSI values obtained through heterogeneous short-range wireless technologies, such as Wi-Fi and Bluetooth. As widely recognized in the literature, this kind of prediction is expensive, typically works on shorter time intervals, but gives more accurate information about when to trigger handoff and consequently the migration of more dynamic data parts. Finally, the availability of both prediction modes enables higher flexibility for handoff management, as better explained in the next section.
- **Data Manager**. This component enables the application-aware handoff by embedding the application-specific knowledge to manage finer-grained data migration. In other words, this module observes the underlying data container by providing several data connectors and returns data to be migrated based on the migration strategy.
- **Decision Module**. This module contains a set of strategies that are used to determine data mobility. Several strategies can be used for this purpose; this dissertation proposes an approach based on data access frequencies where data accessed less have a higher

migration probability. Moreover, this module is in charge of choosing the best place (MEC node or Fog node if there are multiple edge nodes in the same region) to forward the handoff procedure by evaluating the service affinity relationship.

- **Handoff Module**. This module executes the handoff process. This component contains a set of steps that enable the interactions of the handoff protocol between all involved distributed entities via the APIs layer. This module relates to the same Handoff Module of the target edge node.

4.4 Handoff management

This section describes the proposed handoff protocols such as reactive handoff, proactive handoff, and application-aware handoff.

4.4.1 Reactive Handoff

Figure 4.3 depicts the primary steps of the baseline (reactive handoff based on Docker tools) of default Docker Containers migration. Generally, the reactive handoff procedure starts when the mobile node loses connection with the old edge node and sends a *handoff request* message to the target edge node (step 1). Upon the old edge node receives the handoff request (step 2), it starts the migration process by exporting the container to be migrated by using the Docker

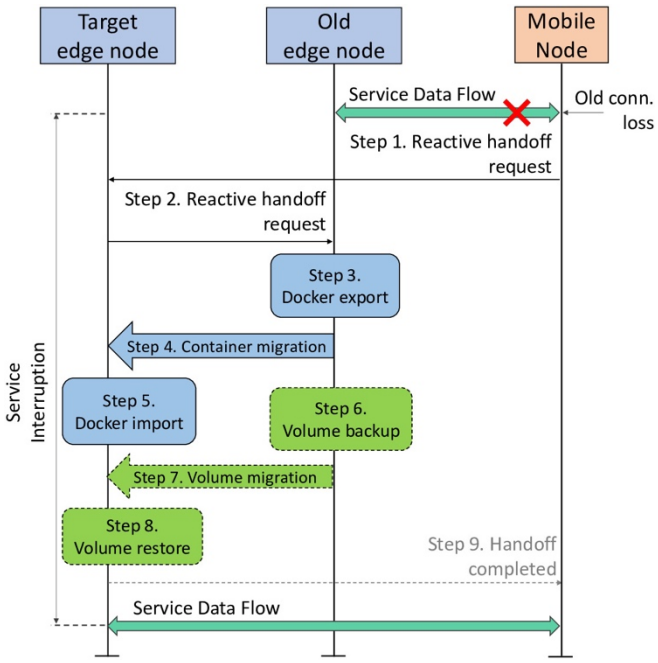


Figure 4.3 Docker basic reactive handoff

export command (step 3). Then, the old node sends the compressed container to the target edge node via network file transfer (step 4). Once the target edge node has received the compressed container it restarts it via docker import command (step 5). In parallel, the old edge node starts to prepare the backup of Data Container, if necessary and sends it to the target edge node which restores it (steps 6-7-8). Finally, the handoff procedure ends (step 9). As one can see, Figure 4.3 describes the basic Docker migration protocol (reactive handoff), which impose the service interruption from step 1 to step 9 typically suffered by monolithic services and VMs as well. Note that, to better understand our proposal, we implemented the basic Docker migration protocol also as a baseline to use for comparison in our experimental evaluation (see experimental evaluation Section 4.5). Of course, MESH would allow optimizing also this reactive handoff management, e.g., leveraging service/data software layering to avoid migrations (if needed layers are already available at the target edge) and, similarly, applying application-aware data management if possible. The next section focuses on novel protocols that implement and enhance a proactive approach in order to reduce the service interruption interval due to user handoff between different edges, including pre-loading of all selected services/data software layers at edge nodes. In this case, it is possible to distinguish different types of handoff management improvements depending on how the data state is transferred in application-agnostic and application-aware cases.

4.4.2 Proactive Handoff

The next two subsections describe the design principles behind the service/data migration protocols, detailing also the application-aware optimization. Figure 4.4 shows our optimizations of basic reactive handoff based on Docker tools. The optimizations leverage both long- and short-term predictions to enable proactive provisioning. Our handoff procedure begins when the Prediction Module predicts the migration and triggers the proactive execution of the handoff procedure (steps 1-2). Then, the protocol starts the migration of the service part (steps 3-4) and the installation of the data part (steps 5-6). In this approach, the data part is considered as a black box with no information about its inner characteristics; in the next optimizations, additional modalities of operation of the Decision Module which can operate also the application-aware handoff will be described. Therefore, steps 5-6 install only the data container while the protocol postpones the request for data backup migration until the mobile node loses connection from the old edge node, so to make sure to receive a more consistent data state, with all changes made at the old edge node (step 7). Once completed the data container backup, the old edge node starts to send the backup to the target edge node (step 8)

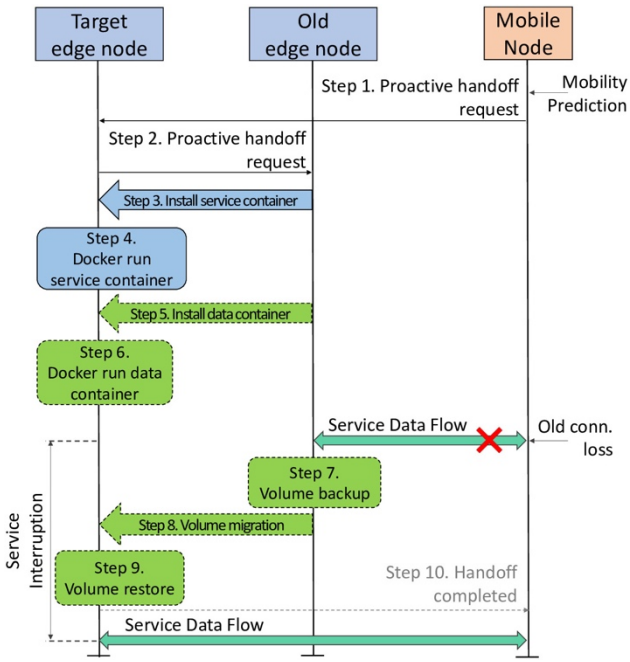


Figure 4.4 Docker proactive handoff

and then the target edge node restores the data backup with all the latest changes made at the old edge node by the user (step 9). Finally, the target edge node sends the handoff complete signal to the mobile node. As one can see from Figure 4.4, this approach of service/data migration decreases the service interruption time compared to the previous one (Docker basic reactive handoff). Let us note that this does not imply any application-specific knowledge and requirements to perform it. Furthermore, we can further reduce the service interruption time leveraging the Decision Module that contains decision mechanisms to move proper data. This is the core of our proposal and we will explain it in detail in the next subsection.

4.4.3 Application-aware Handoff

The idea behind application-aware optimizations is the exploitation of our Decision Module; thus, beyond the strategies, the Decision Module selects proper data to be moved proactively to the target edge node. In our solution, the application-aware service/data migration process is composed of multiple steps, as depicted in Figure 4.5. The user's mobility is observed by Prediction Module that can activate a trigger when the user mobile node is likely to go towards the new edge node. The Prediction Module can take advantage of both short- and long-term user's mobility prediction; for the purpose of application-aware optimizations, we need to use a long-term user's mobility prediction in order to select and move data proactively (steps 1-2). Let us note that compared to the Docker proactive application-agnostic handoff, application-aware handoff allows us to proactively move certain data to the target edge node. Thus, the old

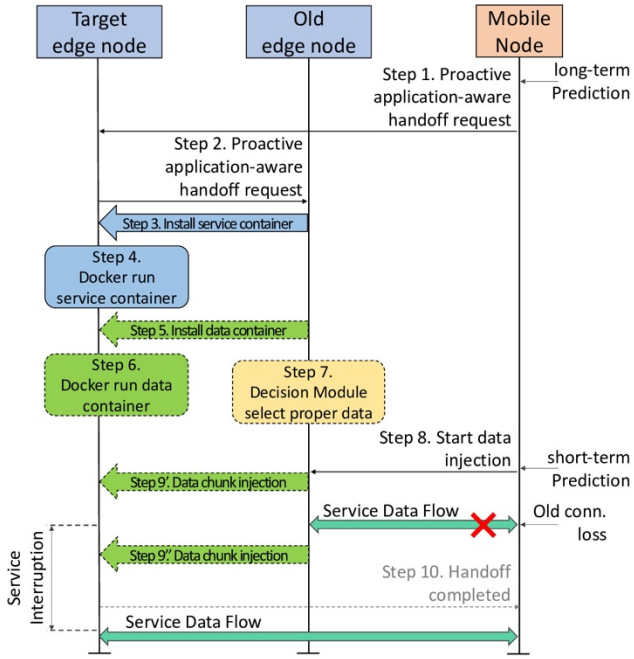


Figure 4.5 Docker proactive application-aware handoff

edge node migrates the service (steps 3-4) and the data container part (steps 5-6) to the target edge node. Then, in order to select proper data to be moved, based on the migration strategy, we need to invoke the Decision Module (step 7). In this time interval, the mobile node continues to be provisioned through the old edge node. Consequently, our procedure introduces a periodic data reconciliation phase, triggered by a short-term mobility prediction (steps 8-9'), to reduce the service interruption interval, by limiting the whole data state migration to those data chunks. This periodic data injection phase (managed by the Decision Module by using its strategies) terminates when the mobile node loses the connection with the old edge node: the protocol guarantees data consistency by sending another data chunk update from the old edge node to the target edge node, and that completes the whole handoff procedure (steps 9''-10).

4.5 Experimental Evaluation and Simulation Work

As already stated, one of the key contributions of this research is that the service/data migration solution has been implemented and completely integrated into a real MEC/Fog architecture. As a valuable side-effect, differently from seminal efforts available in the existing literature, we are able to report results obtained in our lab deployment scenario, with heterogeneous edge devices, and simulation work for additional quantitative evaluations and comparisons. To thoroughly test and evaluate the performance of MESH, this section reports three different sets of experiments, respectively, for Docker basic reactive handoff, for our application-agnostic

proactive handoff, and for our application-aware proactive handoff. The results reported in this section are average values; all presented measurements have exhibited a limited variance (under 5% for 30 runs).

4.5.1 Real in lab testbed experimental measurements

To better understand improvements in terms of system complexity and migration time, we quickly introduce our in-lab deployment scenario. The evaluation testbed consists of three Linux boxes (Ubuntu 18.04 distribution): two 3.06GHz Intel(R) CORE i5 and 8GB 1300 MHz DDR3 memory as MEC-based edge nodes, and one Raspberry Pi3 equipped with 64-bit quad-core ARM Cortex-A53 processor, 1 GB of RAM and 16 GB of storage as a Fog-enabled node. We present a case with heterogeneous edge devices where the old edge node is a micro datacenter and the target edge node is a fog node. Because this research wants to highlight one of the critical scenarios of this work. Those nodes host Docker 18.09-ce and Java 12, and all the illustrated framework components. During our experiments, we have considered the service migration performance of MESH for a specific cloud- edge-enabled application based on Docker Compose [89] defined by the Docker Compose yml file as follow:

```
version: '3'
    services:
        java:
            build: .
            ports:
            - "8080:8080"
        mongo:
            image: mongo
            volumes_from:
                - mongo: dbdata
```

This is a general schema that we used for implementing multilayering Docker-based applications. Our test service consists of a Java web application defined as the service layer and an instance of MongoDB as the data layer. The test service consists of a web-based application where users report some information, they find along the road such as obstacles, restaurants, and groceries. In particular, the Java part provides a simple human interface which users compile and insert information through a form that are stored in the MongoDB database. The MongoDB container is linked to another container (dbdata) that acts as Docker Volume [90] via the *volumes_from* Docker primitive. Let us recall that in Docker, a Volume is a

mechanism for persisting data in the local filesystem used by a Docker container. To create a Docker container for persisting data, it is possible to use the following dockercli command:

```
docker create -v /data/db --name dbdata mongo /bin/true
```

This specific characterization of our test service helps to better understand how application-aware handoff works. Indeed, the data layer (composed by a MongoDB instance) is physically separated from the rest of the service, which means that the framework optimizations can exclusively focus on this part. We chose MongoDB for its simplicity and because it provides mechanisms that allow us to implement each step of our application-aware handoff. In particular, MongoDB assembles the data in the form of collections which represent our data chunks. Finally, MongoDB provides mechanisms for sending and restoring only portions of data (i.e., data chunk) by using *mongodump* and *mongorestore* integrated tools. Unless otherwise specified, edge nodes connect each other via IEEE 802.11n connections and their maximum nominal available bandwidth is 40 Mbit/s. During the first set of experiments (Docker container migration, Figure 4.3), the persistent layer to migrate is around 300 MB until 330 MB depending on the different number of records (from 10K to 100K). Let us clarify that we considered the service already installed at the target edge node, hence we need to migrate only the data container. The handoff process starts when the mobile node loses connection with the old edge node. Hence, the total time of migration is obtained by the sum of three different steps: *export container*, *send container*, and *restore container*. The first step (step 6 Figure 4.3), *export container*, is done at the old edge node and consists in collect all container files

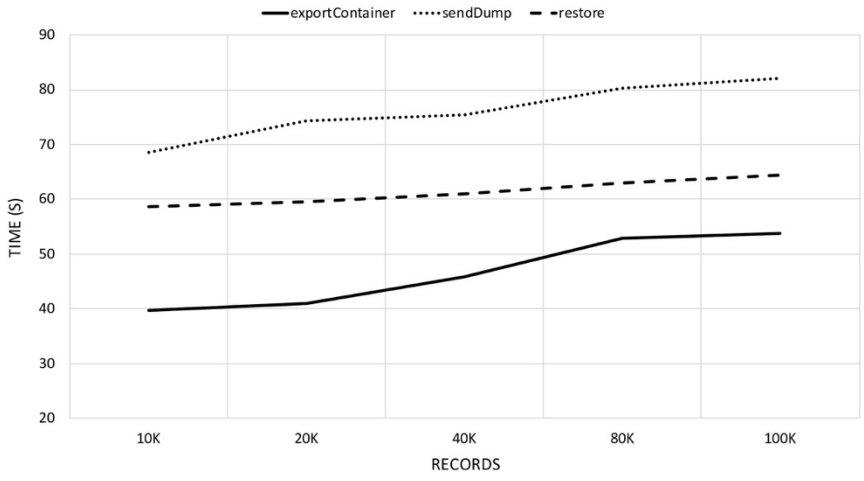


Figure 4.6 Docker basic handoff total migration time

into one tar archive file. The second step (*send container*) allows the system to transfer the tar through the network from the old edge node to the target edge node. In order to study the impact of the database size, this research has run the test several times by changing the amounts of data stored on MongoDB, from 10K to 100K records. Figure 4.6 shows the total migration time for different amounts of data by highlighting the time needed to complete each step. As depicted in that figure, the number of records affects the total migration time by a factor of around 10 s per each stage. In the worst case, the overall service interruption is around 170 s. The remaining sets of experiments are related to the more interesting proactive scenario. We have evaluated the proposed application-aware approach in terms of total migration time. In this scenario, when the Prediction Module foresees the handoff (long-term), the Decision Module at the old edge node starts to calculate data to migrated (cold data) and migrates the data toward the target edge node. Then, when the handoff happens, the system sends only the remaining data (hot data) towards the destination edge node. Finally, the destination edge node has to check the consistency of all cold data (some data may have changed value during the handoff). For this set of experiments, we set the number of cold blocks at 35% of the total blocks. Each block in the proposed implementation correspond to a MongoDB collection; in order to calculate the migration probability of each block, we used the collection stats command (*db.collection.stats()*), provided by mongo API, that returns statistics about the collection including the total number of insert and update operations. We simulated different percentages of a correct guess that means the correctness of the forecast made on the cold data. The different

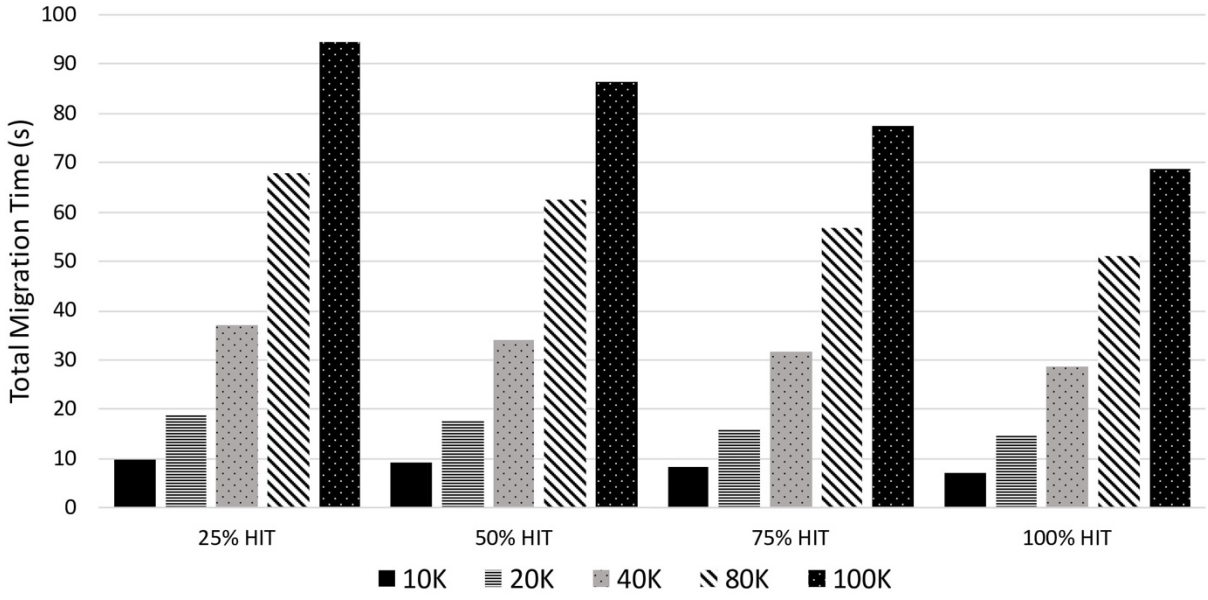


Figure 4.7 Docker application-aware handoff total migration time

simulated percentages of correct guesses are 25%, 50%, 75%, and 100%. If the forecast on the cold data is incorrect, we need to resend all cold blocks that do not match. Figure 4.7 shows the performance of the container migration for different amounts of data at different percentages of a correct guess. On the one hand, when the percentage of correct guess increases the total migration time decreases. On the other hand, when more records need to be processed the total migration time increases.

Finally, in order to understand the energy impact of MESH we have also measured CPU and RAM consumption at Raspberry PIs in the case of Docker basic handoff (which includes all long-running operations). In our in-the-field measurements, we have used the RPi-Monitor tool, i.e., a monitoring application designed to run on Raspberry PI nodes. RPi-Monitor provides an interactive Web interface to display status and graphs (for further information, see http://rpi-experiences.blogspot.fr/). The associated results are reported in Figures 4.8 and 4.9. As clearly shown in Figure 4.8, the maximum CPU consumption in the handoff process is around 200% which averages that up to 2 cores are used over the 4 total cores available. Moreover, we noted that the data startup procedure generates the maximum CPU effort. Figure 4.9, instead, shows the RAM usage during handoff, with the maximum peak during service/data migrations. In any case, these seminal results about CPU and RAM consumption demonstrate the feasibility of the MESH framework even on resource-limited platforms for the possible realization of low-cost edge nodes.

