Pós-Graduação em Ciência da Computação

Warley Muricy Valente Junior

A Context-sensitive offloading system using machine-learning classification algorithms with seamless mobility support

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
http://cin.ufpe.br/~posgraduacao

RECIFE

2018

Warley Muricy Valente Junior

A Context-sensitive offloading system using machine-learning classification algorithms with seamless mobility support

A Ph.D. Thesis presented to the Informatics Center of Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Philosophy Doctor in Computer Science.

Advisor: Kelvin Lopes Dias

RECIFE
2018

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

V154c Valente Junior, Warley Muricy
A context-sensitive offloading system using machine-learning classification algorithms with seamless mobility support / Warley Muricy Valente Junior. – 2018.
120 f.: il., fig., tab.

Orientador: Kelvin Lopes Dias.
Tese (Doutorado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2018.
Inclui referências.

1. Redes de computadores. 2. Computação em nuvem. I. Dias, Kelvin Lopes (orientador). II. Título.

004.6 CDD (23. ed.) UFPE- MEI 2018-041

Warley Muricy Valente Junior

A Context-sensitive Offloading System Using Machine-learning Classification Algorithms with Seamless Mobility Support

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação.

Aprovado em: 06/02/2018.

_____

Orientador: Prof. Dr. Kelvin Lopes Dias

BANCA EXAMINADORA

_____

Profa. Dra. Ana Carolina Brandão Salgado
Centro de Informática / UFPE

_____

Prof. Dr. Carlos André Guimarães Ferraz
Centro de Informática / UFPE

_____

Prof. Dr. George Darmiton da Cunha Cavalcanti
Centro de Informática / UFPE

_____

Prof. Dr. Carlos Alberto Kamienski
Centro de Matemática, Computação e Cognição / UFABC

_____

Prof. Dr. Jó Ueyama
Instituto de Ciências Matemática e de Computação / USP

Acknowledgments

This thesis is the result of a four year journey in Recife-PE. I found many new colleagues and friends. I would like to thank them all for their influence on the outcome of my PhD work, and ultimately on this thesis. I will mention some names briefly, as space dictates, and, if your name has been left out, please accept my non-nominative thanks. I would like to start by thanking the Federal University of Pernambuco for showing me, as a researcher, that does not matter where we are, but who we want to become. I would like express my gratitude to professor Kelvin Dias, my adviser throughout the four years of the PhD. Kelvin taught me how to do research always targeting the perfection. He gave me all support not only as an adviser but as a friend. Thank you for accepting me in your IANG research group. The IANG group represented an essential collaboration environment to reach my goals. I would like to mention my co-authors — Adriano, Eduardo, Bruno, and Albertinin — for our mutual support and our continuous collaboration. I want to thank my colleagues — Atrícia, Francisco, Andson, and Edivaldo — for providing a stimulating research environment. A special thank for my Scientific Initiation undergraduate students — Eduardo and Albertinin — for helping me with the hard work. Foremost, I would like to thank my wife, mom, dad and the rest of my family, for being close to me even when I was far away and for their continuous support.

"Continue fazendo o certo, mesmo que tudo esteja errado. A obediência a Deus sempre traz recompensas."

(Pr. Antonio Junior)

Abstract

Mobile Cloud Computing (MCC) enables resource-constrained smartphones to run computation-intensive applications through code/data offloading to resourceful servers. Nevertheless, this technique can be disadvantageous if the offloading decision does not consider contextual information. Another MCC challenge is related to the change of access point during an on-going offloading process, since it impacts on or is impacted by resource scarcity, finite energy, and low connectivity in a wireless environment. This PhD research has developed a context-sensitive offloading system that takes advantage of the machine-learning reasoning techniques and robust profilers to provide offloading decisions with the best levels of accuracy as compared to state-of-the-art solutions. In addition, this work proposes a way to support seamless offloading operations during user mobility through the software-defined networking (SDN) paradigm and remote caching technique to speed up the offloading response time. Firstly, in order to address the offloading decision issue, the approach evaluates the main classifiers under a database comprised of cloud, smartphone, application, and networks parameters. Secondly, it transforms raw context parameters to high-level context information at runtime and evaluates the proposed system under real scenarios, where context information changes from one experiment to another. Under these conditions, system makes correct decisions as well as ensuring performance gains and energy efficiency, achieving decisions with 95% of accuracy. With regards SDN-based mobility support, the results have shown that it is energy efficient, especially considering the low-cost smartphone category, while remote caching proved to be an attractive alternative for reducing the offloading response time.