In addition, further experiments and the MESH source code are available at https://github.com/domen88/migrationModule.

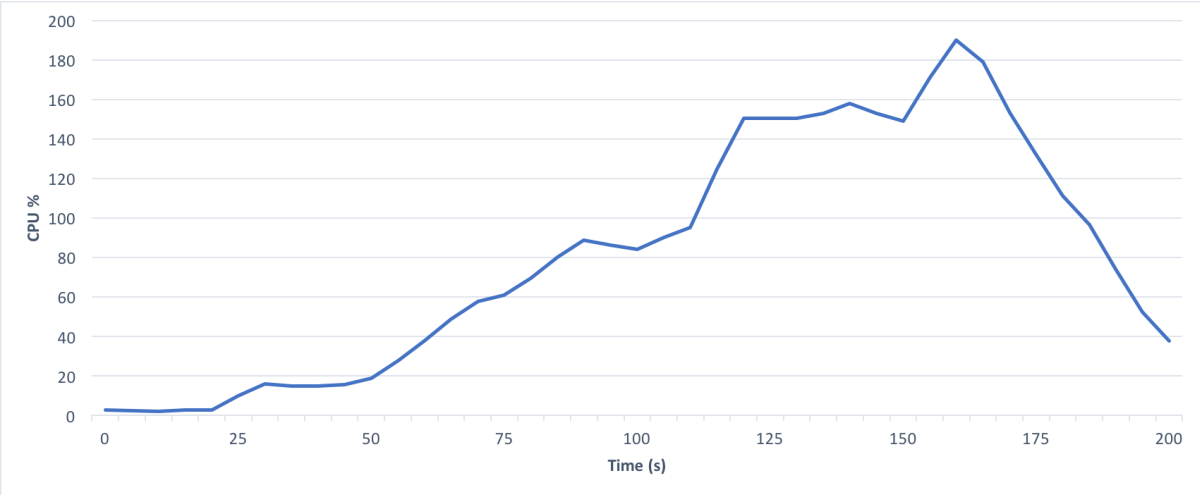


Figure 4.8 Docker basic handoff process CPU consumption at Raspberry Pi

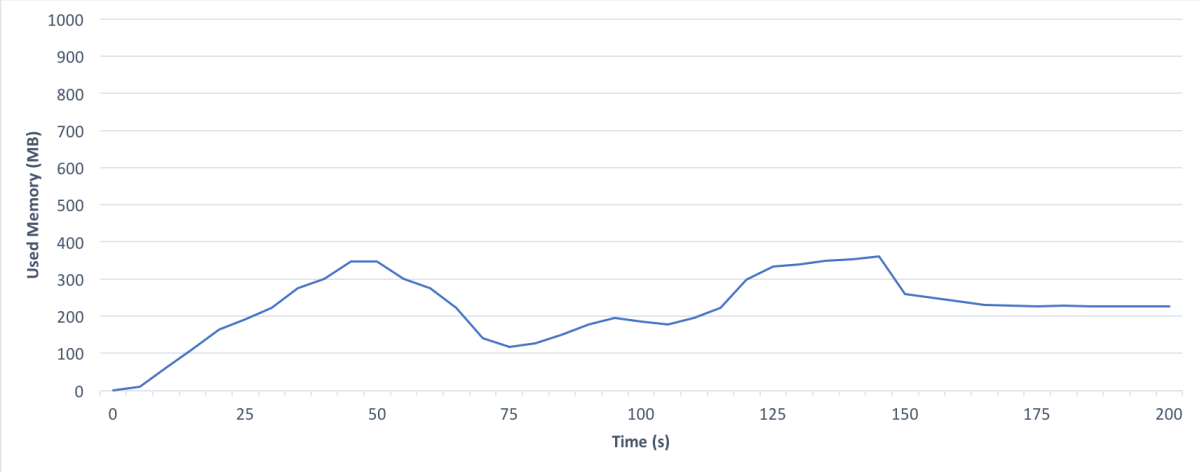


Figure 4.9 Docker basic handoff process RAM usage at Raspberry Pi

4.5.2 Simulation results about total migration time and data loss compared with data variability

For additional quantitative evaluations and comparisons, we employed CloudSim [91], an extensible and widely adopted simulation toolkit that enables the modeling and simulation of cloud computing environments. In particular, the CloudSim simulation framework supports the modeling and creation of infrastructures and application environments for distributed multiple clouds. A recent extension of CloudSim, named EdgeCloudSim [92], builds the concept of Edge Computing upon CloudSim by adding necessary functionalities in terms of computation and network capabilities. In particular, we map MESH within the simulator by creating:

- two micro datacenters, used to migrate our service to;
- one host per datacenter, with 2GB RAM and 250GB storage each;
- two VMs for each host, with 512MB RAM, 100GB storage, and 1 CPU each;
- one process per VM representing MongoDB instance.

In this simulated environment, we extensively compared our application-aware solution with two baseline approaches, such as reactive migration and proactive migration. The reactive migration adopts the approach of migrating all data at once when the handoff happens. Thus, it is characterized by high migration time (because it sends all data), and also may cause significant data loss in case of a high amount of data received during the migration process. The proactive approach, instead, moves the data in advance before the handoff happens according to the migration probability. We simulated different values of data variability and migration probability in order to show how migration time and data loss vary. Figures 4.10 and

4.11 show, respectively, the results about the total migration time and data loss in relation to the data variability for the reactive and proactive migration. The total migration time for the reactive migration always remains the same regardless of data variability value, that is so because the reactive migration ever sends all data. The same does not apply to data loss. If we have a high value of data variability, a long interruption of the service (caused by the migration), may generate a high value of data loss, because a mobile device can still use the service at the old edge node during the handoff. The situation is different for proactive migration. Let us note that the results reported in Figure 4.11 have been obtained by simulating a migration of data container with size 200MB, and migration probability at 0.7. Then, the figure shows how the total migration time depends on the choice of the migration probability. Therefore, if the system has more than 1kB/s of data variability rate, the choice to have 0.7 as

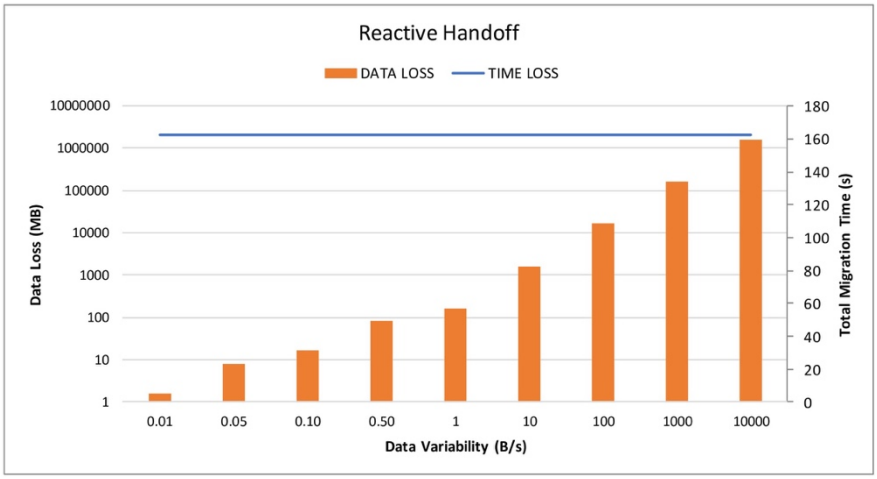


Figure 4.10 Total migration time for reactive handoff

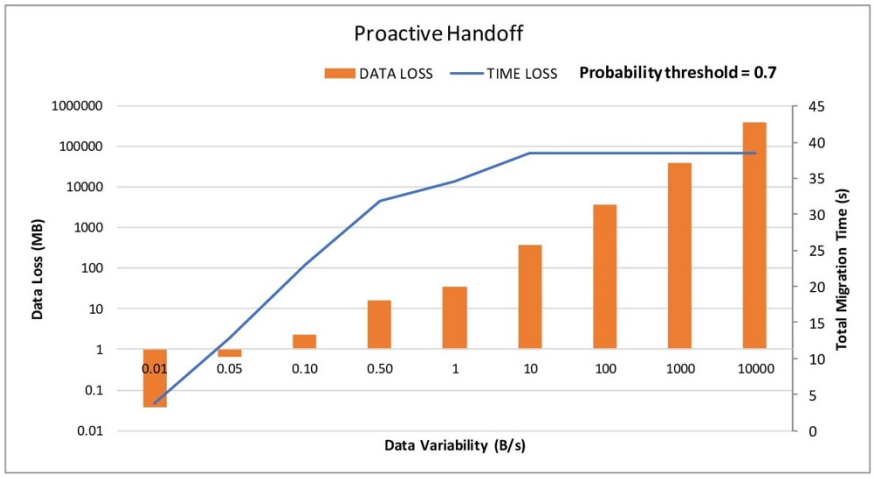


Figure 4.11 Total migration time for proactive handoff

migration probability does not lead to any benefits. In other words, the results in Figures 4.10 and 4.11 highlight the relevance of being able to dynamically adapt the migration behavior to expected data variability, as in MESH where we use the migration probability as a threshold to decide whether or not to move data. Finally, Figure 4.12 reports about how we have modeled the migration probability in our simulations, by showing how the total migration time changes in relation to the migration probability and the data variability rate. Indeed, the figure represents a general model to choose the more suitable migration probability value in relation to how quickly data records change. With the simulation results, we want to give a baseline guide to choose the best value of migration probability related to the data variability if available.

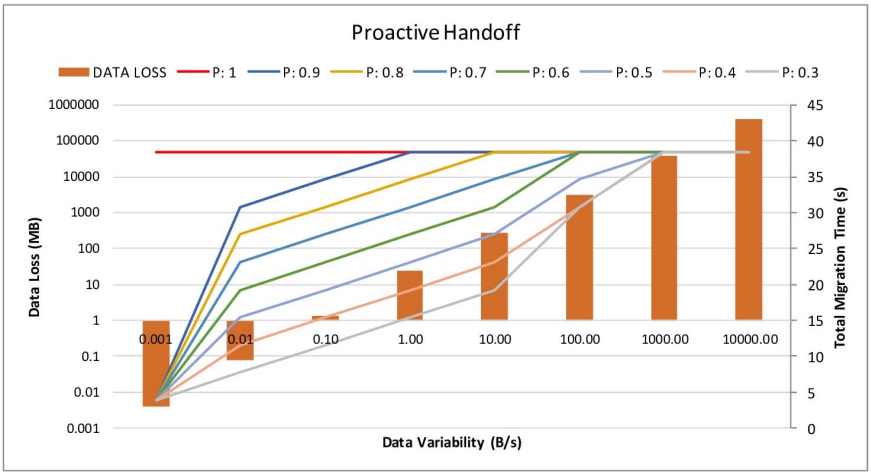


Figure 4.12 Total migration time relates to migration probability

4.6 Lessons learnt and Ongoing work

MESH supports the mobility of edge-enabled services in a three-layer edge computing environment. In particular, MESH works either in application-agnostic mode and application-aware mode (if possible), and it manages the heterogeneity of the edge environment. We have already validated our approach both via real experiments and using simulations with synthetic values of data variability. The reported results confirm that proactive migration adopted can significantly minimize the service downtime in the case of layered services (total migration time reductions of 30% ~ 50%), by imposing a very limited overhead on the overall support infrastructure. Moreover, in the experiments, Docker has demonstrated to be a properly mature solution for running different applications and managing them in a relatively easy way, without specialist knowledge. Future research work should investigate whether other container

technologies offer a better overall solution, in particular in terms of additional management flexibility. To the proper selection of the most suitable container technology, we recommend carefully considering also which technology is likely to receive the most widespread adoption from industry: in this perspective, OCI (https://www.opencontainers.org/), which aims to create an open industry standard for container technologies, can become a relevant reference point. Furthermore, MESH may involve different software components, not only related to container technologies, such as filesystems. In this case, MESH should be able to realize which portion of data can be moved proactively towards the target edge node. Another important and open aspect to pave the way to the adoption of MESH is to broadly study service migration performance under real and large-scale deployment environments. In fact, while the performance of migration mechanisms for wide-scale scenarios has mainly been studied via simulation, there are still not available real application migration experiments over real testbed environments.

Finally, fueled by these significant results, we are working on two main ongoing research directions. On the one hand, we are deploying the realized solution, already widely tested in the geographically distributed Edge Computing testbed, in a federated cloud environment with heterogeneous devices. On the other hand, we are integrating our handoff solution in a wider supportable to leverage also human sociality and mobility effects to broaden the MEC coverage through the impromptu formation of groups of peer devices acting as logical edge nodes over a localized area [93].

5 MOBILE EDGE FILE SYSTEM (MEFS)

As stated in Chapter 2, computation offloading is employed by mobile apps running over resource-constrained devices to leverage the cloud in overcoming their resource limits. The advent of the Edge Computing paradigm further extends the potential opportunities of mobile-cloud offloading, allowing new service provisioning scenarios, such as mobile gaming and multimedia, where responsiveness of mobile devices at the network edge significantly benefits from low latency interactions. However, state-of-the-art offloading platforms for EC architecture have not addressed the technical challenge of supporting file systems, due to user's mobility, and system resilience. This chapter will present the MEFS an application-level distributed file system tailored for 5GEE architecture and designed to be highly resilient and able to efficiently maintain consistency among the mobile, edge, and cloud entities. Also, this chapter will address the problem of application migration by proposing an efficient live migration algorithm and a mechanism to prevent faults. Finally, it will present a prototype of the system built on the Android platform and a set of experimental results.

5.1 Motivation

During the past decade, the users' requirements on data rates and Quality of Service (QoS) have increased substantially. Furthermore, the technological evolution of smart phones has led to mobile apps requiring huge processing power, while the battery life and power consumption still pose significant technical challenges toward achieving optimal users' Quality of Experience (QoE). As already stated, this motivates the idea and development of MCC platforms, which allow mobile users to seamlessly leverage powerful resources available in the cloud. MCC has already demonstrated several advantages in terms of QoS, QoE, and energy consumption; for instance, it enables computation offloading from mobile users to the cloud [59, 60, 62, 94]. However, MCC solutions often exhibit the drawback of increased latency due to mobile-to-cloud communication. But luckily Edge Computing paradigm can overcome this by reducing the communication latency and the overall app response time. Compared to MCC, EC can offer significantly lower latency and jitter; moreover, since EC can be deployed in a fully distributed manner, it can improve the overall scalability for mobile apps. In the EC paradigm, an edge-enabled application can have components running at three hosting environments, i.e., the mobile device, the cloud, and the edge servers that are selected based on the current location of the mobile device and may change when the mobile device moves.

Despite the potential advantages of the EC paradigm, proposed EC platforms have not addressed yet the technical challenge of supporting specific file systems for the envisioned edge-enabled class of applications. Generally, mobile-edge-cloud applications execute tasks at mobile side and edge/cloud side. Therefore, computation offloading cannot be employed for most mobile-edge-cloud apps because tasks need to read and write files concurrently on both mobile and edge/cloud side. At the current stage of specification, both MEC and Fog paradigms do not provide any default mechanism to support concurrent file access and strong consistency. Let us specify an example of edge-assisted application that we have recognize as a significant case: augmented reality and real-time video analytics (shown in Figure 5.1).

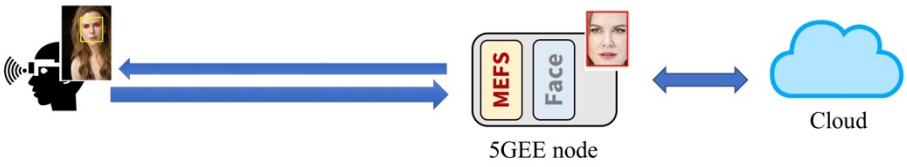


Figure 5.1 Example of edge-assisted application

The idea of this application is that a mobile user can start recording a video (i.e., from a smartphone, or from a smart goggles) and sharing it with the closest edge node that have installed on it our proposed MEFS and a face recognition application. Once the user detects a specific frame, the edge can return the augmented information about that frame including information of face recognition. Others class of edge-assisted applications that we have taken in consideration are real-time mobile gaming, large-scale video analytics, and augmented reality. All of these share the characteristics of file I/O operations for edge-assisted applications such as read and write files on both mobile and edge/cloud, require strong consistency, and long I/O latency to transfer the file to the cloud, but low latency to transfer it to the edge. Unfortunately, existing offloading systems cannot handle file I/O operations efficiently. This because, some of them do not support offloading tasks that perform file I/O operations including a recent proposal named COMET [62]. Some other systems lack the support for consistent remote file access such as CloneCloud [59], MAUI [60], ThinkAir [61], and Sapphire [95]. On the contrary, we also may have problems when we use network and distributed file systems including Dropbox [96] and NFS [97]. Generally, they are not designed to achieve strong consistency with low latency and low network overhead. Furthermore, do not guarantee concurrently file access; in the most time we need, for instance, to close the file first in the mobile side and reopen the file in the edge side and sometimes we need root privilege to

access the file in write or execute mode. Moreover, all mentioned existing solutions do not fit the EC scenarios primarily for two reasons: i) they cannot handle the switch of edge nodes when mobile users move; and ii) they are not resilient to the faults in edge nodes. By carefully considering these relevant gaps, we propose Mobile Edge File System (MEFS), an application-level file system that runs on mobile devices, edge nodes, and the cloud to efficiently and seamlessly handle file accesses for edge-assisted mobile apps. MEFS supports file accesses with low-latency for the components of a mobile app that possibly offloads some tasks to either the edge or the cloud (mobile-to-edge or mobile-to-cloud offloading) and guarantees strong data consistency between these components. It is fully compatible with the MEC standard and our 5GEE specifications. With MEFS, an entirely new class of mobile apps (i.e., apps that need access to files) can take advantage of the EC infrastructure for faster response time and lower energy consumption on mobiles.

To conclude, MEFS contributes to the literature in the field in multiple ways. First, we propose and design MEFS to make it possible that the components in edge-assigned apps can access files concurrently and consistently from cloud, edge, and mobile nodes. Second, we have implemented a MEFS prototype based on Android. Third, in order to test the MEFS design, we have implemented edge-assisted mobile apps, one of which is a real-time video analytics mobile app. Each app can utilize the assistance from either the edge or the cloud, such that we can use the app to compare the performance of MEFS with that of a mobile-cloud file system and demonstrate the benefits of MEFS. Finally, we have conducted extensive experiments with a test app and real mobile user traces, to validate the functionality and performance of MEFS.

5.2 MEFS: Requirements, Background and Architecture

MEFS aims at supporting edge-assisted mobile apps for EC networks. MEFS provides support for mobility management and fault-tolerance, while offering strong consistency and low latency for concurrent file accesses. This section describes the main requirements for MEFS and presents its architectural model.