Key-words: Mobile cloud computing. Context-sensitive. Machine-learning. Classification algorithms. Software-defined networking. Mobility management

Resumo

A computação em nuvem móvel (MCC) permite que *smartphones* com recursos limitados executem aplicações intensivas de computação através do *offloading* de código/dados para servidores potentes. No entanto, esta técnica pode ser desvantajosa se a decisão de *offloading* não considera informações contextuais. Outro desafio da MCC está relacionado à mudança de ponto de acesso durante um processo de *offloading* contínuo, uma vez que impacta ou é impactado pela escassez de recursos, energia finita e baixa conectividade em um ambiente sem fio. Esta pesquisa de doutorado desenvolveu um sistema de *offloading* sensível ao contexto que tira proveito das técnicas de raciocínio de aprendizagem de máquina e perfiladores robustos para prover decisões de *offloading* com os melhores níveis de acurácia em comparação com soluções do estado da arte. Além disso, este trabalho propõe uma maneira de suportar operações de *offloading* contínuas durante a mobilidade do usuário através do paradigma de redes definidas por software (SDN) e técnica de cache remoto para acelerar o tempo de resposta do *offloading*. Primeiramente, para resolver o problema da decisão de *offloading*, a abordagem avalia os principais classificadores sob uma base de dados composta de parâmetros relacionados a nuvem, smartphone, aplicativos e rede. Em segundo lugar, ela transforma parâmetros de contexto bruto em informações de contexto de alto nível em tempo de execução e avalia o sistema proposto em cenários reais, aonde as informações de contexto mudam de um experimento para outro. Nessas condições, o sistema toma decisões corretas, bem como garante ganhos de desempenho e eficiência energética, alcançando decisões com 95% de acurácia. Com relação ao suporte à mobilidade baseado em SDN, os resultados mostram que o sistema é eficiente em termos energéticos, especialmente considerando a categoria de *smartphones* de baixo custo, enquanto o cache remoto provou ser uma alternativa atrativa para reduzir o tempo de resposta de *offloading*.

Palavras-chaves: Computação em nuvem móvel. Sensibilidade ao contexto. Aprendizagem de máquina. Algoritmos de classificação. Redes definidas por software. Gerenciamento de mobilidade

List of Figures

List of Tables

List of abbreviations and acronyms

ACC Accuracy.

ANOVA Analysis of Variance.

AP Access Point.

ARC AnyRun Computing.

ASICs Application-Specific Integrated Circuits.

BCE Binding Cache Entry.

CD Critical Difference.

CSOS Context-Sensitive Offloading System.

D2D Device-to-Device.

DMM Distributed Mobility Management.

DNF Disjunctive Normal Form.

FME Follow Me Edge.

FMIPv6 Fast Mobile IPv6.

FNR False Negative Rate.

FPR False Positive Rate.

FPS Frame Per Second.

GPS Global Positioning System.

IBL Instance-Based Learning.

ICMP Internet Control Message Protocol.

IP Internet Protocol.

K-NN K-Nearest Neighbor.

LLDP Link Layer Discovery Protocol.

LTE Long Term Evolution.

MAC Media Access Control.

MCB Multiple Comparison with Best.

MCS Multiple Classifier Systems.

mCSOS Mobile Context-Sensitive Offloading System.

MD Mobile Device.

MD-SAL Model-Driven Service Abstraction Layer.

MEC Mobile Edge Computing.

M-MIP Multi-homed Mobile IP.

MCC Mobile Cloud Computing.

ML Machine Learning.

MSN Mobile Social Networking.

mW Miliwatts.

NFV Network Functions Virtualization.

OCs Offloading Candidates.

ODL OpenDaylight.

OFLMA OpenFlow Local Mobility Anchor.

OFMAG OpenFlow Mobility Access Gateway.

ONF Open Networking Foundation.

OS Operating System.

OSGI Open Services Gateway Initiative.

PBA Proxy Binding Ack.

PBU Proxy Binding Update.

PMIPv6 Proxy Mobile IPv6.

PoA Point-of-Attachment.

PPMF Probability Predictive Mode Failure.

QoS Quality of Service.

RAN Radio Access Network.

RIPPER Repeated Incremental Pruning to Produce Error Reduction.

RPC Remote Procedure Calls.

RSSI Received Signal Strength Indication.

RSUs Roadside Units.

RTT Round Trip Time.

SCC Small Cell Cloud.

SCeNB Small Cells.

SDN Software-Defined Networking.

SIP Session Initial Protocol.

SLA Service Level Agreement.

SML Supervised Machine Learning.

SMS Systematic Mapping Study.

SVM Support Vector Machine.

TET Total Execution Time.

TOPSIS Technique for Order of Preference by Similarity to Ideal Solution.

UDP User Datagram Protocol.

V2I Vehicle-to-Infrastructure.

V2V Vehicle-to-Vehicle.

VANETs Vehicular Ad-Hoc Networks.

vCPU Virtual Central Processing Unit.

VM Virtual Machine.

WAN Wide Area Network.

WCDMA Wide-Band Code-Division Multiple Access.

WiFi Wireless Fidelity.

WLAN Wireless Local Area Network.

WWAN Wireless Wide Area Network.

Contents

1 INTRODUCTION

This chapter introduces the PhD research with regard to the context and motivation, including the use of the offloading technique in Mobile Cloud Computing, followed by main challenges involved in this technique. Next, we describe the key issues that are addressed by this work and list specific objectives that make up the purpose of this research. Finally, the published papers and thesis structure are presented.

1.1 Context and Motivation

Cloud computing has taken the world by storm. In this category of utility computing, a collection of computing resources (e.g., network, servers, storage) are pooled to serve multiple consumers, using a multi-tenant model. These resources are available over a network, and accessed through standard mechanisms (ROMAN; LOPEZ; MAMBO, 2016). The cloud computing paradigm provides a variety of deployment models and service models, from public clouds (organizations provide cloud computing services to any customer) to private clouds (organizations deploy their own private cloud computing platform), and from Infrastructure as a Service models (IaaS, where fundamental computing resources are offered as a capability) to Software as a Service models (SaaS, where applications are offered as a capability), among other things. The benefits of cloud computing – minimal management effort, convenience, rapid elasticity, pay per use, ubiquity – have given birth to a multi-billion industry that is growing worldwide (ROMAN; LOPEZ; MAMBO, 2016).

Meanwhile, the latest developments in mobile device technologies and demand for sophisticated mobile applications have changed user trends towards computing. Mobile devices are expected to provide complicated and ubiquitous services instead of simple telephony and computation. Applications such as m-commerce, mobile telemedicine, multiplayer mobile gamming, natural language processing, augmented reality service and interactive service make mobile devices an important part of everyday life (GANI et al., 2014; SATYANARAYANAN et al., 2015). Unfortunately, the advances in smartphone hardware and battery life have been slow to respond to the computational demands of applications evolved over the years. Therefore, many applications are still unsuitable for smartphones due to constraints, such as low processing power, limited memory, unpredictable network connectivity, and limited battery life (KHAN et al., 2014).

In general, to make the smartphones energy efficient and computationally capable, major hardware and software level changes are needed, which requires the developers and manufacturers to work together (BARBA; MACINTYRE; MYNATT, 2012; MASCOLO, 2010). Due to size-constraints, hardware level changes alone may not enable smartphones to achieve true unlimited computational power. Therefore, software-level changes are more

effective, where computation is performed on remote resources with partial support of a smartphone's hardware (KEMP et al., 2009).

In order to provide mobile users with a similar experience to using powerful desktop computers, an adequate infrastructure to enable the storage and processing of resource-intensive mobile applications on remote resource servers has been developed, termed Mobile Cloud Computing (MCC) (GAO; GRUHN; ROUSSOS, 2013). MCC is defined by the availability of Cloud Computing services in a mobile ecosystem. It extends computational resources from mobile devices to cloud providers in order to increase service availability, speed and reliability while saving the device's energy (FERNANDO; LOKE; RAHAYU, 2013). Offloading operations may ensure these benefits by enabling code and data migration from a mobile device to resource-rich cloud servers (ENZAI; TANG, 2014).

According to (FLORES et al., 2017), computational offloading is the opportunistic process that relies on external infrastructure to execute a computational task outsourced by a low-power device. Moving a computational task from one device to another is not a trivial endeavor. Network bandwidth, received signal strength, input data size, and cloud capabilities, amongst others, play a critical role in deciding whether or not to offload a task, since these parameters can change suddenly over time. Therefore, the effectiveness of an offloading operation is determined by its ability to infer where the execution of code/data (local or remote) will result in less computational effort for the mobile device. The evaluation of the code/data requires consideration of different perspectives, for instance, *what* code to offload (e.g., method name); *when* to offload (e.g., round-trip times thresholds); *where* to offload (e.g., cloudlet server, edge computing or remote cloud); *how* to offload (e.g., split code into n processes, methods or components); and so on (FLORES et al., 2015).

This problem introduces the necessity of computational offloading systems that can adapt themselves based on the information regarding the resources that are being provided, to decide where and when to perform offloading, as well as to infer the contextual information of mobile devices. In other words, it refers to a system's awareness of its surrounding environment, how it is able to monitor, collect, select, process, and share an entity's context information and how this is involved in decision-making and the execution of computational offloading. Context information is any kind of information that characterizes an entity in a specific domain; an entity in this regard can be a person, mobile device, application, remote cloud, or network element (BETTINI et al., 2010).