5.2.1 MEFS Requirements

From the user perspective, a critical use case regarding EC is computation offloading, as this can save energy and/or speed up the computation. One recognized concern of computation offloading is the proper management of the associated latency: for applications with stringent response time constraints (e.g., gaming, multimedia, augmented reality), the high latency

between mobile devices and the cloud is not tolerable and could be a significant obstacle to the users' QoE. To better emphasize this concept, let us consider a typical real-time video analytics scenario: law-enforcement agencies may need to perform face recognition in real-time across large areas to identify potentially dangerous people. This scenario requires very low latency because the output of the analytics is used to interact with users (i.e., law- enforcement officers), requires high bandwidth for high- definition video streaming, and requires computation at the edge to enable low usage of the cloud. MEFS provides full infrastructure support for MEC environments and addresses three main technical requirements:

- **Strong consistency**. Platforms that offload resource-demanding tasks of mobile apps to the cloud or the edge [59, 60, 62, 98] lead to a scenario were computation tasks run concurrently on both the mobile and the cloud/edge. These tasks may need to access files on both these entities. However, the offloading platforms either do not support offloading of tasks with file I/O [62, 98] or allow access only to the files that are available locally [62]. MEFS leverages OFS system [99], which is an application level file system that sits between mobile-cloud apps and the offloading middleware. OFS allows mobile-cloud apps to access files from both mobile and cloud concurrently, while providing strong consistency and low latency.
- **Application portability**. MEFS can portably transfer apps between edge nodes. It overcomes the application portability challenge by creating a set of APIs useful for developers to manage user mobility. Once the handoff is started, MEFS automatically communicates with its MEFS module at the new edge node and transparently moves the app. Handoff is the process of switching one connection endpoint from one edge node to another in the midst of communication. More specifically, MEFS transfers the file system state and associated metadata, while the offloading middleware transfers the app state (i.e., app variables).
- **Resilience**. To protect against node or communication failures, MEFS leverages the cloud, as a controller entity, to provide fault-tolerance in two cases. First, if a edge node fails, the cloud is in charge of restoring the affected app either in the cloud or at a new edge node. Second, if the user moves away from the current edge node and there is no other edge node available in her proximity (single- hop coverage range), the app is again restored in the cloud. To this end, MEFS provides a transparent mechanism that synchronizes the file system state and associated metadata between edge nodes and the cloud.

5.2.2 MEFS Background

To better understand MEFS, we first need to introduce some background concepts related to Overlay File System (OFS) [99]. OFS is an application layer file system built for cloud-assisted mobile applications. It has these primary features:

- does not need system-wide management;
- can work with any native file system;
- does not incur costly context switches.

Moreover, it guarantees great advantages in terms of consistency, transparency, the overhead introduced, and deployment.

As illustrated in Figure 5.2, OFS is a component of the system that offloads and manages computation tasks. As OFS works at the application level, it can run in user mode and its data structures (e.g., user's state, and data buffer) are maintained in user space. The objective of OFS is to provide efficient, transparent, and consistent file access and file sharing for tasks in a cloud-assisted mobile app. For this purpose, OFS intercepts and monitors file access requests from tasks in the application. For the requests accessing remote files, OFS maintains a buffer named *Block buffer* to cache the blocks read from remote files through the network. To fulfill the requests, OFS looks up the block buffer and serves the re-quests if the desired file blocks are cached there. Otherwise, it redirects the unsatisfied requests to the platform storing the files. OFS maintains consistency between the blocks in the block buffer and their counterparts saved

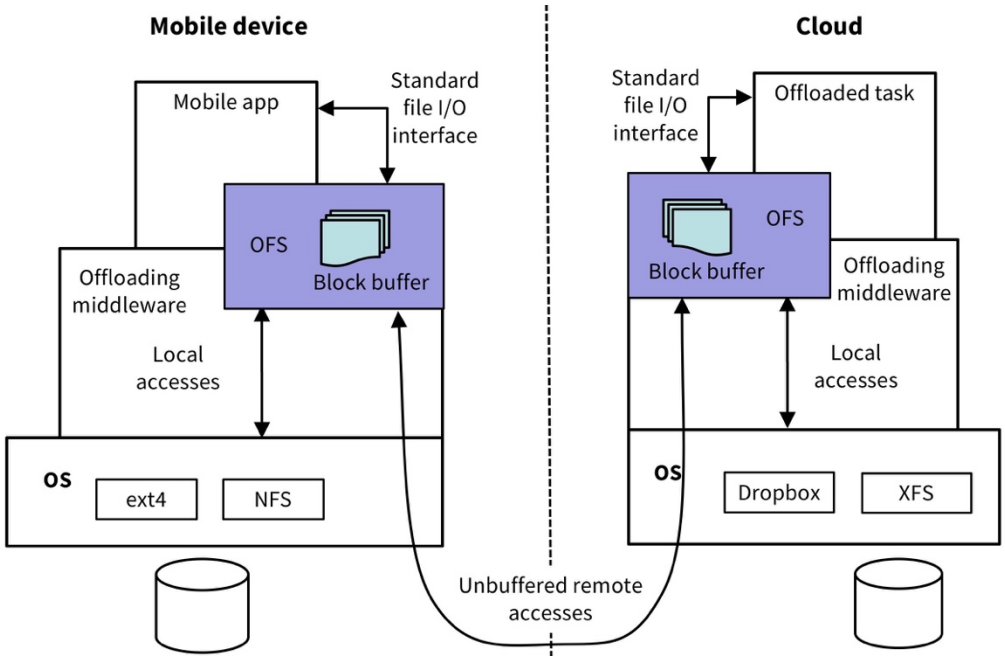


Figure 5.2 Overall architecture of OFS

in remote files. To ensure this, OFS proposes a novel algorithm named *Delayed-update* that is the combination of two common algorithms such as write-invalidate and write-update. As one will see in the next sub-section, MEFS is built on top of OFS, thus ensuring strong consistency and adding capabilities to work in the EC environment. Further elements within OFS will be described in the next sub-section.

5.2.3 MEFS Architecture

As already stated, MEFS leverages OFS to manage remote file access and file sharing among the distributed components of edge-assisted mobile apps. Furthermore, it provides support for application portability and resilience. Figure 5.3 depicts the general architecture of our solution, with MEFS and the offloading middleware deployed on mobiles, edges nodes, and the cloud; in this scenario, a mobile app can offload its computation to a nearby edge node. The cloud is used as a controller that helps with fault-tolerance but is not generally involved in app computation. When the user moves from one edge node to another (e.g., from Edge1 to Edge2 in the figure), MEFS is able to seamlessly perform handoff in order to maintain communication locality and low latency. The figure also shows the interaction between MEFS and the employed offloading middleware, which is kept independent of the edge-enabled file system design and implementation. In our prototype, we assumed the Avatar offloading middleware [94] and we relied on the EC architecture to not only host our framework at the edge but also to manage the mobility management and the fault-tolerance. Figure 5.4 details the MEFS architectural components. To provide *strong consistency*, MEFS uses OFS, which is designed

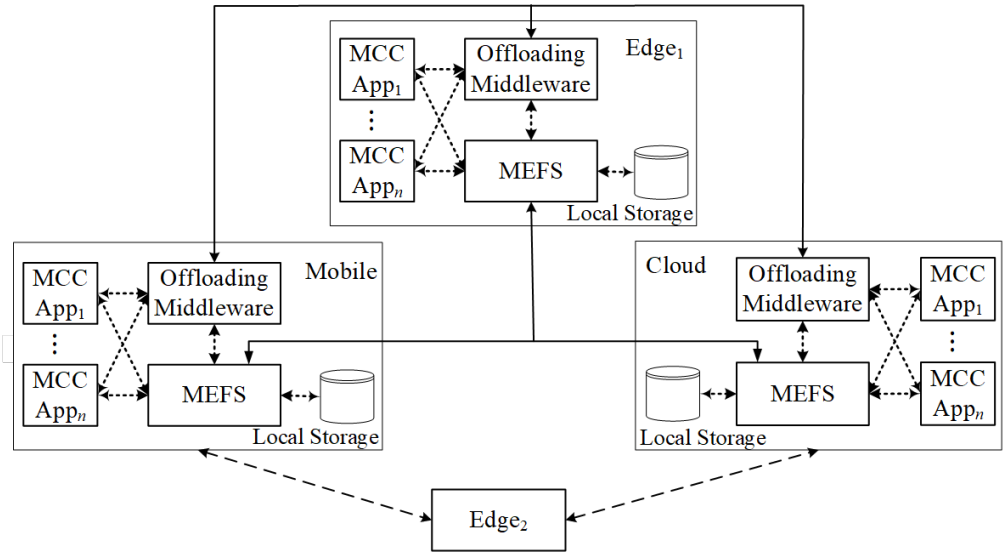


Figure 5.3 Overall architecture of MEFS for EC environment

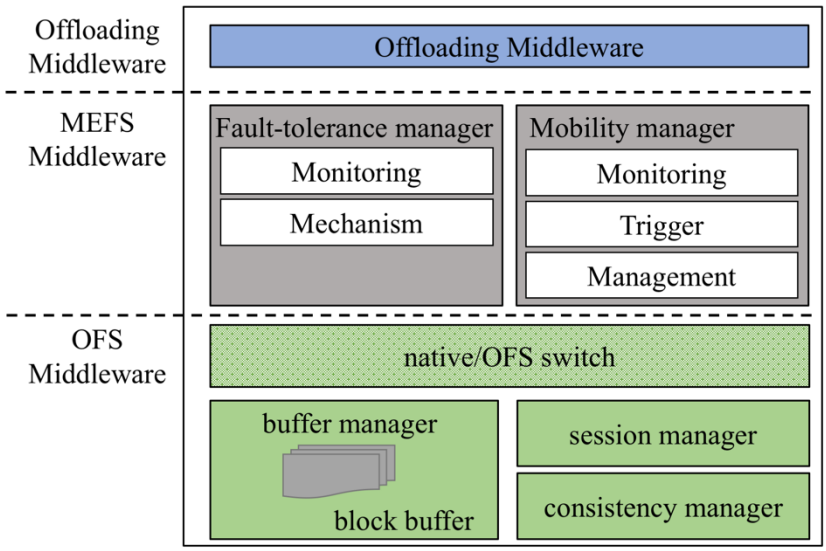


Figure 5.4 MEFS Architectural components

and implemented as an event-driven middleware. The components of OFS are shown at the bottom of Figure 5.4. The events handled by OFS can be divided into two categories: Control events generated by OFS and the offloading middleware, and messages that represent file I/O operations generated by the apps. OFS has four major components: native/OFS switch, session management, buffer management, and consistency management. The *native/OFS switch* is included in the mobile-cloud app as a support library. It decides whether the file can be accessed locally or from OFS. The *session manager* manages file states by maintaining file sessions. The *buffer manager* oversees the block buffer that contains file blocks that are currently being accessed through OFS. It also maintains metadata for each file block. Finally, the *consistency manager* maintains consistency between file I/O operations from tasks running on both mobile and cloud. It implements a delayed-update consistency algorithm. Focusing on MEFS middleware, it includes two other modules on top of the OFS such as the mobility manager module and the fault-tolerance module. The *mobility manager* module guarantees the application portability required by the MEC standard. Specifically, in an EC environment, mobile users may change their location, which makes the system location dependent. That is why this module has to manage the handoff process. The module is composed of three components: monitoring, trigger, and management. The monitoring component monitors users' locations in order to predict their movement. Several monitoring strategies have been proposed in the literature, and we designed the mobility manager module to work with any strategies. The trigger component is in charge of determining the appropriate time to initiate the handoff. Particularly, this component collects information about users and calculates several metrics to

start the handoff process. Management is the component that executes the handoff process between edge nodes. Moreover, this component defines the information flows between the system entities, which guarantee the correct and efficient execution of handoff among them. The *fault-tolerance manager* module is in charge of managing the recovery from faults that may occur at the edge nodes. According to the MEC standard, system resilience is a mandatory requirement. For this purpose, we have implemented this module with two main components: monitoring and mechanism. The monitoring component needs to figure out when faults happened. In the literature, several strategies have been proposed in order to overcome system failures. MEFS is designed to be agnostic to these strategies. The mechanism component defines the algorithm used for maintaining system consistency and for restoring the session during the recovery process.

5.3 MEFS: Implementation Highlights

We have implemented a MEFS prototype in Java on Android. However, it can be adapted to other mobile OSs. We integrated the MEFS stub in the existing native/OFS module switch using AspectJ [100]. MEFS uses an IPC service to communicate with other apps, a network service to communicate with the edge and the cloud and runs the mobility manager module and fault-tolerance manager module as Android application services, which run perform long-running operations in the background. Lastly, we used Android's Binder IPC mechanism for IPC and an NIO-based TCP library named Kryonet [101], which provides high network throughput and low latency, for the network service.

5.3.1 Mobility Management

The MEFS mobility management is targeted to the future 5G networks and is fully compliant with the ETSI MEC technical requirements for managing end-to-end mobility aspects between edge nodes [102]. ETSI MEC has defined the requirements for mobility such as continuity of service, mobility of application, and mobility of application-specific user-related information. Moreover, it specifies the standard end-to-end information flows between edge nodes that systems have to manage. The flow is composed of five macro functionalities: (1) user bearer change detection, (2) service relocation management, (3) application instance relocation, (4) updating traffic rules, (5) terminating the source service. The MEFS handoff protocol was built on these specifications. For stateful apps, such as the apps that use files, it is beneficial migrating user's data and state from edge_node1 to edge_node2 as a consequence of the

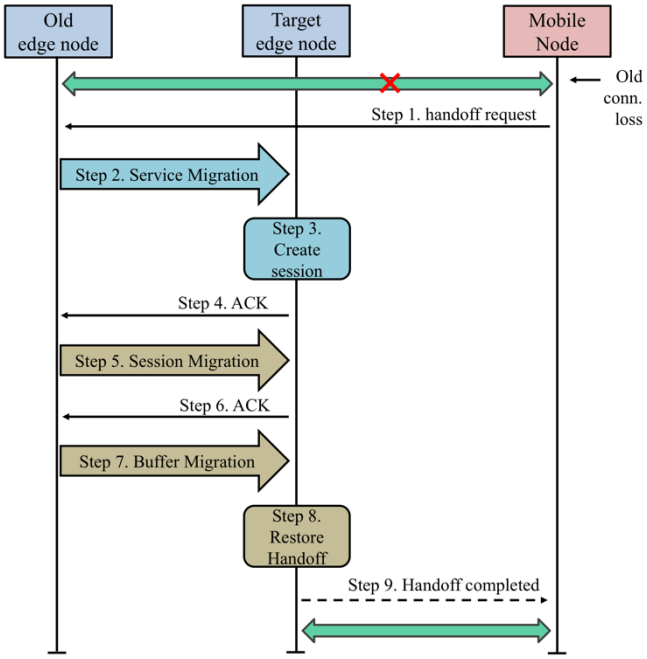


Figure 5.5 MEFS basic handoff protocol

associated handoff, in order to support the efficient continuation of the offloaded computation with reduced latency. For this purpose, MEFS defines and implements a reactive handoff protocol, as depicted in Figure 5.5. In general, the handoff can be triggered in two ways: one is infrastructural-dependent, where the system starts the handoff; the other is triggered by the mobile node. In the previous chapter, we explained in detail issues related to the handoff trigger. In MEFS, we present a solution for triggering the handoff from the mobile node based on the user's mobility path (explained in the following). Thus, the mobile node starts the handoff process by sending a specific handoff message to the old edge node (Figure 5.5 - step 1). After that, the old edge node sends the service to the target edge node (steps 2, 3). Let us note that we use the MEC term service to denote the offloaded app tasks that run at the MEC nodes. As one can see, step 2 appears to be the bottleneck of the handoff protocol by introducing high overhead to send all contents about the service. For this reason, we also implemented a proactive migration strategy, which involves the cloud that proactively installs services on the target edge node. Figure 5.6. explains how our user mobility path strategy works. When possible, mobile nodes provide their expected route to the MEFS infrastructure at the starting of a new session. For example, if the mobile node has to go from A to B, the cloud knows that in the path there are edge_node2 and edge_node3 and it can proactively install the needed app components there. In principle, the target edge nodes can be reasonably well predicted based on the Received Signal Strength Indication (RSSI) or TCP throughput on the expected user's path. In the current implementation of MEFS, the prediction is based on the

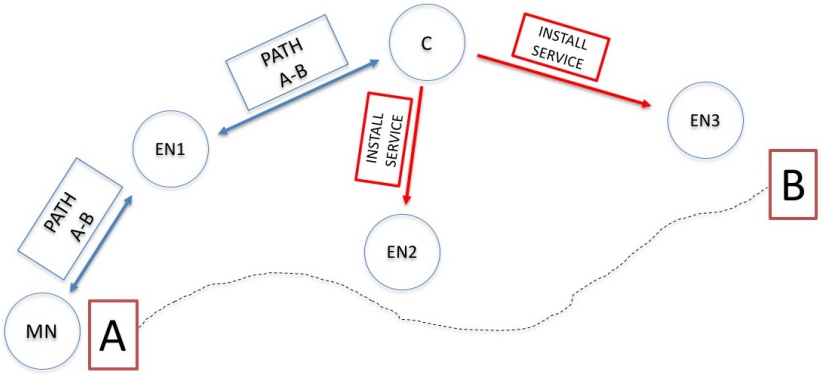


Figure 5.6 User mobility path strategy

user history of previously explored paths and on application-specific path constraints that may be defined at configuration time. Furthermore, the prediction logics can be easily improved and extended in the future without changing the remainder of MEFS and its APIs, which are prediction logic independent. An incorrect prediction can result in extra-overhead: either the state is transferred to an edge node unnecessarily or the edge node does not benefit from proactive migration and needs to retrieve the state from the cloud on-demand, as described in the next sub-section. Back to our handoff protocol, steps 5-7 identify the core of the migration phase. In MEFS, this phase involves both the user's data and state. In OFS, data about computation are contained in the **BufferManager** Java class, while the state is stored in the **SessionManager** Java class. To ensure that the target edge can restore the computation offloading process after the handoff, MEFS moves the Buffer and Session objects from the old edge to the target edge. This is implemented via an abstraction, called Handoff, which contains the Buffer, the Session, and an associated manager class. The primary APIs of the manager class are:

Table 5.1 List of methods for supporting user mobility

Method	**Description**
prepareHandoff	This method is in charge of converting the handoff object to a Parcelable Android object. Parcelable is a class used in Android for more efficient object serialization.
sendHandoff	This method allows to send the handoff object from the old edge node to the target edge node. The object is sent via the OFS event support. In particular, we have defined a new event, called HANDOFF, that is useful for managing the entire handoff process.

restoreHandoff	This method runs at the target edge node and is in charge of receiving the handoff object and restoring the session and buffer. After that, the method sends a HELLO message to the mobile node in order to establish a new connection and sends the HANDOFF_FINISHED message to the old edge to inform it that the handoff is completed.