On the other hand, the mobility feature is a critical issue concerning MCC environments once it impacts on or is impacted by resource scarcity, finite energy, and low connectivity in a wireless environment (LI et al., 2015). These problems have emerged because during the execution of an application in the cloud, a mobile user may move from one network to another, where unmanaged mobility in a wireless environment causes communication disruption when the mobile device moves across two different communication

coverage areas (ZEKRI; JOUABER; ZEGHLACHE, 2012). Moving between different attachment points and concomitantly performing offloading continuously is an unfilled gap in the MCC literature. Hence, mobile applications that demand Quality of Service (QoS) requirements are prone to handovers (i.e., the change of access point serving mobile device during ongoing sessions), remote cloud service unavailability, insufficient network bandwidth, and long-latency periods (JUNIOR et al., 2017). So, due to the nature of distributed computing, MCC necessitates an uninterrupted and unobtrusive communication medium between the remote cloud and the mobile device, as well as cooperation between clouds. This has stimulated a new concept of "seamless" connectivity and service, i.e., it refers to the seamless service provisioning across different wireless access networks and optimal service delivery via the most appropriate cloud resource (GANI et al., 2014).

Traditional handover management techniques based solely on network and device contexts (e.g., bandwidth availability and residual battery) do not fulfill the requirements of MCC applications, since these applications are part of different domains (e.g., Internet of Things, mobile healthcare, mobile crowdsourcing), which involve concerns related to energy, runtime, costs, scalability, and flexibility. For instance, in mobile e-health environment, an energy-poor smartphone can offload human vital signs from wearable sensors to be processed continuously while the user is moving in a car. However, simply choosing another access point on the street may not suffice to provide seamless execution of cloud application. This is because the service provided by co-located cloud resource can be exhausted or yet, in case of service migration from one cloud to another, the delay should be prohibitive to re-instantiate the service in another location. In addition, the surrounding environment is also important, such as vehicular traffic at that moment, which could be taken into account in handover decision. Thus, mobility management mechanisms and techniques suitable for MCC are two very important elements to maintain seamless connectivity and services.

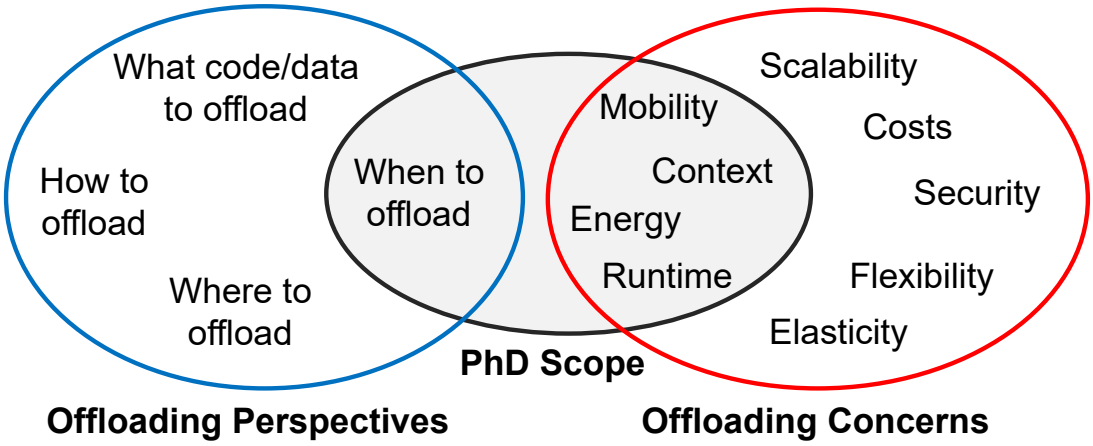


Figure 1 – PhD research scope.

As illustrated in Figure 1, this PhD research has focused mainly on the "When"

perspective due to its challenging and important features. The offloading system should decide whether it is counterproductive or not to execute the tasks remotely. Thus, it does not make sense, for example, to offload one application through a very low quality wireless connection or when the application does not need much resources. Several parameters (such as connectivity and remote cloud capacity) should be considered when offloading, otherwise the application may loose performance.

In terms of offloading concerns, this PhD research has focused on context-awareness, mobility, runtime, and energy. The last two have always been users' requirements and consequently a mobile industry interest. The runtime or total execution time is the most common metric to assess mobile applications performance (SILVA et al., 2016). Great importance is given to this factor because it influences directly on the user experience. For example, if one automobile in a cohort hits a pothole or swerves to avoid a fallen tree branch, the coordinates of the hazard can be rapidly offloaded within the cell tower cloudlet and then used for avoidance by the other automobiles in that cohort (SATYANARAYANAN et al., 2015).

On the other hand, it has long been recognized that mobile hardware is necessarily resource-poor. Since the smartphone has limited resources (i.e., processor, battery, wireless interface, etc.), is infeasible to finish some computational-intensive tasks (e.g., real-time image processing for video games, augmented reality, and location-based service). Computing speeds of these mobile devices, however, will not grow at the same pace as servers' performance. This is due to several constraints, including: hardware constraints, weight and heat dissipation limit the computational resources of the device; user preferences, as users want devices that are smaller and thinner and yet with more computational capability; low connectivity, end-to-end latency increases the remote processing response time, and consequently also increase the per-operation energy consumption on the mobile device (CHEN et al., 2015); power consumption, insofar the current battery technology constrains the clock speed of processors, doubling the clock speed approximately octuples the power consumption. It becomes difficult to offer long battery lifetimes with high clock speeds (SILVA et al., 2016). Therefore, context, mobility, runtime, and energy will continue being an MCC concern in a long term, motivating further research of these topics.

1.2 Problem Statement

Consider the scenario where a mobile user is walking in an urban area and needs to run heavy tasks from a mobile application. Offloading these tasks to a remote cloud may improve the application performance. One problem is that the latency in the communication, CPU (smartphone) and vCPU (cloud) usage, and input data size can change abruptly. Thus, the opportunistic moments to offload a task in a remote cloud system are sporadic. In addition, remote cloud offloading is sensitive to the multiple parameters

of the system (the context of the device, the application, and the network), which means that it is challenging to pinpoint an opportunistic moment to offload. An other problem is related to mobility, because during offloading operations the mobile user may move from one network to another, which consequently causes communication disruptions and application crashes.

In this context, two questions arise:

- When to offload tasks considering context information in an MCC environment?
- How to ensure the smoothing of offloading operations while the user moves between Point-of-Attachment (PoA)s?

1.3 Objectives

The main objective of this research is to design and develop new strategies in MCC that can improve the performance and energy saving in mobile devices, as well as ensure seamless offloading in mobility scenarios.

Among the specific goals of the research, we can list:

1. Propose a object-oriented middleware designed for MCC.
2. Propose a SDN-based mobility management application for locating a mobile device, initializing handover, selecting a new network, and registering the mobile device with the new network.
3. Create and implement an algorithm for context-sensitive offloading decision, as well as a set of machine-learning classifiers for assisting the decision algorithm.
4. Building a testbed environment which includes dynamic and mobile scenarios.

1.4 Publications

Following, a list with the published papers related to this research is presented.
As main author:

- Warley Junior, Adriano França, Kelvin Dias. Avaliação de Desempenho da Técnica de Offloading Computacional em Nuvens Móveis. WPerformance. 2015.
- Warley Junior, Adriano França, Kelvin Dias. Supporting mobility-aware computational offloading in mobile cloud environment. Journal of Network and Computer Applications. 2017.

- Warley Junior, Bruno Roberto, Kelvin Dias. A systematic mapping study on mobility mechanisms for cloud service provisioning in mobile cloud ecosystem. Computers and Electrical Engineering. 2018.

  As co-author:

- Bruno Roberto, Atrícia Sabino, Warley Junior, Eduardo Oliveira, Francisco Junior, Kelvin Dias. Performance Evaluation of Cryptography on Middleware-Based Computational Offloading. VII Brazilian Symposium on Computing Systems Engineering (SBESC). 2017.

1.5 Thesis Structure

This thesis is structured as follows. Chapter 2 clarifies some relevant background themes that the reader should know for properly understanding this document. Chapter 3 discusses noteworthy works found in literature that have some topics in common to those addressed in this thesis. Chapter 4 details the core contribution of this thesis. The Chapter describes an approach that evaluate machine-learning classifiers and presents a context-sensitive offloading system based on classification algorithms and robust profilers. Chapter 5 presents an extension to the context-sensitive offloading system by applying mobility-specific features, such as mobility management and remote caching. Finally, Chapter 6 traces some conclusions and future work.

2 BACKGROUND

This chapter discusses the basic concepts of mobile cloud, offloading mechanisms, software-defined networking, context-aware, and machine learning. The background presented here shall provide the necessary knowledge for a clear comprehension of the chapters ahead, including the aspects surrounding the proposed methodology and subsequent case studies.