Thus, when a handoff request occurs, the old edge node invokes the *prepareHandoff* method to create the proper handoff object. To send it, the edge uses the *sendHandoff* method. Lastly, when the target edge receives the handoff object, it can invoke *restoreHandoff* to restore the session and to start a new connection with the mobile node. Another important aspect of any edge-related handoff process that has widely investigated in the previous chapter is which type of migration is performed. As already stated, from the recent literature we can distinguish between service migration and live migration. MEFS implements its specific mechanism for live migration in the case of task offloading. Figure 5.7 shows an overview of the phases of our live migration implementation. When the handoff process is started, we put the MEFS middleware into a special state ("*handoff_mode*"); in this mode, the edge node stores each file access request sent by the mobile user in a specific queue called **DirtyQueue**, without performing the associated request. When the mobile node performs the connection handoff from the old edge to the target edge, the DirtyQueue is sent to the target edge. After that, the target edge can restore all the requests contained in DirtyQueue. In this way, we guarantee a

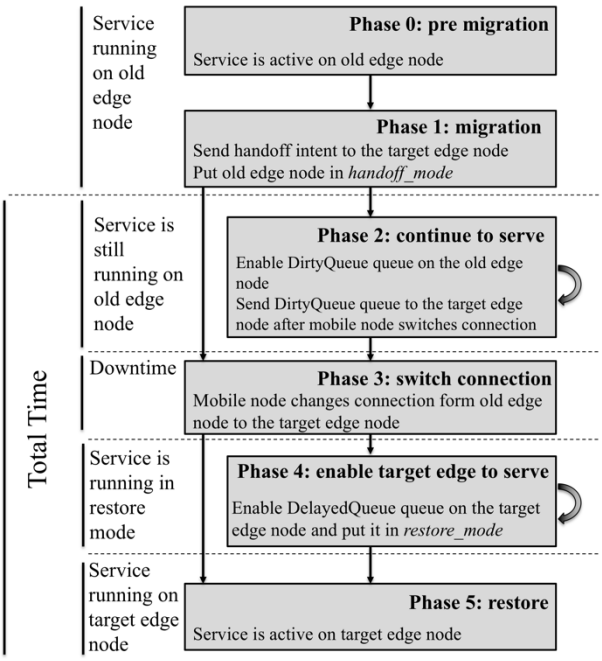


Figure 5.7 MEFS live migration overview

short service downtime, practically almost the same with the short time interval when the mobile node is temporarily with no connection. This mechanism works coupled with a symmetric one running at the target edge. After the mobile node has completed its handoff from the old edge node to the target edge node, it cannot perform any request at the target edge before the DirtyQueue is restored. MEFS has two types of requests: READ and WRITE. It is easy to understand that one cannot perform a consistent READ operation if there are WRITE operations in the DirtyQueue. To overcome this problem, we create a new special state for our offloading middleware called "*restore_mode*": when the old edge notifies the target edge of the intent to perform the handoff, the target edge enables the "*restore_mode*". Once the mobile node connects to the target edge and the "*restore_mode*" is active, all requests performed by the mobile node are stored to a specific queue called **DelayedQueue**. The requests are still stored into the DelayedQueue until the DirtyQueue is restored. Finally, the target edge node can restore the DelayedQueue and can deactivate the "*restore_mode*". In conclusion, migration can also be done by saving the latest data and state from one edge node into the cloud and restoring them onto the other edge node. Such a saving and restoring mechanism has already been designed and implemented in MEFS for fault-tolerance (next sub-section). We chose to not involve the cloud in migration management for three reasons: 1) the network latency is usually lower between edge nodes than that between an edge node and the cloud; 2) a decentralized design has better scalability; 3) the way that edge nodes work autonomously and separately from the cloud provides additional reliability.

5.3.2 Fault-tolerance

Failures may happen in an EC environment due to various reasons. One of the common reasons is network coverage. For example, a mobile node is connected with an edge node in a certain location and then is moving to a location where there are no new edge nodes. Another common reason for failures is a crash of an edge node. In the case of a failure, MEFS works to prevent the latest states of the files at the edge nodes from being lost or becoming inaccessible, such that edge-assisted mobile apps can continue to run correctly on the smartphone alone or via offloading to another edge. As shown in Figure 5.3, MEFS exploits the cloud for this solution. At the beginning of the session, the mobile node connects both with the cloud and the edge and starts the session normally with the edge. The basic idea is to maintain data/state consistency between the edge and the cloud; this can be modeled as a traditional problem of coherency between storage at different network layers. Given that it is recognized that there is no best solution for every deployment environment and application domain to detect which data to

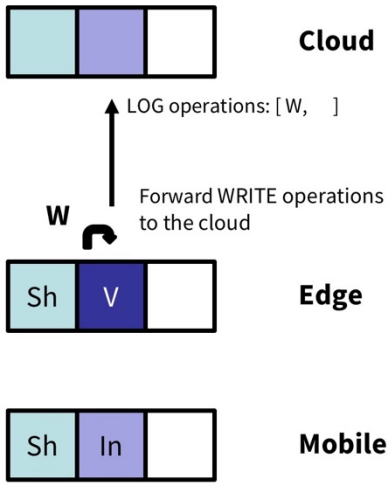


Figure 5.8 MEFS log-based approach

move from the edge to the cloud and how frequently to do it, we have decided to implement a solution based on the Log-structured file system [103]. In particular, our MEFS support for fault tolerance sends to the cloud each WRITE operation performed by the edge; note that WRITE operations are smaller and faster than a backup of the whole data. In the case of a failure, the cloud will perform all the received WRITE operations in order to restore the same data/state conditions at the edge. Figure 5.8. shows how the log-based approach works. By delving into finer implementation details, we have implemented a **FaultToleranceManager** that offers several methods for handling failures, which may be invoked during the four phases of our fault tolerance protocol:

1. *Send WRITE operation*. This is a functionality in the FaultToleranceManager module that sends each WRITE operation from the edge to the cloud. We have also created a new event named FAULT_TOLERANT that contains the WRITE operation. Each WRITE operation sent to the cloud is stored in a queue named *operationQueue*.
2. *New Block*. When the edge node creates a new file block (this may happen when the edge node tries to write more than 8KB, which is the standard block size) there is a functionality that sends that block to the cloud. A new event is created to support this: FAULT_TOLERANT_BLOCK.
3. *PUSH message*. Each time the edge node sends a PULL request (i.e., the edge node retrieves the latest blocks from the mobile node) to the mobile node, or the mobile node sends a PUSH operation (i.e., the mobile node sends the latest blocks to the edge node) to the edge node, we have to propagate these operations via an associated PUSH operation that sends all interested blocks to the cloud. Hence, the cloud must clear the

operationQueue (associated with the PUSH_CLEAR event) because with this event it already has the latest version of the blocks.

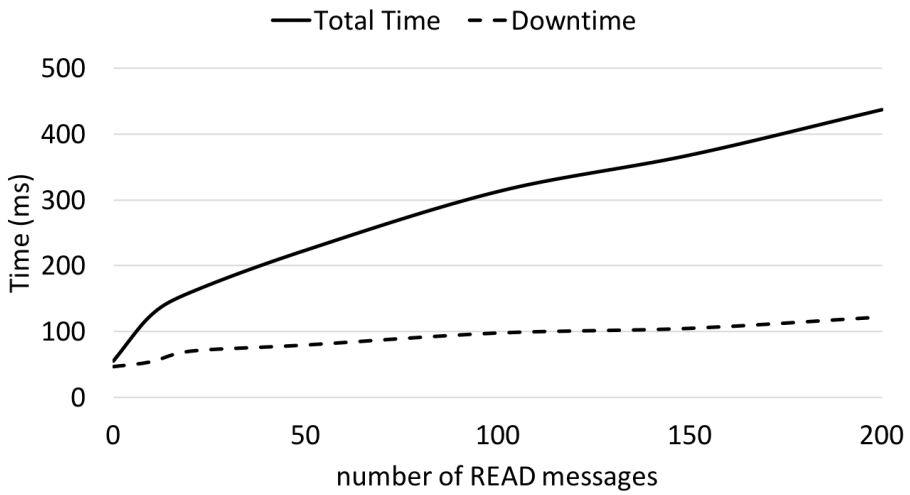
4. *Restore*. To detect failures at edge nodes, we have implemented a simple mechanism based on ACK message exchange between the edge node and the cloud. If the cloud does not receive ACK messages from the edge after a configurable time threshold, the cloud is triggered to restore the session by performing all the operations in the *operationQueue*; after that, the cloud starts a new connection with the targeted mobile node.

5.4 MEFS Performance Evaluation

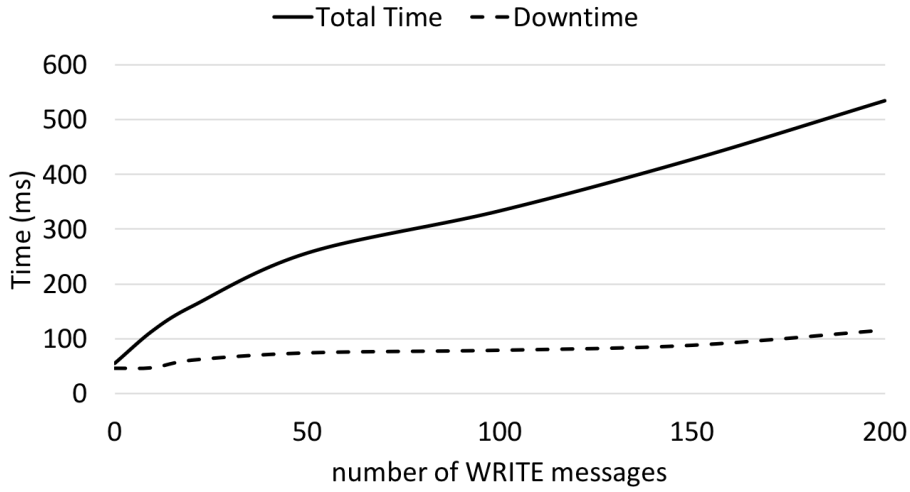
The goals of our experiments are three-fold: (1) evaluate the MEFS mobility management performance, with a focus on service downtime due to migration; (2) evaluate the MEFS fault tolerance performance, with a focus on overhead; and (3) compare the performance of MEFS on EC vs. OFS on MCC. For experiments, we built two mobile apps: an edge-assisted test app that replays the file access traces of real mobile users and a video analytics app, assisted by the edge or the cloud. We use the first app to evaluate the performance of mobility management and test the overhead incurred by the fault-tolerance mechanism in MEFS and use the second app to compare MEFS with OFS. The experiments use a prototype implementation of MEFS running on Android smartphones and Android x86 virtual machines (VMs). The phones act as mobile nodes and the VMs act as edge nodes and cloud nodes. The VMs are hosted in a Linux OS. Each VM runs a 64-bit Android-x86 OS version 6.0 and has 2 virtual CPUs and 2 GB of RAM. The phones communicate with the edge nodes and cloud nodes using a secure WiFi network. The communication between edge nodes and cloud nodes is through wired connections.

5.4.1 Mobility Management Performance

To verify that the migration process of an app component from one edge node to another does not impose significant impact to the quality of user experience, we run several experiments, in which an edge-assisted app performs different file I/O operations when the mobile device switches the edge node it uses. We measured the service downtime and the total time used for migration in two scenarios: (1) the app performs READ operations on the mobile device at different rates: (2) the app performs WRITE operations on the mobile device at different rates. Selecting these two scenarios is to examine the impact of migration separately for READ and



(a) The edge-assisted app performs READ operations



(b) The edge-assisted app performs WRITE operations

Figure 5.9 Service downtime and total time of migration

WRITE operations. The service downtime is the time period during which MEFS cannot respond to any file access requests from the edge component of the app. The total time used for migration is the time period between the creation of a handoff request and the time when the mobile node is connected to a new edge node and the latter finishes the restore phase. The results in Figure 5.9 show that MEFS works well during migrations, and thus it is practical in real-life scenarios. Migrations impose minimal service downtime, which is usually lower than 150ms. For user QoE, 300ms is considered as an acceptable delay [104, 105]. The service downtime of MEFS incurred by migration is lower than this value. The total time used for migration is larger than the service downtime. It is longer than 300ms when the app performs more than 100 READ or WRITE operations per second. However, this does not reduce QoE.

Since migrations happen in the background and are transparent to apps, except for the service downtime, users may not experience any degraded service for most of the time. We also notice that the total time to finish a migration is higher when the app performs WRITE operations than that with READ operations. This is because the total time used for migration is mainly determined by the amount of data that MEFS must copy from one node to the other. When the app performs WRITE operations, there will be more data to be copied to the destination edge node.

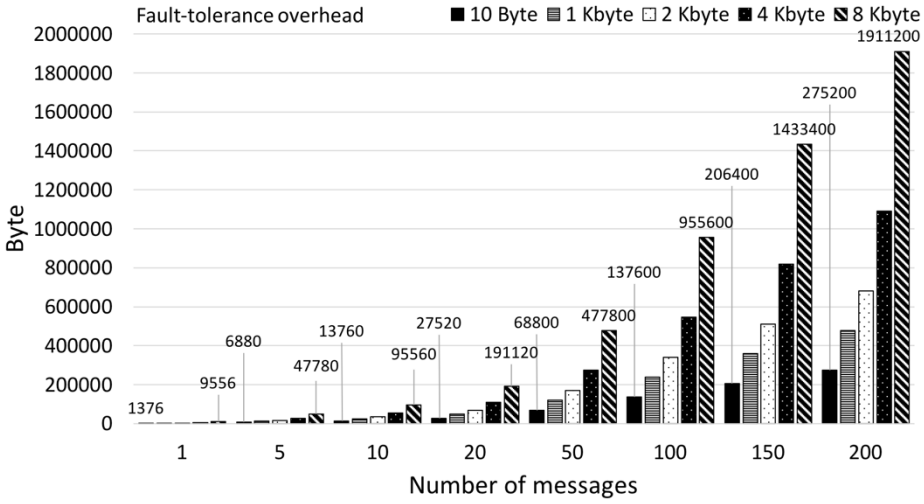
5.4.2 Fault-tolerance Performance

MEFS sends WRITE operations from an edge node to the cloud to tolerate faults at the edge layer. The cost of this mechanism is determined by the amount of data to be transferred between the edge node and the cloud (i.e., the total number of messages and the data's payload). To evaluate the cost, we have measured the amount of data transfer in a few experiments. Figure 5.10a shows the amount of data transfer when the number of messages is varied from 1 to 200, and the payload size is varied from 10B to 8KB. The highest overhead (2MB) is incurred when the number of messages is 200 and payload is 8KB. Since this overhead happens over the wired network, we consider it acceptable for fault-tolerance. The overhead is not proportional to the payload sizes. This is because the overhead is determined by message sizes, which are not proportional to the payload, as shown in Table 5.2.

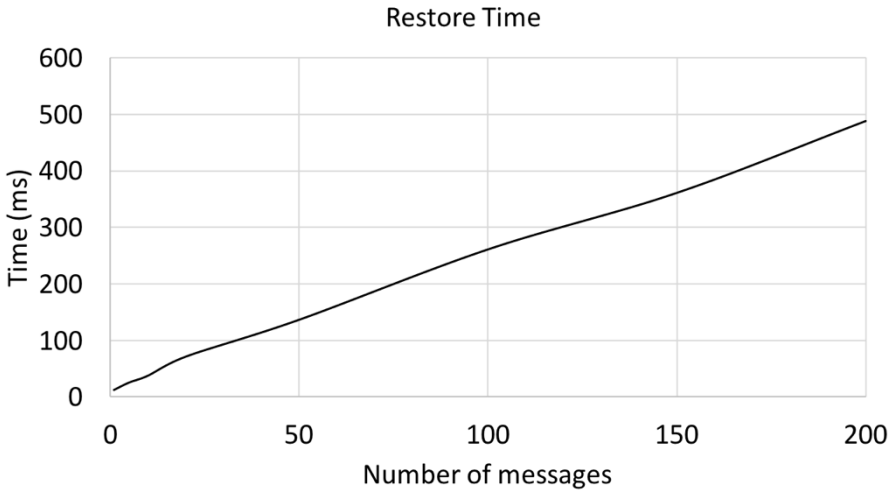
Table 5.2 Size (in Byte) of the WRITE messages

PAYLOAD	SIZE OF MESSAGE
10	1376
1024	2388
2048	3412
4096	5460
8192	9556

We also measured the time to restore the state of MEFS in an edge node based on the data saved in the cloud and show the results in Figure 5.10b. The restore time is also determined by the amount of data to be copied between the cloud and the edge node (i.e., the total number of messages and the data's payload). The restore time increases with the number of messages, as shown in Figure 5.10b (the size of each message is 1376 Byte). When more than 100 messages are needed, the restore time increases to more than 300ms. Thus, we noted that to limit the



(a) Overhead of fault tolerance mechanism



(b) Restore time at the cloud

Figure 5.10 Fault-tolerance performance evaluation

restore time within 300ms, the amount of data transferred between the cloud and the edge node could be smaller than 200KB.

5.4.3 Comparison of MEFS on EC vs. OFS on MCC

To evaluate the benefits of using EC and MEFS over MCC and OFS, we have developed a video analytics application for face recognition purposes. According to Ananthanarayanan et al [106], large-scale video analytics may well represent the killer application for edge computing. The app captures real-time video streams received by a mobile user, analyzes the video streams for faces, recognize people from these faces, and displays the faces. The core component of this app for analyzing the streams, including face recognition with a machine-

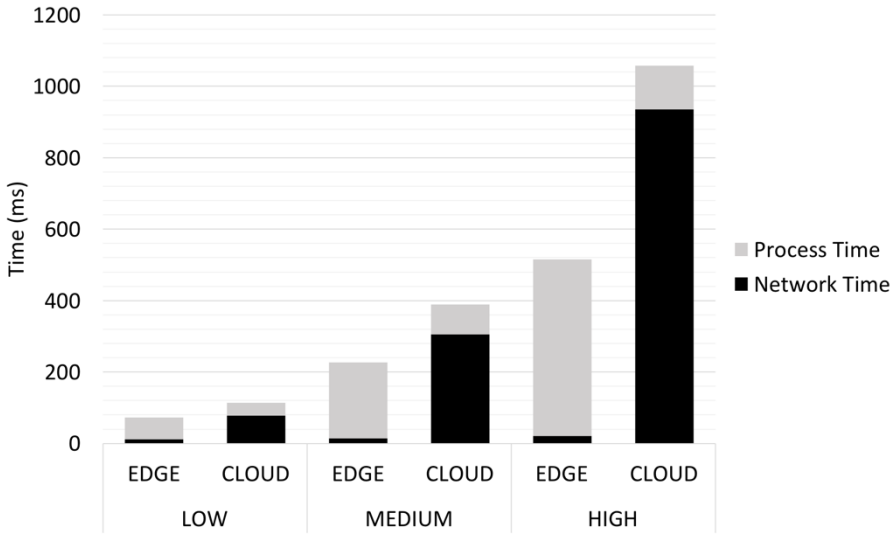


Figure 5.11 Average response time of the video analytics app

learning algorithm, runs at the edge/cloud. The progressive knowledge is also kept in the edge/cloud. On the mobile side, the app component is mainly to send real-time video streams to the edge/cloud and receive the photos generated by the analysis. In our tests, the cloud entity is a virtual machine hosted on Amazon Web Services Cloud equipped with 8 GB RAM and 4 virtual cores; the edge is a Linux box (Ubuntu 16.04 distribution) with 3.1 GHz Intel Core i5 and 4 GB RAM. The mobile node is connected to the edge node via Wi-Fi, and the edge node is connected to the cloud via Ethernet. We have recorded continuous videos for 10 minutes with three different video qualities: a low-quality video with 720x480 resolution, a medium quality video with 1280x720 resolution, and a high-quality video with 1920x1080 resolution. We have measured the response time of the app from the mobile component of the app streaming the video to its edge/cloud component until it receives the photos. The response times are shown in Figure 5.11. Compared to offloading to the cloud, offloading to the edge can significantly reduce the time spent on data transfers and thus reduce response time. For the videos with different qualities, when offloading tasks to the cloud, the response times are dominated by network communication. Due to the high network overhead, the powerful computation capabilities at the cloud cannot effectively reduce the response times. When offloading tasks to the edge, the communication bottleneck can be effectively mitigated. Though the edge node is not as powerful as the cloud node, and the edge node spends more time on computation than the cloud node, the overall response times are lower when offloading tasks to the cloud. The advantage of using edge is more pronounced for the video with the highest quality. The response time with the edge is 50% lower than that with the cloud. These

tests highlight the necessity of using MEC for data-intensive apps and justify the design of MEFS.