2.1 Mobile Cloud Computing

There are several existing definitions of Mobile Cloud Computing, and different research alludes to different concepts of the 'mobile cloud'. To avoid the exhaustion of these definitions, we can highlight the work referenced in (KHAN et al., 2014), which defines MCC as "*an integration of cloud computing technology with mobile computing in order to make mobile devices resource-full in terms of computational power, memory, storage, energy, and context awareness*".

According to (GAO; GRUHN; ROUSSOS, 2013), MCC often involves three foundations, namely cloud computing, mobile computing, and networking. Thus defined MCC as "*an emergent mobile cloud paradigm which leverage mobile computing, networking, and cloud computing to study mobile service models, develop mobile cloud infrastructures, platforms, and service applications for mobile clients*".

Therefore, the communication between mobile device and remote server plays an important role in the successful execution of cloud-based applications. For better understanding and maturation of MCC concepts, we will describe below, the four MCC models widely studied in the state-of-the-art.

2.1.1 Mobile Cloud Models

The MCC paradigm can be categorized into four different models: public cloud, small cell cloud, cloudlet server, and ad-hoc cloudlet. They are described in detail below.

2.1.1.1 Public Cloud

The mobile device that uses the public cloud to augment resources acts as a thin client while connecting through any of the wireless technologies to a remote cloud server. A public cloud is formed from computational resources that are located in centralized data centers and maintained by cloud service providers, as shown in Figure 2. Therefore, mobile users can access public cloud services in two ways: base stations or access point.

Access point are equipments that are used to establish connectivity between mobile devices and the Internet through Wireless Local Area Network (WLAN). They use the Wireless Fidelity (WiFi) which has recently made use of 802.11n and 802.11ac standards. On the other hand, base stations are fixed locations for telecommunications networks used to establish connection between the smartphone and the phone provider through the Wireless Wireless Wide Area Network (WWAN). These make use of the Wide-Band Code-Division Multiple Access (WCDMA) and Long Term Evolution (LTE) standards also called 3G and 4G, respectively.

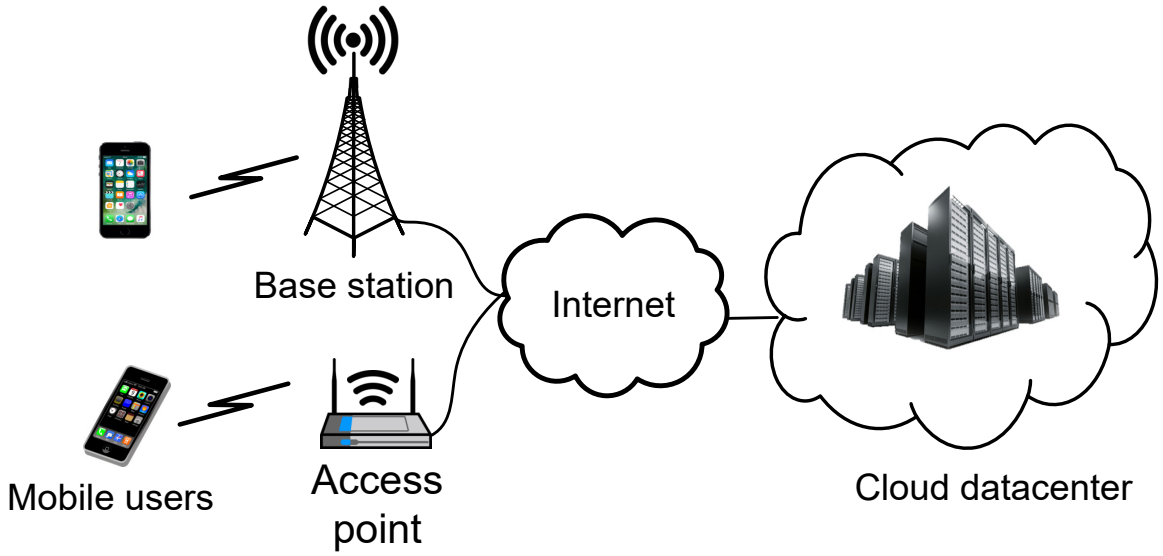


Figure 2 – Public cloud.

The application that leverages on the cloud is required to exchange the data between two ends, the mobile device and the cloud, using the wireless Internet resources, such as wireless backbone networks. The communication across the Internet is affected because of the high WWAN latency, interoperability among the underlying different networking technologies, non-guaranteed bandwidth reservation, bursty losses and excessive delay due to congestion, and non-deterministic traffic load along the path. The increase in the mobile communication traffic in the wireless backbone network may cause congestion that increases the data transfer time from a mobile device to the cloud and vice versa. The use of congested backbone links and non-guaranteed access to these backbone networks increase the probability of disruption during the access of services provided by the public cloud.

2.1.1.2 Small Cell Cloud

In common cellular networks, the closest place for deployment of computing resources is a base station. With increasing density of deployed cells, the small cells are seen as a mean to provide cloud computing services to users in proximity (see Figure 3).

This concept is known as Small Cell Cloud (SCC) (BECVAR; PLACHY; MACH, 2014). In the SCC, the Small Cells (SCeNB) are empowered by additional computing and storage resources in order to enable efficient exploitation of delay sensitive and computation demanding applications, such as augmented reality or virus scanning. To satisfy even high demands of the devices on computation, the computing power distributed over nearby cloud-enhanced SCeNBs can be virtually merged together under one Virtual Machine (VM).

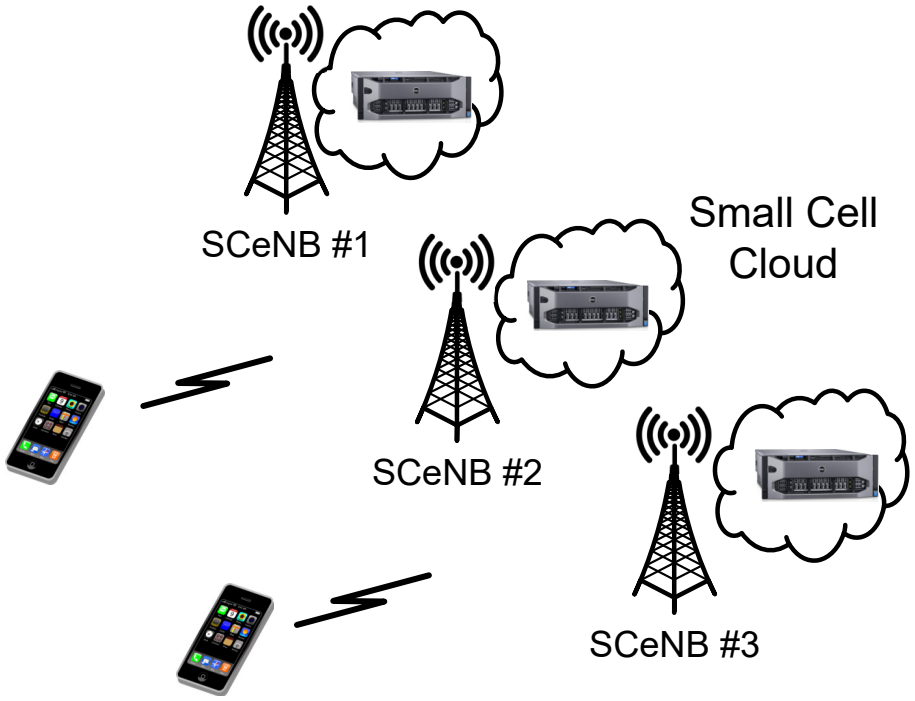


Figure 3 – Small cell cloud.

The application is then offloaded from the device to the SCeNBs if it is profitable from energy or delay point of view. The VMs are deployed at SCeNBs with respect to their communication and computation capabilities (LOBILLO et al., 2014).

2.1.1.3 Cloudlet Server

A Cloudlet server is a desktop or a cluster of low-cost multicore computers located in the same WLAN as the mobile device; it can provide cloud services on a small scale and is commonly found in domestic, corporate, and public environments (JUNIOR et al., 2017). Unlike the Public cloud, a Cloudlet server exists at physical proximity to the mobile device, usually at a 1-hop distance and is accessible using a high speed wireless link such as WiFi (SHAUKAT et al., 2016). Also referred to as a "data center in a box", a Cloudlet server is self-managing, requiring little power, Internet connectivity and access control (SATYANARAYANAN et al., 2009).

In a MCC paradigm, a cloudlet-based approach offers the following advantages over a cloud-based approach: low latency, higher bandwidth, offline availability, cost effectiveness, and decentralized (SHAUKAT et al., 2016).