5.5 Lessons learnt and Ongoing work

In this research, we presented MEFS, the first mobile edge file system for edge-assisted mobile apps. MEFS provides strong consistency with low latency, and it overcomes MEC challenges such as mobility management and fault-tolerance. Furthermore, MEFS is completely transparent for edge-assisted mobile apps developers. We have implemented the MEFS in Android, and we evaluated it under several experimental scenarios based on real apps and real mobile user traces. The experimental results demonstrated that MEFS can effectively support the user's mobility and fault-tolerance at the edge nodes. Moreover, we proved that MEFS works with low latency at the edge nodes. Therefore, MEFS can be used for many types of context-aware mobile apps, including apps that have tight real-time constraints such as video streaming, augmented reality, and mobile gaming.

The prospects for the future are to provide an extension of MEFS able to work for multiple collaborating users. This is justified because there are a plethora of applications that involve collaborating users, such as car-to-car collaboration or multi-player mobile gaming. This will open new issues and challenges on the MEFS design including the decision in where keep the latest version of the file(s).

6 ADDITIONAL SUPPORT FUNCTIONALITY FOR THE 5GEE INFRASTRUCTURE

This chapter will present an overview of additional support functionality that was designed for the 5GEE architecture. In particular, it will present two major works related to service discovery functionalities and machine learning at the edge. The first section of this chapter will describe the service discovery functionality which has been designed for heterogeneous environments, a typical characteristic of EC networks. The second section will discuss the importance of having intelligence distributed at the edge of the network in order to enable, for instance, accurate predictive models based on machine learning techniques. To ensure this, we will present an architectural model that exploits the EC paradigm and we will show the feasibility of our proposal.

6.1 DRIVE: Discovery service for fully Integrated 5G environment in the IoT

The main goal of this research is to realize a discovery service system capable of overcoming the lack of support for 5G networks. Indeed, as already stated, the main problem of the MEC and Fog is that in most cases the software is embedded within the edge node and operates only for a specific ecosystem. In order to overcome this limitation as well as to evaluate one solution for future 5G networks, we designed and implemented DRIVE, a framework for service discovery functionalities. The framework has been designed to run on both microservers and devices with limited resources (e.g., Raspberry Pi) and to support connection with heterogeneous devices and pluggable services. In the following sections, we will present the design, implementation details, and experimental results of DRIVE based on two primary directions of gateway node improvement: i) dynamic reconfiguration of the edge node, and ii) Docker-based containerization over resource-limited Raspberry Pi devices. On the contrary, we claim that by utilizing DRIVE we significantly enrich the edge intermediate layer by giving the opportunity of exploiting container-based virtualization on top of IoT gateways, with full infrastructure support (download, update, and management of virtualized images based on Docker). To the best of our knowledge, this is one of the first cases of implementation and experimentation of virtualization techniques over edge nodes, while working with IoT gateways with very limited resources, such as Raspberry Pi nodes.

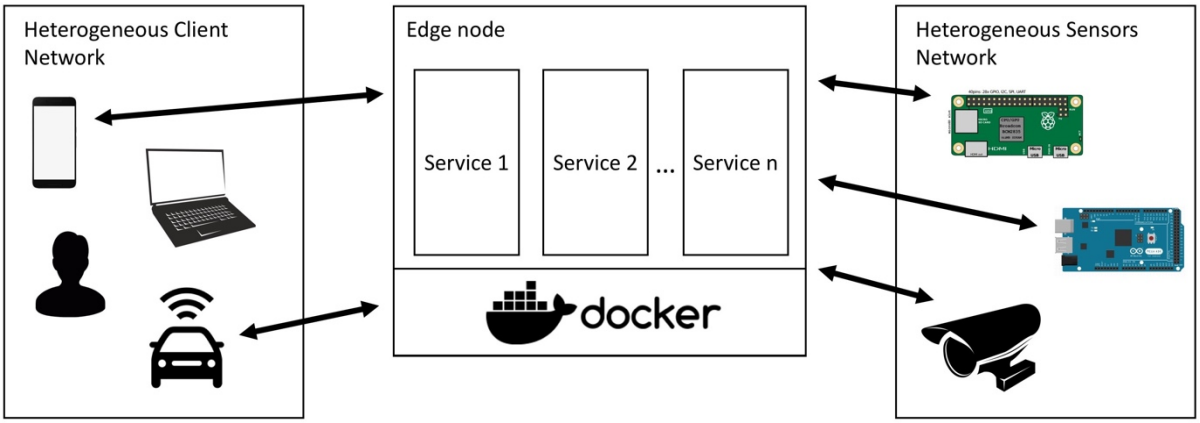


Figure 6.1 DRIVE general architecture

6.1.1 DRIVE: Architecture

DRIVE platform consists of three main components as shown in Figure 6.1, clients network, edge nodes, and sensors network. In the client domain, there are not only smartphones and laptop devices but also various types of devices that connect to the edge node via different communication technologies. On the other side, there are different types of IoT devices that are also connected to the edge nodes via different network connections. The edge node, on the contrary, is based on our 5GEE specifications and operates with both MEC and Fog infrastructures. In general, the architectures mentioned in chapter 2 operate by utilizing a specific hardware and software ecosystem. For instance, as described in the MEC white paper [6], the mobile nodes are connected to the MEC Server via LTE through a Radio – Access Network (RAN) and the services, installed in the MEC server, are accessible via IP communication. On the contrary, in the Fog architecture, the ecosystem is more heterogeneous than in the MEC architecture due to the affinity with the IoT. Moreover, as mentioned in [4], the IoT environment is composed of several heterogeneous "smart objects" each one with different network protocols and different features. The edge node is responsible for offering compute, storage, and networking resources to underlying devices. Thus, our proposed architecture targets to overcome the limitations imposed by each architecture. According to our 5GEE specifications, DRIVE aims to bring the best of the functionalities provided by MEC and Fog. These functionalities are deployed on top of containerization technology (such as Docker Container) in order to simplify the execution and distribution of services across the infrastructure. As result, DRIVE architecture is a vertical multi-layer architecture, composed

of a heterogeneous client network, an edge computing layer, and a heterogeneous sensor layer. All of them are described as follows:

1. *Heterogeneous Client Network Layer*. It consists of a set of heterogeneous clients that want to communicate with edge nodes via different network protocols. The clients must be able to search which services are available in edge nodes and to use those services in the best possible way.
2. *Edge node Layer*. It consists of multiple distributed nodes to provide functionalities for both heterogeneous client network layer and heterogeneous sensor network layer. From the application perspective, edge nodes provide, to client layer, a containers-based self-intelligent application able to perform calculations and statistical analysis on the information sent. In fact, it retrieves data from the sensor layer, stores it into a database, performs analysis and communicates the results back to the client devices layer. The edge layer manages the mobile devices layer providing them the functionalities needed for IoT endpoints to analyze environmental information and to act on the environment accordingly to the goal of the application scenario use case considered.
3. *Heterogeneous Sensor Network Layer*. It consists of IoT endpoints immerse into the environment. IoT devices are all the sensors that sense the information from the surrounding environment and the actuators that modify it. We consider IoT devices as the smallest possible entity, with no internal computational power and only network ability, to send the data towards the nearest edge node. Like the clients, the sensors are able to communicate with the edge node via different network protocols.

6.1.2 DRIVE: Implementation Details

In order to guarantee the maximum interoperability between components, the DRIVE framework has been implemented using three different layers: Application, Session, and Communication layer. Each layer is able to provide service discovery functionalities to a node in the network. We also provide an implementation of the client-side together with an implementation of the sensors layer. Figure 6.2 illustrates the overall implementation of the DRIVE framework and a more detailed description of all layers can be found below:

- **Application Layer**. This layer contains the implementation of the REST APIs for client requests. The APIs contain the invocation of functionalities provided by sensors. This layer was implemented in Java using Retrofit API [107] and allow clients to bypass the Service Discovery protocol. We have realized several APIs, such as *get_All_Services*,

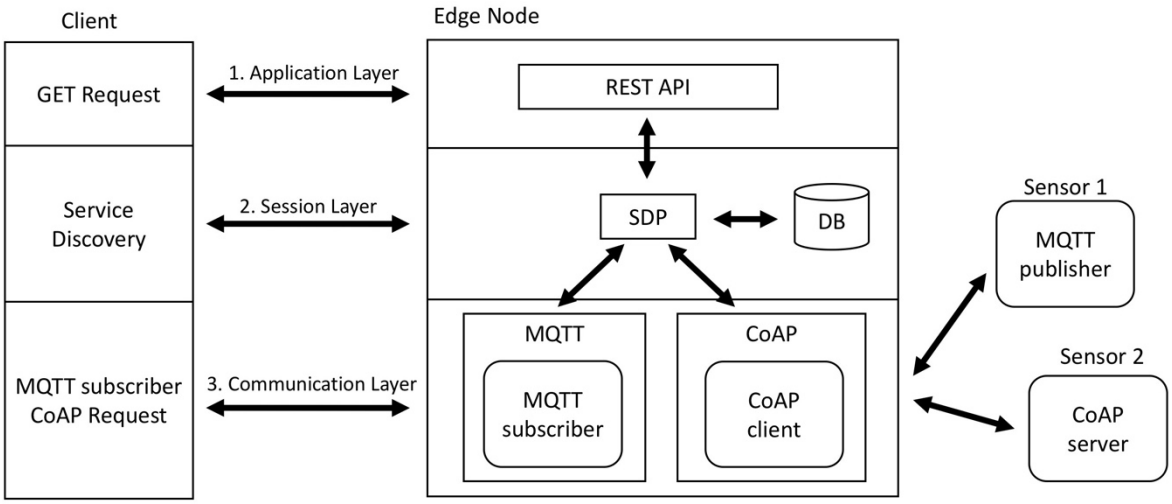


Figure 6.2 DRIVE implementation details

for retrieving the list of all services deployed in the edge node, *get_Humidity*, *get_Aggregate_Humidity*, *get_Temperature*, *get_Aggregate_Temperature* and so on, for invoking a specific service.

- **Session Layer**. This layer includes the Service Discovery Protocol and the Database. The JmDNS [108] library that was used to develop this layer is an implementation of multi-cast DNS in Java. The multicast Domain Name System (mDNS) resolves hostnames (in our case, it is service names) to IP addresses within small networks. It is based on zero-configuration [109], [110], this means that we don't have to configure any network setting programmatically to make it work. It used the same programming interfaces, packet format, and operating semantics as the unicast DNS. The job of mDNS is to send IP multicast User Datagram Protocol (UDP) packets that include the services to a certain multicast address. Then all mDNS capable hosts (the android client is the host we are interested in) listen to this address. The client will discover the services by specifying the service type in the code. In particular, the first step in this procedure is to find the name of the proxy that provides the services. This will give us the mDNS name of the proxy. Then the second step is to resolve this name of the host using mDNS. By multicasting the name, the proxy will listen to the packet that looks for him and responds via broadcast with its IP address, port number and return the services.
- **Communication Layer**. By utilizing this layer the clients are able to communicate directly with the participant sensors in the network. This is possible because there is a block of software that allows the client to communicate with the sensors. For instance,

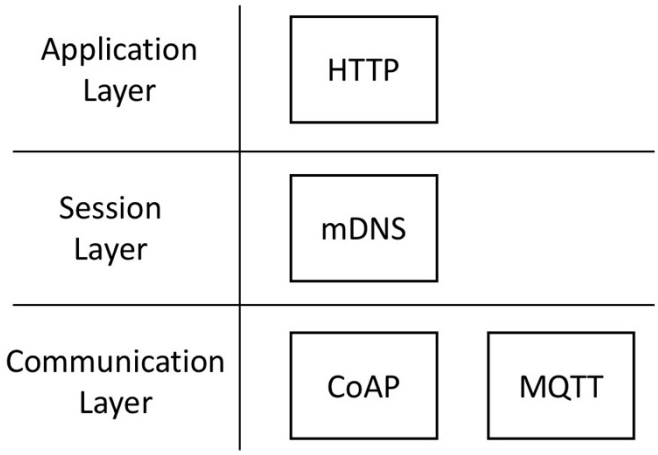


Figure 6.3 DRIVE protocol stack

in our implementation, there is a CoAP server and an MQTT publisher inside the sensors.

Figure 6.3 clearly reports the whole protocol stack showing all protocols specifically used on each stack layer. Message Queuing Telemetry Transport (MQTT) and Constrained Application Protocol (CoAP) [111, 112] are the two main protocols used in this implementation to able to send and receive values from different devices. Two of the most promising protocols for small devices that are very well suited to constrained environments. They provide mechanisms for asynchronous communication and are open standards. CoAP is a specialized web protocol based on the REST model. More extensively, the REST model makes possible for servers to create resources and let the clients access these through a URL using methods like GET, PUT, POST, DELETE. On the other side, MQTT follows a different logic, it is a lightweight publish/subscribe messaging transport protocol that is ideal for mobile applications due to its small size, low power usage, minimized data packets and efficient distribution of information to one or many receivers. A broker is required to exist in the middle of the two ends (the subscriber and the publisher). The client was realized with an Android smartphone. The implementation presented in this work is based on independent layers so one can simply implement only one layer of the three or only the layers that he/she needs. The Android application includes an integration of the MQTT and CoAP communication protocol, the JmDNS library on the session layer and the Retrofit API which makes it possible for the user to connect with the REST APIs by sending HTTP requests and receiving JSON objects as java objects. Finally, we realized sensors using Raspberry Pi Zero and installed on them the right software for communicating with the edge node and the client as well. For instance, we

provided the implementation of JmDNS in order to register the services on edge node as well as the implementation of MQTT and CoAP protocols.

6.1.3 DRIVE: experimental results

We widely assessed and validate the feasibility of our framework for service discovery. The current sub-section presents a significant selection of experimental results obtained in our lab deployment scenario. We carried out several different sets of experiments including experiments on the session layer and application layer. The prototype of our experimental environment is set up from clients/sensors to the edge nodes. The edge node is a Raspberry Pi 3 Model B equipped with a 64-bit quad-core ARM Cortex-A53 processor, 1 GB of RAM and 16 GB of storage space, Wi-Fi and Bluetooth connection. Instead, the two sensors are achieved by Raspberry Pi Zero equipped with 1GHz single-core CPU and 512MB RAM. Finally, the client is an Android-based smartphone (Android 6.0.1 Marshmallow version and Android API level 23).

Experiments on the Session Layer

In this series of experiments, we stressed out the edge node by sending service requests via the session layer. In the case of CoAP services requests, the edge node asks to the sensor for a new value. Instead, for MQTT services requests the edge node uses MQTT subscriber and the database for saving and retrieving new values. To be able to stress the edge node in a more realistic scenario, we used the Poisson distribution that we implemented within the test application that makes service requests. The algorithm that represents the Poisson distribution was embedded in the implementation to mimic realistic arrival patterns. The Poisson algorithm receives as a configuration parameter the lambda value to change the frequency of incoming requests. We measured the CPU consumption and bandwidth usage at Raspberry Pis during service requests. We used the same tools used for measuring CPU and RAM consumption in MESH, namely RPi-Monitor. As clearly shown in Figure 6.4, the maximum CPU consumption is around 50% which averages that up to 2 cores are used over the 4 total cores available (100% means that all 4 cores are used). The tests considered the execution of the service requests from the client to the edge node, we carried out the experiments using several lambda values between 1000 and 10. In this dissertation, we report the experiments with 10 as lambda value for the Poisson algorithm (that mimics a realistic scenario with 10-100 requests-per- second), over a 10-minutes observation period. We described the behavior of our work over time to show the stability of the framework. The results are shown in Figure 6.4 and Figure 6.5 respectively for

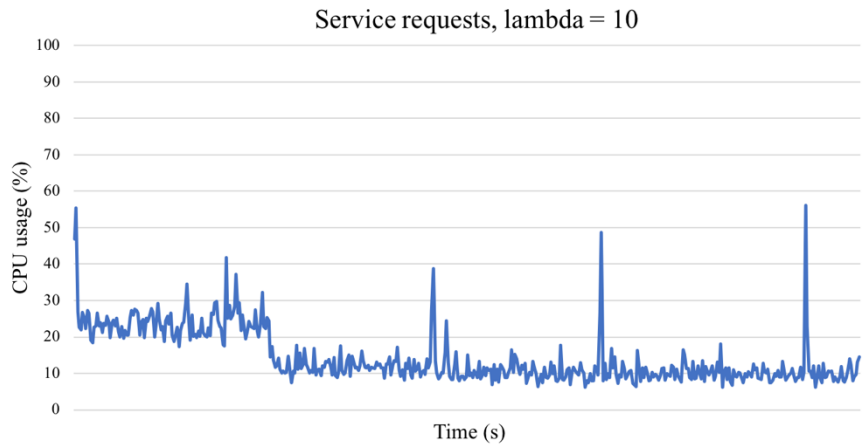


Figure 6.4 Performance evaluation for service discovery: CPU usage

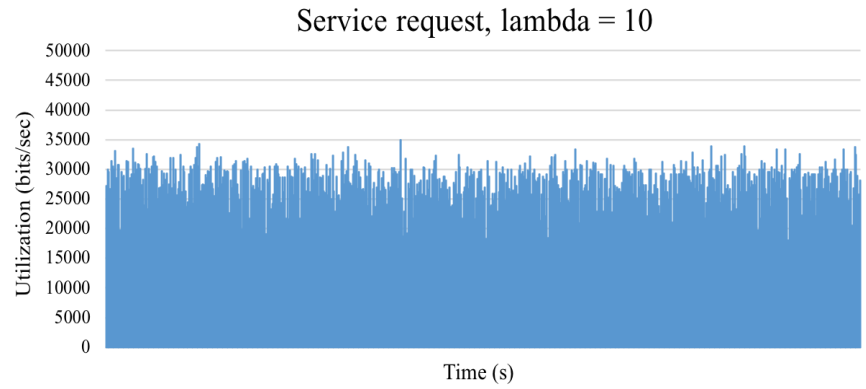


Figure 6.5 Performance evaluation for service discovery: bandwidth usage

CPU usage and bandwidth usage. It is possible to see that the amount of bandwidth used is extremely low and does not exceed 30 KB due to the good characteristics of communication protocols. As for CPU usage, the results show that the processing load of the edge node is around 15%.

Experiments on the Application Layer

In this series of experiments, we stressed out our edge node by sending continuous HTTP requests. The edge node is able to accept HTTP requests via the Application Layer. Moreover, we simulate multiple requests done by clients. For this purpose, we have used Apache ab (for further information, see https://httpd.apache.org/docs/2.4/programs/ab.html), which is a tool for benchmarking an HTTP server. As done for the previous experiment, we measured the CPU consumption and bandwidth usage during HTTP requests at the edge node. As shown in Figure 6.6 and Figure 6.7, the average use of CPU is around 15%. While the amount of bandwidth

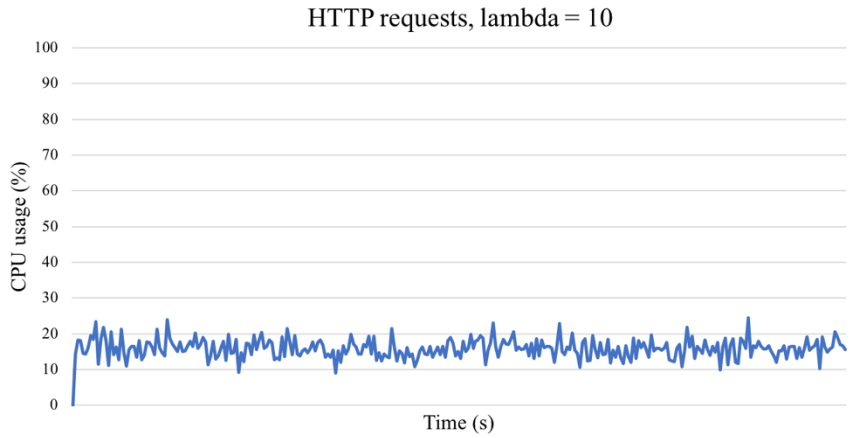


Figure 6.6 Performance evaluation for application layer: CPU usage

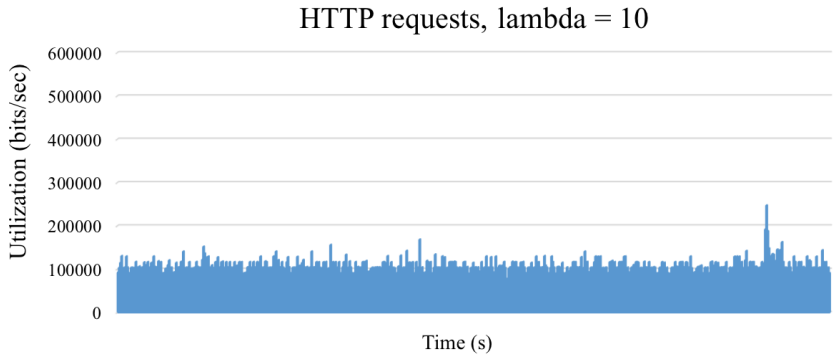


Figure 6.7 Performance evaluation for application layer: bandwidth usage

usage has increased slightly due to the size of objects exchanged between the edge node and device.

6.1.4 Lessons learnt and Ongoing work

In this research, we have introduced DRIVE, a new framework for service discovery functionalities for a fully-integrated 5G environment in the IoT. In order to work within a heterogeneous environment, a typical scenario for future 5G networks, DRIVE operates at three different layers of the protocol stack: i) at the Application Layer, it is able to give fast access for smart clients such as smartphones and provides several functions that are used to enable resource discovery and interoperability among applications, ii) at the Session Layer, it provides a fully-access to service discovery functionalities using DNS facilities, and iii) at the Communication Layer, it provides a direct communication with sensors in the network, without using any service discovery facilities. The experiments have been conducted in a real-world scenario comprising the deployment on a resource-constrained device such as Raspberry Pi.

DRIVE introduces significant benefits and has proved to be an enabler for resource discovery for a fully-integrated 5G environment in the IoT. Even if we did not present dedicated experimental results on DRIVE scalability, as shown in the experimental results section (6.1.3) the lambda value is set on a low value to simulate a high number of incoming requests. Therefore, the obtained results can be read, also, in order to show the DRIVE scalability.
Encouraged by these positive results, possible new related research scenarios can be categorized in two significant directions. On the one hand, the possibility to explore the impact of heterogeneous storage and overall performance on the basis of diverse services. On the other hand, to study the distribution and coordination of the DRIVE framework on multiple devices to grant scalability in widely distributed large deployments.

6.2 A Support Infrastructure for Machine Learning at the Edge

This research aims at presenting an infrastructure to support distributed intelligence at the edge of the network by enabling edge devices to collaboratively learn a shared model while keeping local intelligence at the edge. In addition, edge nodes have the possibility to improve the global model (generally, stored at the cloud) by sending reinforced local models towards the cloud. This proposed infrastructure finds its outlet in several scenarios including smart environments (smart home and smart city) where there is a high rate of collaboration between entities and in IIoT environments where it enables real-time process optimization, quality inspection, and preventive diagnostics. Those scenarios are characterized by an enormous quantity of data generated. However, to extract valuable information and consequently producing real-time analytics, Machine Learning (ML) techniques are often applied. The definition of ML is very broad, ranging from simple data summarization with linear regression to multi-class classification and deep neural networks [113]. One key enabler of ML is the ability to train models using a very large amount of data. With the increasing amount of data being generated by IoT devices, we claim that ML tasks will become a dominant workload in distributed edge-enabled IoT systems in the future. However, it is challenging to perform distributed ML on resource-constrained EC systems. To address the above relevant gaps in the existing solutions, we propose an original support infrastructure for running ML algorithms in a collaborative EC architecture (3-layers architecture) where edge nodes are wired-connected with the cloud and wireless-connected with heterogeneous devices. The proposed solution has the following primary innovation elements and features. First, in our infrastructure raw data is collected and

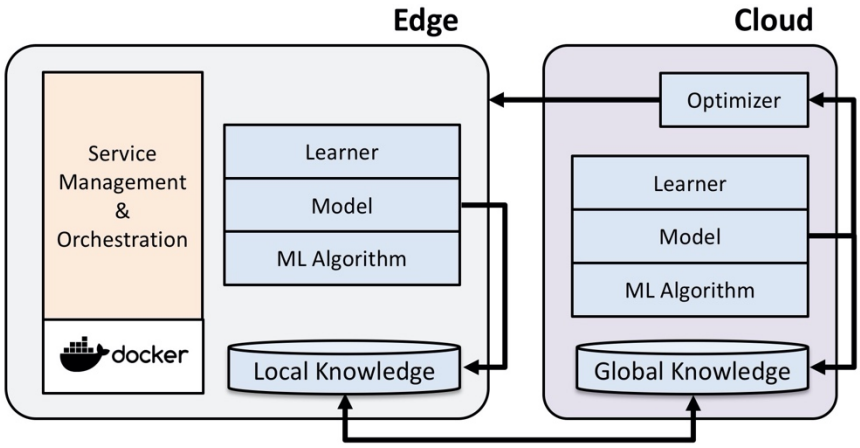


Figure 6.8 Proposed Architecture

stored at multiple edge nodes, and an ML model is trained from the distributed data without sending the raw data from the nodes to the cloud. Second, the infrastructure includes global aggregation steps where model parameters, obtained at different edge nodes, are sent to the cloud, which contains the global model. Third, we presented the primarily guidelines of our distributed architecture that enables ML at the edge of the network. Finally, we presented two real cases one based on a video streaming processing for face recognition (typically deployed in a smart city environment) and another for predictive diagnostics in an IIoT environment. To demonstrate the benefits of using a collaborative approach for learning-based algorithms we quantitatively evaluate the advantages of processing local tasks based on knowledge at the edge of the network.

6.2.1 Proposed Architecture

Since our objective is to increase the interoperability of edge nodes and collaboration among them, we propose an EC architecture platform that facilitates communication between edge resources and enables intelligence at the edge. As one can see from Figure 6.8, our proposed architecture consists of five main building modules: i) a Service Management and Orchestrator support based on OpenBaton to take over all needed management issues; ii) a set of ML algorithms including decision trees, regression trees, random forests, gradient boosting trees, neural networks, and deep networks for giving back a prediction; iii) a Learning module able to train the model via basic types of learning algorithms including Supervised Learning, Unsupervised Learning, and Reinforcement Learning; iv) a Model that accurately represent a system (infrastructure, process, machine, etc.); v) an Optimizer module that sends feedback about models in order to enhance and optimize them.

- **Service Management and Orchestrator**. This module is in charge of managing services running at the edge. We developed edge services, including all machine learning features, as a Docker Container. Moreover, it is compliant with all 5GEE specifications including OpenBaton as resources manager. Services and functionalities are developed as a Docker container which guarantees the maximum possible interoperability between nodes. Furthermore, container-based virtualization enables working with IoT gateways with very limited resources, such as Raspberry Pi nodes. From the point of view of a network administrator, a services orchestrator is a key enabler for a dynamic architecture.
- **ML algorithm**. This module contains a composition of ML algorithms and mathematical models able to take decisions based on knowledge gained from a learning algorithm. In our architecture, ML algorithms are built as a Docker container and are running both at the edge and at the cloud of the network. Usually, the model enables the description of a system itself and its dynamics. Indeed, ML algorithms are used for instance to detect and diagnose anomalies, to determine an optimal set of actions, to enforce the quality of production processes, and to provide predictions for strategic planning.
- **Learner**. This module is in charge of training the model used by the ML algorithm. Normally, the training phase is a time-consuming task and requires several computational resources. It could be challenging to perform training algorithms on resource-constrained edge nodes. Despite local updates consume computation resources of the edge node, this module allows training the model both at the edge and from the cloud.
- **Model**. This module contains the knowledge of the node. As already stated, in our architecture, we claim on the possibility to have local models defined at the edge nodes (that generates several local knowledge), and a global model defined at the cloud (that generates a global knowledge). The main idea is to host ML models, already trained at the cloud side, at the edge of the network with the possibility to improve the local model via the interaction with underlying devices (from mobile devices to Industrial IoT devices). Finally, local models at the edge return feedback to the cloud manager in terms of models, and updated the global model at the cloud, by exploiting collaboration among them.

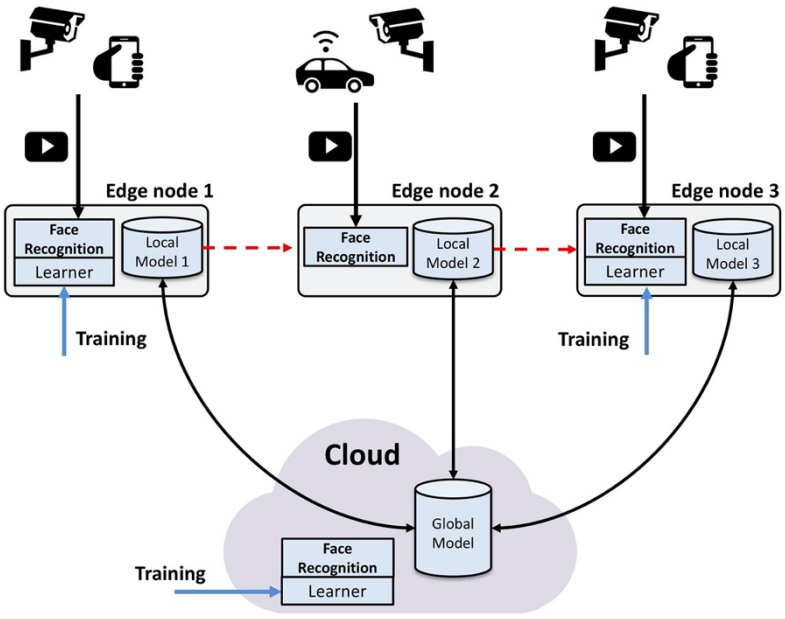


Figure 6.9 Secure City use cas

- **Optimizer**. This is a logical component that runs at the cloud and is in charge of optimizing the model. In particular, data generated at the edge of the network may grow local models which are also forwarded to the cloud for ML models optimization. Therefore, this module selects the more appropriate model and sends it back to selected edge nodes. On the contrary, ML models already trained at the cloud side may be sent back for model optimization towards edge nodes.

6.2.2 Use cases and Experimental Results

In this section, we will present two use cases in order to illustrate our collaborative edge machine learning vision comprehensively. In particular, we extensively analyze a face recognition application in the field of secure city, where the system has to recognize "dangerous people" from video streaming. Second, we analyze a predictive maintenance task in the field of IIoT where the system has to monitor the operation of a machine. In either case, the experimental results show great benefits of using the proposed approach.

Secure City use case

Let us describe how this scenario works: a person using a smartphone, a surveillance camera, or anything that can capture video can stream the video to the local edge node. Previously, local edge nodes have been trained at the edge by the learner module or by the cloud. Then the edge is doing all the necessary procedures to detect faces in the video stream. If a "dangerous

person" is recognized by the edge node, an alert is sent to the nearest police station including the location of the camera. Figure 6.9 shows how our distributed architecture works in the mentioned context. As already stated, a 5GEE node is originally set to provide telco functionalities and a few additional network support functionalities. In a moment, for instance, when police need to provide face recognition functionalities, the available functionalities over edge nodes in given localities must be updated via the orchestrator in order to support the police via (as it was done to support Alice and Bob tasks, chapter 3). Note that each edge node has its local knowledge. On the one hand, the knowledge could be initialized by a training phase, in this step, we will use training data to incrementally improve the edge's ability to predict "dangerous people". On the other hand, the knowledge could be shared among edge nodes and the cloud. In the latter case, one edge node may even be trained by another edge node; the same thing could be possible at the cloud as well. The cloud can train edge nodes at the beginning or when something happened also can improve its knowledge from edge nodes. We implemented this testbed by using Python and Android SDK. The first for face recognition tasks and the second for video recording functionalities. In detail, the Python script takes the video streaming from a smartphone and reads it frame by frame. For each frame, the script recognizes (if exists) a face by comparing the face histogram with the ones in the .yml file. If a face is recognized, the id of the person appears. The recognition is being made by the LBPH algorithm again with a 90% - 94% accuracy rate for both frontal and side face, this depends on the image quality [114]. The trainer (Learner module) is implemented by another script Python. The goal of this script is to create a knowledge able to recognize faces starting from a dataset. For the purposes of this work, we used existing faces dataset in the Kaggle website [115], which consists of 152 people and 20 images per person. The image resolution is 180 x 200 pixels. We trained the

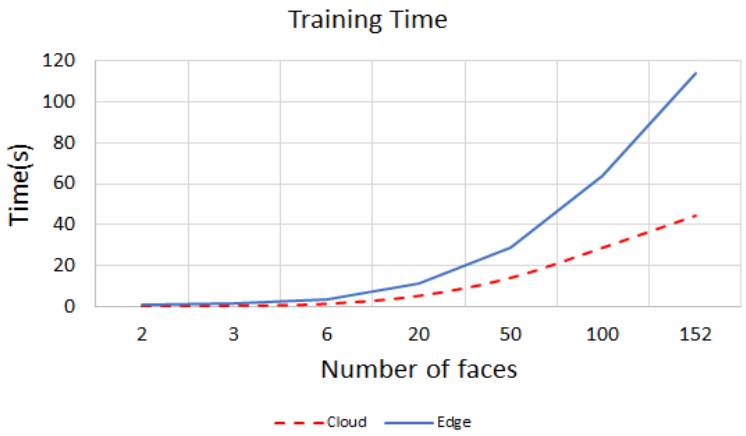


Figure 6.10 Training time at the edge node and at the cloud

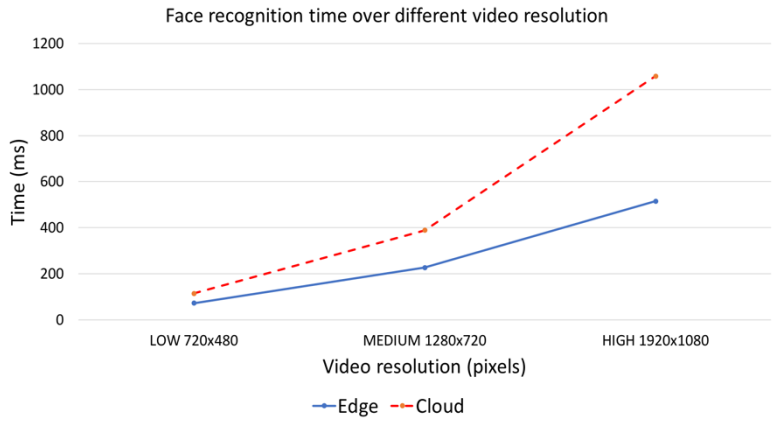


Figure 6.11 Total recognition time over different video resolution

classifier at first with 2 people, then for 3, 6, 20, 50, 100 and finally 152 from the Kaggle dataset. Figure 6.10 shows the time needed for training a classifier both at the edge node and at the cloud. As one can see, the training time exponentially increases when the dataset grows. Moreover, due to resources limited at the edge, the training phase performs better at the cloud. In the literature, most of the efforts are focused either on finding novel training algorithms and on exploiting distributed architecture in order to reduce training time. In this work, we studied the possibility of having knowledge shared among the edge nodes and the cloud in order to exploit the infrastructure, therefore there is no need to train all edge nodes with the whole knowledge. However, for small datasets, training at the edge performs almost as at the cloud. We also evaluated the recognition script by measuring the response time of the app from the device of the streaming the video to its edge/cloud component until it recognizes the faces. In particular, Figure 6.11 shows the face recognition processing time by the edge node and the cloud when the video quality grows. In the cloud the recognition time goes up rapidly as the video quality increases. This because, for videos with high quality, when executing recognition at the cloud, the response times are dominated by network communication. Due to the high network overhead, the powerful computation capabilities at the cloud cannot effectively reduce the response times. When sending video to the edge, the communication bottleneck can be effectively mitigated. Though the edge node is not as powerful as the cloud node, and the edge node spends more time on computation than the cloud node, the overall response times are lower when performing face recognition to the edge. The advantage of using edge is more pronounced for the video with the highest quality. In this case, face recognition at the edge is 50% lower than recognize faces at the cloud. We also measured the face recognition time at the edge and at the cloud when the number of faces increases. Figure 6.12 shows the face

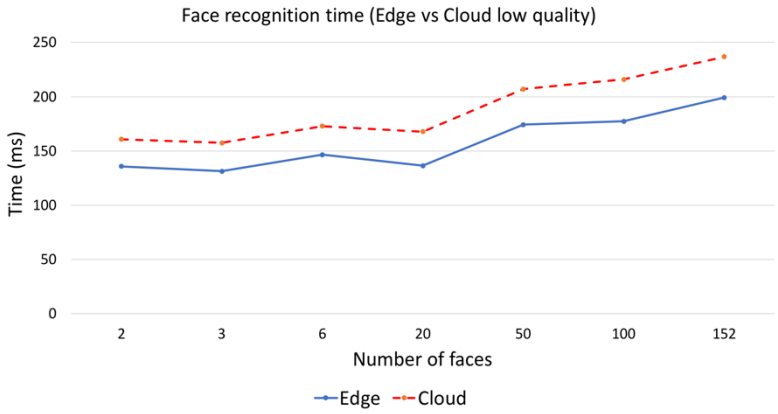


Figure 6.12 Total recognition time for low quality video at the edge ad at the cloud

recognition time variation with a different number of faces, from low-quality video, for the edge node and the cloud. The choice to run experiments with low-quality video is justified by the fact that is the best scenario for the cloud. Anyway, in both cases, the face recognition time remains mostly invariable or slightly increases. This demonstrates the scalability of the edge node when the number of faces increases.

IIoT use case

This section presents a real case study in order to illustrate how to leverage distributed digital twins of production plants and facilities, enabled by off-the-shelf edge/cloud technologies for big data processing. Digital twins are essentially models that accurately represent a system (processes or machines) by using, generally, data generated by IoT. Among the many things these models enable: i) the description of systems; ii) the prediction of systems evolution; iii) the management and maintenance of systems. They are used to detect and diagnose anomalies, to determine an optimal set of actions that maximize key performance metrics, to effectively and efficiently enforce on-line quality management of production processes under latency and reliability constraints, and to provide predictions for strategic planning to help companies to significantly improve their profitability through digitalization, as well as to open up new opportunities for them for the creation of new services and business models. In particular, this testbed is aimed at creating a digital twin of the air pressure system at Scania trucks by aggregating ML models of single trucks for predictive maintenance. Data are used to detect the health status of the APS, to predict failures, and to plan maintenance operations for reducing unexpected breakdowns. The targeted trucks will be divided into several sub-parts/systems devoted to specific tasks, with specific raw monitoring indicators, such as temperature

indicators of each component part. Each component of the system provides data for the monitoring tasks that are properly channeled to closest edge node to produce digital twins enriching the observed data themselves. ML techniques are trained to produce predictive models on health status and to allow the edge to localized anomaly detection. Trained predictive models are then transferred to the cloud to reinforce the global model which, if it is improved (e.g., in terms of accuracy), will be returned in each edge node. Also, the cloud off-line optimizations provide maintenance plans and guidelines for future improved design. In order to simulate the scenario just described, we used an open-source dataset that consists of data collected from Scania trucks in everyday usage [116]. The system in focus is the Air Pressure system (APS) which generates pressurized air that is utilized in various functions in a truck, such as braking and gear changes. Therefore, failures should be predicted before they occur. Falsely predicting a failure has a cost of 10, while missing a failure has a cost of 500. This makes sense because is simpler to manage the case of false positive rather than the case of a missing a failure. The data contains a training set and a test set. The training set contains 60,000 rows, of which 1,000 belong to the positive class and 171 columns, of which one is the class column and the test set contains 16,000 rows. Let us point out that fault diagnosis and prognosis in mechanical systems are considered hot topics for future smart factories in the field of Industry 4.0. This testbed is just an example that can be applied in any IoT diagnostics system. The first test family that we have done for this testbed regards the model accuracy variation. Please note that the edge nodes send reinforced models, with fresh data, to the cloud. Figure 6.13 shows the model accuracy variation in light of edge updates. To trace the quality of the model we considered the recall and specificity that represent respectively the percentage

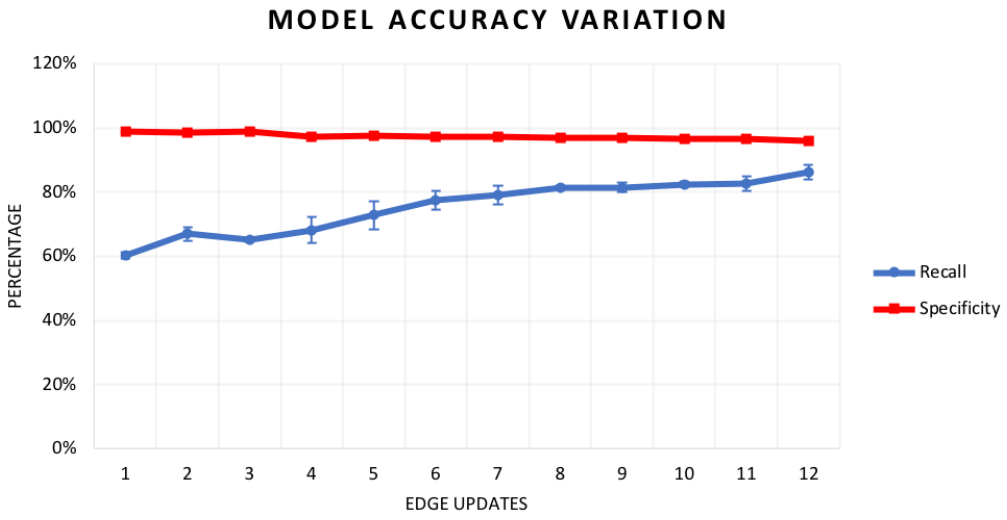


Figure 6.13 Model accuracy variation

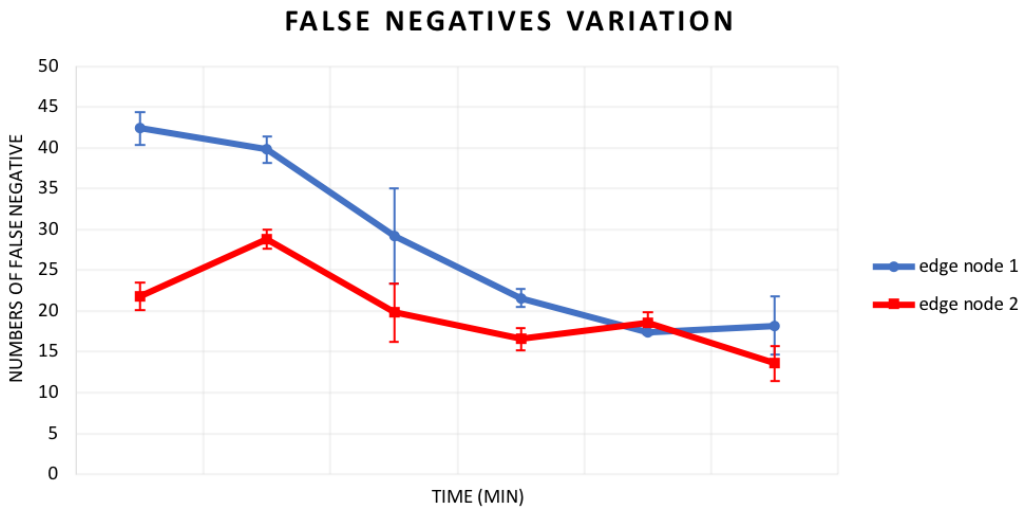


Figure 6.14 False negatives variation

of negative instances within the total and the percentage of positive instances within the total. As one can see from Figure 6.13, the specificity goes down from 0.98 to 0.96 but the recall rising to 0.87 with an increase of 0.3. This because the initial dataset contains a number of negative instances very low compared to the total dataset. Therefore, updating the model with fresh data allows creating a model that the total accuracy is more or less the same but with more accuracy for predicting negative instances. To demonstrate this, Figure 6.14 shows how the number of false-negative instances recognized at the edge decreases over time. For the last set of experiments, we compared how the model size can impact the latency time between edge nodes and the cloud. As shown in Figure 6.15, for consecutive model updates, the experimental results show that the models stored in each edge nodes grow in a linear fashion. In our specific

Figure 6.15 Model size variation

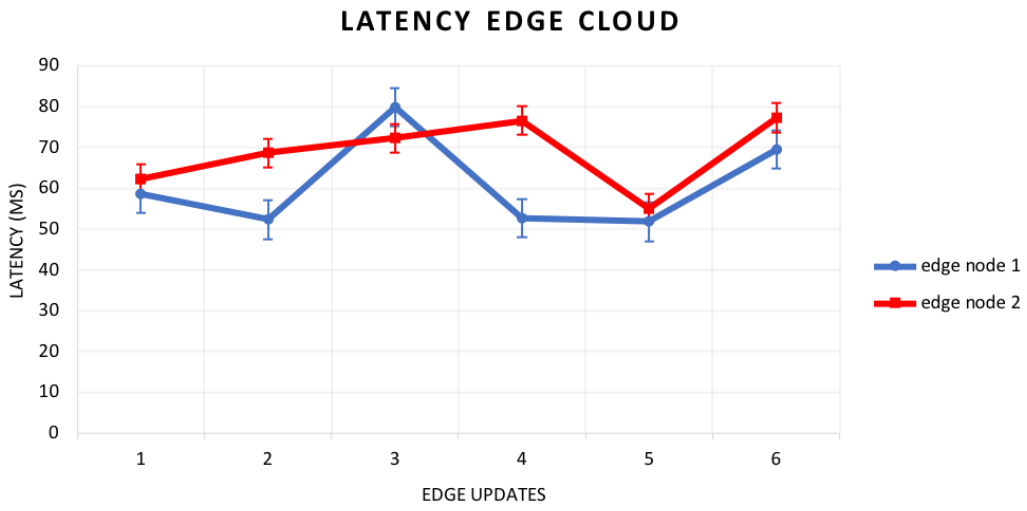


Figure 6.16 Latency edge-cloud

use case, the model increases its size by approximately 50% (i.e., from 6KB to 12KB). Additionally, as shown in Figure 6.16, we measured the latency time introduced from these consecutive model updates between edge nodes and the cloud. Actually, in this work, we do not consider the other side of the latency time between the cloud and edge nodes. This because it's around the same values: 60ms of average latency time. Anyway, in our use case, the latency time introduced does not vary much compared to the model size variation, rather remains the same. Let us note that this delay is completely transparent to the logic of the application. The model updates can be made in a background mode.

6.2.3 Lessons learnt and Ongoing work

In this research, we proposed an edge-enabled cloud-assisted system for distributed ML models covering advanced ML algorithms. This work paves the way for ML at the edge by proposing a support infrastructure and discussing two relevant use cases. From them, we learned that for smart distributed environments we need to enhance the collaboration between entities by, for instance, distributing the knowledge among them. On the contrary, for industrial environments, we need to process data fast in order to produce. valuable outputs for the prediction of systems evolution. The reported results confirm that advanced ML algorithms can be executed at the edge with good performances. On the one hand, results show that the face recognition algorithm at the edge has good performance with low-quality video (cloud best scenario) as well. On the other hand, reinforcing the model at the edge of the network produces several advantages in predictive diagnostics scenarios. Fueled by these significant results, we identified two main research directions. The first direction is to work on the Optimizer module in order

to find novel solutions for model aggregation. The other is to investigate the impact of different ML algorithms in order to improve the cooperation between several IoT devices.

7 CONCLUSION AND FUTURE WORK

The integration of MEC and Fog has surely and remarkably leveraged all the most common issues of the EC paradigm. The introduction of the 5GEE architecture made feasible all the advantages of the Fog paradigm and MEC infrastructure, firstly with a smart combination of their functionalities. Although a complete, comprehensive, fully integrated, framework for the EC networks is still in the early stage, this project has proved the complete feasibility and practicability of idea and the concept. By exploiting both the Fog and MEC the project has overcome:

- **Device Heterogeneity**. The 5GEE architecture totally abstracts the physical layer of devices, overcoming all their hardware, technology, and communication protocol differences. The 5GEE architecture provides heterogeneous communication functionalities (both from the Fog and MEC) to the underlying devices and their functionalities available at the edge.
- **Service Availability**. The 5GEE standardizes the manner in which services are built and deployed by using the Docker technology as the de facto standard for container-based edge services. This enhances the service availability as well as among different edge nodes. Moreover, 5GEE gives the possibility of running services at the edge of the network.
- **Power Efficiency**. This project, through the introduction of 5GEE infrastructure, enhances the power sustainability both of mobile nodes and edge nodes. Providing energy efficiency also for edge nodes is a crucial aspect for future 5G networks. The 5GEE allows to offload intensive tasks for mobile devices to the closest edge node in order to preserve the mobile device battery. In addition, this research provides a strategy based on the service affinity value in order to choose the best edge nodes towards moving the service in terms of energy consumption.

Moreover, this dissertation has proved the feasibility of new service applications in the EC paradigm. This is only the first step towards the definition of a new paradigm and modus-operandi for edge-enable infrastructures. Starting from the 5G network on, telco providers will be able to provide services running at the edge of the network. This will lead to a huge cost restriction and a huge number of available nodes spread on the territory making the telco service pervasive and with much higher performance. This project has presented the support infrastructure for edge-enabled 5G networks and the most essential services as well.

In the following section will be summarized the achieved results, then, in the last section, we will introduce future research directions.

7.1 Achieved results

This section will address and discuss the achieved results of this research. First, the aspect that emerged from the proposed service migration strategy will be addressed, then, the focus will be moved on the results achieved by MEFS. Finally, the discussion will be moved on the results obtained from the service discovery functionality that has made a contribution to overcoming the heterogeneity of the devices and the possibility to leverage 5GEE architecture to running ML procedures at edge layer.

7.1.1 Service Migration

The service migration process is still one of the most important open issues in the field of the EC. This feature is crucial for improving the QoS and QoE of service provisioning in the edge-enable 5G networks. Our proposed solution, named MESH, leverages container-based technologies and effectively triggers the handoff process thus granting low service downtime. Furthermore, we introduced techniques to speed up the handoff process based on service characteristics. In particular, we have developed a novel strategy based on data access frequencies. Thus, data with lower access frequencies will be moved early to the target edge node. This allows MESH to work in a proactive way, moving much data as possible to the target edge node before the handoff happens. The results obtained from this research have been encouraging and have shown the advantages of using our approach.

7.1.2 Task offloading

In the field of MCC, the task offloading process has already been addressed from several works. Some of them, offload tasks in terms of procedures, threads, or files. It has recently been demonstrated that having shared file systems between device and cloud can improve task offloading procedure in terms of time and consistency. Hence, this research project has proposed a shared file system between three entities such as mobile devices, edge devices, and the cloud in order to enable task offloading service in an edge-enable infrastructure. Furthermore, it has addressed two major challenges of the EC infrastructure such as mobility and resilience. To figure out the user mobility, we used part of the work did in MESH and we proposed an efficient solution based on live migration. On the contrary, we also proposed a

solution for guaranteeing system resilience and fault tolerance. The fault tolerance is a crucial point when we move from the cloud to the edge of the network, edge nodes are more inclined to failures.

7.1.3 Device Heterogeneity

Service discovery, data aggregation, data analysis, and so on are a set of services specified for heterogeneous devices. As 5GEE gives the possibility to connect devices from Fog-based networks and MEC infrastructures, this research project has proposed a multi-layer approach for service discovery functionalities in order to work with the heterogeneous devices. The multi-layer architecture allows end devices to communicate with the edge node by using one of the specified layers. In this way, the end device can request a specific service through the more appropriate layer thus reducing the devices heterogeneity problem of the EC paradigm.

7.1.4 Intelligence at the edge

Among the many things that could be moved at the edge of the network, we studied the importance of distributing intelligence. This project proposed a three-layer architectural solution that brings AI techniques on each layer of the architecture. This will enable complex real-time elaborations that permit us to detect and diagnose anomalies, to determine an optimal set of actions, to effectively and efficiently enforce on-line quality management of production processes, and to provide predictions for strategic planning.

7.2 Future Work

To conclude this thesis, in this last section, some future research directions will be addressed. As already stated, and despite the great result already achieved by this research, this project is only the first stage of broader and ambitious research.

7.2.1 5GEE research directions

The 5GEE architecture is already proposed as a reference architectural model and technology for future edge-enable 5G networks. The first possible development of our 5GEE architecture should enable the self-adaptable of 5GEE nodes to the different dynamics and variations of the city pulse, for instance to the different behaviors that might present along the year, such as working vs. vacation periods, and the week, such as working days vs. weekends. The dynamic adaptation is the central core of the future 5G networks that are intended to provide connectivity

to future smart cities. A valuable extension of this work would be to implement and to test a new kind of services orchestrator with relevant schedule algorithms. Moreover, it would be interesting to develop a services scheduler algorithm that leverages the user's traceability data in order to minimize the cost of 5GEE nodes reconfiguration. For example, an orchestrator that dynamically can provide complex financial algorithms during the day and switch to social functionalities during the night. Another point to address as a future development regarding the 5GEE architecture is a fully study on power efficiency. This branch offers a quite large of ideas for future development. For example, the already proposed study is only focused on the service migration feature, this study might be extended as well as to the service orchestrator. By scenario definition, it has been assumed that, for instance, a smart city contains a large number of 5GEE nodes. A question might arise, what is the best 5GEE node towards to install the service in terms of power-efficient? In other words, what is the best location for the dedicated service? Those questions will be addressed by a full and comprehensive power consumption study.

7.2.2 Advanced Efficient Handoff of Services

It has already been stated that one of the problems of the EC networks is the node mobility. With our proposed architecture, mobile devices may have their services running at the edge of the network with very low latency and with a high rate of QoS and QoE. Service migration techniques are essential to ensure service continuity in contrast to the users' mobility. However, to ensure the best QoS and QoE during the handoff we need to explore efficient handoff methods. In this research, we proposed an efficient handoff method that leverages service characteristics. The method has proved to be very efficient by exploiting Docker technology. A future implementation may involve different software components, not only related to virtualization technologies, such as file systems. In this case, new techniques should be explored trying always to reduce the service downtime and guaranteeing a high rate of QoS and QoE.

7.2.3 Task offloading at the edge Future Directions

In this research, we proposed an efficient distributed file system among three layers such as mobile devices, edge devices, and the cloud that enables task offloading system. We believe that a meaningful extension is to make MEFS work for multiple collaborating users. The users collaborate within the same app. One reason for this proposed extension is that we think it's

easier to find applications that involve collaborating users, rather than just one user (one mobile and edge nodes). For example, for a CityParking app, the goal is to show the drivers on their mobiles (or car displays) the available parking spaces on different road segments around their destinations. In that scenario, cars looking for parking around a given destination, contact the edge node responsible for that destination and get a copy of the parking availability file. When the car parks, they write into this file to show that the parking spot was taken. When the cars leave a parking spot, they write into the file to show that the parking spot is available. So, we do have concurrent read/writes among many user devices and the edge and its realization would be a very interesting and promising future research.

Bibliography

[1]. Cisco says 50 billion connected “things” will be in use in 2020. Online: https://newsroom.cisco.com/feature-content?type=webcontent&articleId=1208342. Published: 2013, visited on October 31, 2019.

[2]. Here’s how the internet of things will explode by 2020. Online: https://www.insider.com/how-the-internet-of-things-market-will-grow-2014-10. Visited on October 31, 2019.

[3]. Fog Computing Defined. Online: https://www.slideshare.net/Angelo.Corsaro/fog-computing-defined. Published: 2017, visited on October 31, 2019.

[4]. F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. 2012. “Fog computing and its role in the internet of things”, In Proceedings of the first edition of the MCC workshop on Mobile cloud computing (MCC '12). ACM, New York, NY, USA, 13-16.

[5]. P. Porambage, J. Okwuibe, M. Liyanage, M. Ylianttila and T. Taleb, “Survey on Multi-Access Edge Computing for Internet of Things Realization,” in IEEE Communications Surveys & Tutorials, vol. 20, no. 4, pp. 2961-2991, Fourth quarter 2018.

[6]. ETSI’s Multi-access Edge Computing White Paper. Online: https://portal.etsi.org/Portals/0/TBpages/MEC/Docs/Mobile-edge_Computing_-_Introductory_Technical_White_Paper_V1%2018-09-14.pdf. Visited on October 31, 2019.

[7]. M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, “The case for VM-based cloudlets in mobile computing,” IEEE Pervasive Comput., vol. 8, no. 4, pp. 14–23, Oct./Dec. 2009.

[8]. Open Edge Computing. Online: http://openedgecomputing.org/. Visited on October 31, 2019.

[9]. H. A. Lagar-Cavilla, J. A. Whitney, A. M. Scannell, P. Patchin, S. M. Rumble, E. De Lara, M. Brudno, and M. Satyanarayanan, “Snowflock: rapid virtual machine cloning for cloud computing,” in Proceedings of the 4th ACM European conference on Computer systems. ACM, 2009.

[10]. D. Thai, C. Lee, D. Niyato, and P. Wang. (2013). “A survey of mobile cloud computing: Architecture, applications, and approaches,” Wireless Communications and Mobile Computing. 13. 10.1002/wcm.1203.

[11]. White Paper. Mobile Cloud Computing Solution Brief. AEPONA, 2010.

[12]. Christensen JH. “Using RESTful web-services and cloud computing to create next generation mobile applications,” In Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications (OOPSLA), 2009.

[13]. W. Tsai, X. Sun, J. Balasooriya, “Service-oriented cloud computing architecture,” In Proceedings of the 7th International Conference on Information Technology: New Generations (ITNG), 2010.

[14]. G. H. Forman and J. Zahorjan, “The challenges of mobile computing,” in Computer, vol. 27, no. 4, pp. 38-47, April 1994.

[15]. K. Kumar and Y. Lu, “Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?,” in Computer, vol. 43, no. 4, pp. 51-56, April 2010.

[16]. Amazon S3. Online: https://aws.amazon.com/it/s3/. Visited on October 31, 2019.

[17]. Google Photos. Online: https://www.google.com/intl/it/photos/about/. Visited on October 31, 2019.

[18]. Y. Mao, C. You, J. Zhang, K. Huang and K. B. Letaief, "A Survey on Mobile Edge Computing: The Communication Perspective," in IEEE Communications Surveys & Tutorials, vol. 19, no. 4, pp. 2322-2358, Fourthquarter 2017.
[19]. A. Huang, N. Nikaein, T. Stenbock, A. Ksentini and C. Bonnet, "Low latency MEC framework for SDN-based LTE/LTE-A networks," 2017 IEEE International Conference on Communications (ICC), Paris, 2017, pp. 1-6.
[20]. A. Manzalini, et al., Towards 5g software-defined ecosystems: technical challenges, business sustainability and policy issues, white paper (2014).
[21]. ETSI GS MEC 003 - Framework and Reference Architecture V1.1.1. Online: http://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/01.01.01_60/gs_MEC003v010101p.pdf. Visited on October 31, 2019.
[22]. Open Fog Consortium. Online: https://www.iiconsortium.org/index.htm. Visited on October 31, 2019.
[23]. M. Ketel, "Fog-Cloud Services for IoT," in Proceedings of the SouthEast Conference, ser. ACM SE '17. New York, NY, USA: ACM, 2017, pp. 262–264.
[24]. Cisco Fog Computing, "Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are". Online: https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf. Visited on October 31, 2019.
[25]. "Cisco IOx: Making Fog Real for IoT". Online: https://blogs.cisco.com/digital/cisco-iox-making-fog-real-for-iot. Visited on October 31, 2019.
[26]. L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," ACM SIGCOMM Computer Communication Review, vol. 44, no. 5, pp. 27–32, 2014.
[27]. IBM. "What is fog computing?" Online: https://www.ibm.com/blogs/cloud-computing/2014/08/25/fog-computing/. Visited on October 31, 2019.
[28]. S. Yi, C. Li, and Q. Li, "A survey of fog computing: concepts, applications and issues," in Proceedings of the 2015 workshop on mobile big data. ACM, 2015, pp. 37–42.
[29]. W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," IEEE Internet of Things Journal, vol. 3, no. 5, pp. 637–646, 2016.
[30]. B. Taylor, Y. Abe and A. Dey, "Virtual Machines for Remote Computing: Measuring the User Experience," Carnegie Mellon University, 2015.
[31]. P. Mekikis et al., "NFV-enabled Experimental Platform for 5G Tactile Internet Support in Industrial Environments," in IEEE Transactions on Industrial Informatics, 2019.
[32]. T. Taleb, A. Ksentini and P. A. Frangoudis, "Follow-Me Cloud: When Cloud Services Follow Mobile Users," in IEEE Transactions on Cloud Computing, vol. 7, no. 2, pp. 369-382, 1 April-June 2019.
[33]. V. Bahl, "Emergence of micro datacenter (cloudlets/edges) for mobile computing," Microsoft Devices Netw. Summit (Keynote Talk), Paris, France, Tech. Rep., 2015.
[34]. Micro-DataCenter from IBM. Online: https://www.ibm.com/blogs/research/2017/06/ibm-astrons-micro-datacenter-wins-hpc-vendor-innovation-award/. Visited on October 31, 2019.
[35]. Y. Jararweh, A. Doulat, O. AlQudah, E. Ahmed, M. Al-Ayyoub, and E. Benkhelifa, "The future of mobile cloud computing: Integrating cloudlets and mobile edge computing," in 2016 23rd International Conference on Telecommunications (ICT), May 2016, pp. 1–5.
[36]. R. P. Luijten and A. Doering, "The DOME embedded 64-bit microserver demonstrator," Proceedings of 2013 International Conference on IC Design & Technology (ICICDT), Pavia, 2013, pp. 203-206.

[37]. C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow and P. A. Polakos, “A Comprehensive Survey on Fog Computing: State-of-the-Art and Research Challenges,” in IEEE Communications Surveys & Tutorials, vol. 20, no. 1, pp. 416-464, Firstquarter 2018.
[38]. B. Omoniwa, R. Hussain, M. A. Javed, S. H. Bouk and S. A. Malik, “Fog/Edge Computing-Based IoT (FECIoT): Architecture, Applications, and Research Issues,” in IEEE Internet of Things Journal, vol. 6, no. 3, pp. 4118-4149, June 2019.
[39]. A. Corsaro and G. Baldoni, "fogØ5: Unifying the computing, networking and storage fabrics end-to-end," 2018 3rd Cloudification of the Internet of Things (CIoT), Paris, France, 2018, pp. 1-8.
[40]. ETSI OSM Community White Paper: Open Source MANO. Online: https://osm.etsi.org/images/OSM-Whitepaper-TechContent-ReleaseTWO-FINAL.pdf. Visited on October 31, 2019.
[41]. J. Oueis, E. C. Strinati and S. Barbarossa, “The Fog Balancing: Load Distribution for Small Cell Cloud Computing,” 2015 IEEE 81st Vehicular Technology Conference (VTC Spring), Glasgow, 2015, pp. 1-6.
[42]. M. Aazam and E. Huh, “Dynamic resource provisioning through Fog micro datacenter,” 2015 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops), St. Louis, MO, 2015, pp. 105-110.
[43]. S. Li, M. A. Maddah-Ali and A. S. Avestimehr, “Coding for Distributed Fog Computing,” in IEEE Communications Magazine, vol. 55, no. 4, pp. 34-40, April 2017.
[44]. K. Ha, Y. Abe, Z. Chen, W. Hu, B. Amos, P. Pillai, and M. Satyanarayanan, ‘‘Adaptive VM handoff across cloudlets,’’ Comput. Sci. Dept., Carnegie Mellon Univ., Ar-Rayyan, Qatar, Tech. Rep. CMU-CS-15-113, Jun. 2015.
[45]. S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, ‘‘Dynamic service placement for mobile micro-clouds with predicted future costs,’’ IEEE Trans. Parallel Distrib. Syst., vol. 28, no. 4, pp. 1002–1016, Apr. 2017.
[46]. W. Cerroni and F. Callegati, ‘‘Live migration of virtual network functions in cloud-based edge networks,’’ in Proc. IEEE Int. Conf. Commun. (ICC), Sydney, NSW, Australia, Jun. 2014, pp. 2963–2968.
[47]. R. M. Hines, U. Deshpande, and K. Gopalan, ‘‘Post-copy live migration of virtual machines,’’ ACM SIGOPS Operating Syst. Rev., vol. 43, no. 3, pp. 14–26, Jul. 2009.
[48]. K. Ha, Y. Abe, T. Eiszler, Z. Chen, W. Hu, B. Amos, R. Upadhyaya, P. Pillai, and M. Satyanarayanan, ‘‘You can teach elephants to dance: Agile vm handoff for edge computing,’’ in Proc. 2nd ACM/IEEE Symp. Edge Comput., Oct. 2017, p. 12.
[49]. A. Machen, S. Wang, K. K. Leung, B. J. Ko, and T. Salonidis, ‘‘Live service migration in mobile edge clouds,’’ IEEE Wireless Commun., vol. 25, no. 1, pp. 140–147, Feb. 2018.
[50]. D. Lezcano. LXC—Linux Containers. Online: https://github.com/lxc/lxc. Visited on October 31, 2019.
[51]. Kernel Virtual Machine (KVM). Online: https://www.linux-kvm.org/page/Main_Page. Visited on October 31, 2019.
[52]. Checkpoint/Restore in Userspace, or CRIU. Online: https://www.criu.org/Main_Page. Visited on October 31, 2019.
[53]. Open Source Container-Based Virtualization for Linux OpenVZ. Online: https://openvz.org/. Visited on October 31, 2019.
[54]. Docker. Online: https://www. docker.com/. Visited on October 31, 2019.
[55]. P. Haul. Container Live Migration. Online: https://criu.org/P.Haul. Visited on October 31, 2019.

[56]. Virtuozzo Storage. Online: https://openvz.org/Virtuozzo_Storage. Visited on October 31, 2019.
[57]. P. Emelyanov. Live Migration Using CRIU. Online: https://github.com/xemul/p.haul. Visited on October 31, 2019.
[58]. S. Nadgowda, S. Suneja, N. Bila, and C. Isci, ‘‘Voyager: Complete Con- tainer State Migration,’’ in Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS), Atlanta, GA, USA, Jun. 2017, pp. 2137–2142.
[59]. B. G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, “CloneCloud: Elastic execution between mobile device and cloud,” in EuroSys 2011, 2011, pp. 301–314.
[60]. E. Cuervo, A. Balasubramanian, D. ki Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, “MAUI: making smartphones last longer with code offload,” in MobiSys ’10, 2010, pp. 49–62.
[61]. S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, “ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading,” in Infocom ’12, 2012, pp. 945–953.
[62]. M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen, “COMET: Code offload by migrating execution transparently,” in OSDI ’12, 2012, pp. 93–106.
[63]. C. You and K. Huang, “Multiuser Resource Allocation for Mobile- Edge Computation Offloading,” 2016 IEEE Global Communications Conference (GLOBECOM), 2016, pp. 1-6.
[64]. G. Orsini, D. Bade and W. Lamersdorf, “Computing at the Mobile Edge: Designing Elastic Android Applications for Computation Offloading,” 2015 8th IFIP Wireless and Mobile Networking Conference (WMNC), Munich, 2015, pp. 112-119.
[65]. Service Location Protocol Project. Online: http://srvloc.sourceforge.net/. Visited on October 31, 2019.
[66]. UDDI Standard. Online: http://uddi.xml.org/. Visited on October 31, 2019.
[67]. UPnP Forums. Online: https://openconnectivity.org/. Visited on October 31, 2019.
[68]. Jini protocol. Online: https://river.apache.org/. Visited on October 31, 2019.
[69]. Bonjour protocol. Online: https://developer.apple.com/documentation/foundation/bonjour. Visited on October 31, 2019.
[70]. M. Aazam and E. Huh, “Fog Computing and Smart Gateway Based Communication for Cloud of Things,” 2014 International Conference on Future Internet of Things and Cloud, Barcelona, 2014, pp. 464-470.
[71]. S. Cirani, G. Ferrari, N. Iotti and M. Picone, “The IoT hub: a fog node for seamless management of heterogeneous connected smart objects,” 2015 12th Annual IEEE International Conference on Sensing, Communication, and Networking - Workshops (SECON Workshops), Seattle, WA, 2015, pp. 1-6.
[72]. K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. Gander, P. Gibbons, and O. Mutlu, “Gaia: Geo-distributed machine learning approaching lan speeds,” 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17), pp. 620–647, 2017.
[73]. T. Yang, Y. Hu, M. C. Gursoy, A. Schmeink and R. Mathar, “Deep Reinforcement Learning based Resource Allocation in Low Latency Edge Computing Networks,” 2018 15th International Symposium on Wireless Communication Systems (ISWCS), Lisbon, 2018, pp. 1-5.
[74]. P. S. Chandakkar, Y. Li, P. L. K. Ding and B. Li, “Strategies for Re- Training a Pruned Neural Network in an Edge Computing Paradigm,” 2017 IEEE International Conference on Edge Computing (EDGE), Honolulu, HI, 2017, pp. 244-247.

[75]. S. Raileanu, T. Borangiu, O. Morariu and I. Iacob, “Edge Computing in Indust00rial IoT Framework for Cloud-based Manufacturing Control,” 2018 22nd International Conference on System Theory, Control and Computing (ICSTCC), Sinaia, 2018, pp. 261-266.
[76]. B. Chen, J. Wan, A. Celesti, D. Li, H. Abbas and Q. Zhang, “Edge Computing in IoT-Based Manufacturing,” in IEEE Communications Magazine, vol. 56, no. 9, pp. 103-109, Sept. 2018.
[77]. Open Baton. Online: https://openbaton.github.io/. Visited on October 31, 2019.
[78]. G. Cardone, A. Corradi, L. Foschini, R. Ianniello, “ParticipAct: A Large-Scale Crowdsensing Platform”, Proc. of IEEE Transactions on Emerging Topics in Computing vol. 4, no. 1, Jan.-March 2016, pp. 21– 32.
[79]. Ruchika, “Evaluation of Docker for IoT Application”, Proc. of International Journal on Recent and Innovation Trends in Computing and Communication, vol. 4 no. 6, June 2016, pp. 624–62.
[80]. “YouTube accounts for 35% of worldwide mobile internet traffic”. Online: https://www.fiercevideo.com/video/youtube-accounts-for-35-worldwide-mobile-internet-traffic-sandvine-says. Visited on October 31, 2019.
[81]. C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, and A. Warfield, ‘‘Live migration of virtual machines,’’ in Proc. 2nd Conf. Symp. Netw. Syst. Design Implement., vol. 2, 2005, pp. 273–286.
[82]. W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, ‘‘An updated performance comparison of virtual machines and Linux containers,’’ in Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS), Philadelphia, PA, USA, Mar. 2015, pp. 171–172.
[83]. S. Wang, J. Xu, N. Zhang and Y. Liu, “A Survey on Service Migration in Mobile Edge Computing,” in IEEE Access, vol. 6, pp. 23511-23528, 2018.
[84]. Y. C. Tay, K. Gaurav and P. Karkun, “A Performance Comparison of Containers and Virtual Machines in Workload Migration Context,” 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW), Atlanta, GA, 2017, pp. 61-66.
[85]. R. Boucher. Live Migration Using CRIU. Online: https://github.com/boucher/p.haul. Visited on October 31, 2019.
[86]. M. H. Rahman, F. B. Al Abid, M. N. Zaman, and M. N. Akhtar, ‘‘Optimizing and enhancing performance of database engine using data clustering technique,’’ in Proc. Int. Conf. Adv. Electr. Eng. (ICAEE), Dhaka, Bangladesh, Dec. 2015, pp. 198–201.
[87]. Y. Ge, Z. Zheng, B. Yan, J. Yang, Y. Yang, and H. Meng, ‘‘An RSSI-based localization method with outlier suppress for wireless sensor networks,’’ in Proc. 2nd IEEE Int. Conf. Comput. Commun. (ICCC), Oct. 2016, pp. 2235–2239.
[88]. J. Kikuchi, C. Wu, Y. Ji, and T. Murase, ‘‘Mobile edge computing based VM migration for QoS improvement,’’ in Proc. IEEE 6th Global Conf. Consum. Electron. (GCCE), Nagoya, Japan, Oct. 2017, pp. 1–5.
[89]. Docker Compose. Online: https://docs.docker.com/compose/. Visited on October 31, 2019.
[90]. Docker Volume. Online: https://docs.docker.com/storage/volumes/. Visited on October 31, 2019.
[91]. R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, “CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,’’ Softw., Pract. Exper., vol. 41, no. 1, pp. 23–50, Jan. 2011.

[92]. C. Sonmez, A. Ozgovde, and C. Ersoy, ‘‘EdgeCloudSim: An environment for performance evaluation of edge computing systems,’’ Trans. Emerg. Telecommun. Technol., vol. 29, no. 11, Nov. 2018, Art. no. e3493.
[93]. P. Bellavista, S. Chessa, L. Foschini, L. Gioia, M. Girolami, “Human- Enabled Edge Computing: Exploiting the Crowd as a Dynamic Extension of Mobile Edge Computing”, IEEE Communications Magazine, Vol. 56, No. 1, 2018.
[94]. C. Borcea, X. Ding, N. Gehani, R. Curtmola, M. A. Khan, and H. Debnath, “Avatar: Mobile distributed computing in the cloud,” in MobileCloud ’15, 2015.
[95]. I. Zhang, A. Szekeres, D. Van Aken, I. Ackerman, S. D. Gribble, A. Krishnamurthy, and H. M. Levy, “Customizable and extensible deployment for mobile/cloud applications,” in OSDI’14, 2014, pp.97– 112.
[96]. Dropbox. Online: https://www.dropbox.com/. Visited on October 31, 2019.
[97]. B. Pawlowski, S. Shepler, C. Beame, B. Callaghan, M. Eisler, D. Noveck, D. Robinson, and R. Thurlow, “The NFS version 4 protocol,” in SANE 2000, 2000.
[98]. A. Zanni et al., “Automated Offloading of Android Applications for Computation/Energy-usage Optimizations,” in Infocom Demo Papers, 2017.
[99]. N. R. Paiker, J. Shan, C. Borcea, N. Gehani, R. Curtmola and X. Ding, “Design and Implementation of an Overlay File System for Cloud- Assisted Mobile Apps,” in IEEE Transactions on Cloud Computing, 2017.
[100]. AspectJ. Online: https://www.eclipse.org/aspectj/. Visited on October 31, 2019.
[101]. Kryonet. Online: https://github.com/EsotericSoftware/kryonet. Visited on October 31, 2019.
[102]. ETSI GR MEC 018. Online: https://www.etsi.org/deliver/etsi_gr/MEC/001_099/018/01.01.01_60/gr_MEC018v010101p.pdf. Visited on October 31, 2019.
[103]. M. Rosenblum and J. K. Ousterhout. 1992. “The design and implementation of a log-structured file system,” ACM Trans. Comput. Syst. 10, 1 (February 1992), 26-52.
[104]. “Quality of Service Design Overview”. Online: http://www.ciscopress.com/articles/article.asp?p=357102. Visited on October 31, 2019.
[105]. S. J. Thorpe and M. Fabre-Thorpe, “Seeking Categories in the Brain,” American Association for the Advancement of Science, vol. 291, no. 5502, pp. 260-263, January 2001.
[106]. G. Ananthanarayanan et al., “Real-Time Video Analytics: The Killer App for Edge Computing,” in Computer, vol. 50, no. 10, pp. 58-67, 2017.
[107]. Retrofit. Online: https://square.github.io/retrofit/. Visited on October 31, 2019.
[108]. JmDNS. Online: https://github.com/jmdns/jmdns. Visited on October 31, 2019.
[109]. A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari and M. Ayyash, “Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications,” in IEEE Communications Surveys & Tutorials, vol. 17, no. 4, pp. 2347-2376, Fourthquarter 2015.
[110]. S. Cirani et al., “A Scalable and Self-Configuring Architecture for Service Discovery in the Internet of Things,” in IEEE Internet of Things Journal, vol. 1, no. 5, pp. 508-521, Oct. 2014.
[111]. MQTT. Online: http://mqtt.org/. Visited on October 31, 2019.
[112]. Z. Shelby, K. Hartke, and C. Bormann, “The Constrained Application Protocol (CoAP),” RFC 7252 (Proposed Standard), Internet Engineering Task Force, Jun. 2014. Online: http://www.ietf.org/rfc/rfc7252.txt. Visited on October 31, 2019.
[113]. S. Shalev-Shwartz and S. Ben-David, Understanding machine learning: From theory to algorithms. Cambridge university press, 2014.

[114]. A. Ahmed, J. Guo, F. Ali, F. Deeba and A. Ahmed, “LBPH based improved face recognition at low resolution,” 2018 International Conference on Artificial Intelligence and Big Data (ICAIBD), Chengdu, 2018, pp. 144-147.
[115]. Kaggle dataset. Online: https://www.kaggle.com/gasgallo/faces-data. Visited on October 31, 2019.
[116]. APS Failure at Scania Trucks Data Set. Online: https://archive.ics.uci.edu/ml/datasets/APS+Failure+at+Scania+Trucks. Visited on October 31, 2019.