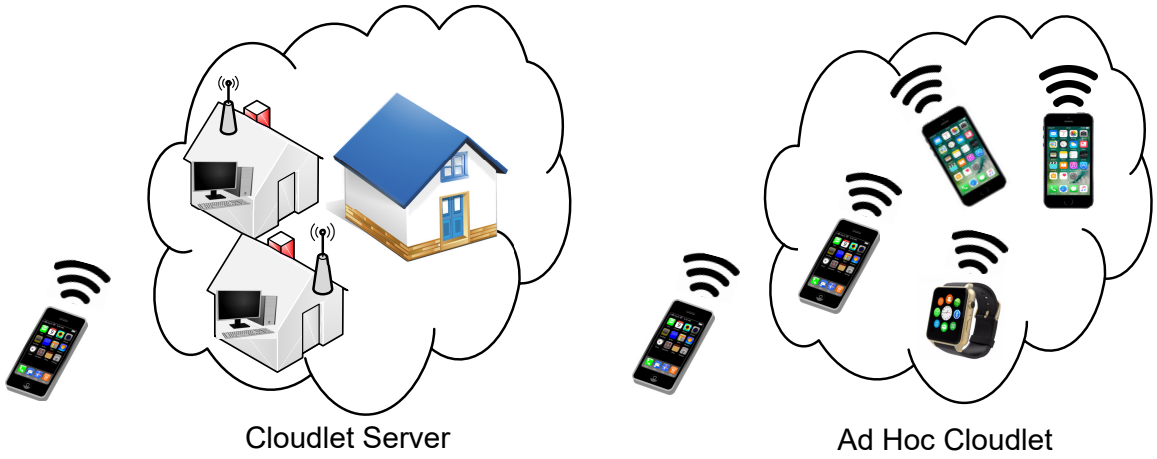


Figure 4 – Cloudlet server and Ad-hoc cloudlet.

Figure 4 illustrates the Cloudlet server on the left. The server performs management, resource monitoring, and task scheduling. Moreover, the server is considered as a secure and trustful resource-rich device. Although the Cloudlet server reduces the response time of the application execution in the MCC by mitigating the Wide Area Network (WAN) latency, this technology suffers from the compute-intensive and delay-inducing processes of cloudlet discovery, VM creation and deployment, application partitioning, authentication, and authorization. The local network service provider also hesitates in deploying the Cloudlet server and providing access to outside users for accessing the private networks. There is a need to provide incentives, such as charge per-resource-usage, to provoke the local network service providers to allow the outside users for accessing and utilizing the network resources. Security and authorization of local network resources are the big challenges in Cloudlet server deployment to prevent outside users from accessing the private resources (SHAUKAT et al., 2016).

2.1.1.4 Ad-Hoc Cloudlet

A Ad-hoc cloudlet is a group of mobile devices with more powerful processing and autonomous energy that shares its resources with local neighbors (e.g., resource-poor mobile devices) (JUNIOR et al., 2017). It is an alternative solution to the Cloudlet server that mitigates several WAN bottlenecks, such as low throughput and longer delay. The Ad-hoc cloudlet provides the resources to mobile devices when either no or weak wireless Internet connections are available to the cloud or the device is unable to find the nearby available resources. In the Ad-hoc cloudlet, the mobile devices have to perform management functions, authentication, resource monitoring, and task scheduling by themselves in a

distributed manner that consumes mobile device energy and processor cycles (AHMED et al., 2015). Figure 4 illustrates the Ad-hoc cloudlet on the right.

This model makes use of Device-to-Device (D2D) communication, which is provided by technologies such as *WiFi-Direct* and *Bluetooth Low Energy*. The limitation of D2D communication is the strict requirement on the contact duration between a producer device and a consumer device to guarantee enough processing time for the offloaded computational task. Once a producer device and a consumer device are disconnected due to mobility or other network dynamics while the offloaded computational task is not finished, the computation execution is failed (CHEN et al., 2015).

Finally, the Table 1 makes a comparison of the main cloud models on three perspectives: applicability, advantages and disadvantages.

Table 1 – Comparison of the main mobile cloud models.

Resource	Applicability	Pros	Cons
Public Cloud	Can be used for delay-tolerant applications (e.g., e-commerce, social network) and when there is little or great demand for computing power.	- Centralized Datacenters and administered by professionals - Wide availability and scalability - Able to support thousands of users simultaneously	- Makes use of the Internet and therefore routing occurs through multiple hops, which may result in unpredictable delays - Incurs additional costs for contracting cloud services and Internet access providers.
Cloudlet Server	Can be used for delay-sensitive applications (e.g., video, audio, image processing) and when there is a great demand for computing power.	- Makes use of WiFi to obtain high throughput and low latency - Better quality of service (QoS) to meet the needs of few users in a local network - Low cost and decentralized architecture	- Does not have management and security policies - Not scalable - Are not always available - Service coverage area restricted to that of WLAN
Ad-Hoc Cloudlet	Can be used when there is no access to the Internet and in the neighborhood there is a group of mobile devices with greater energy autonomy and computational power	- Lower communication cost and short transmission delay - Does not depend on a wired network infrastructure - Mobile devices are producers and consumers of services	- Does not have management and security policies - Local D2D connectivity might be unavailable due to network dynamics

2.1.2 Computation Offloading

According to (ENZAI; TANG, 2014), computation offloading is a mechanism where resource-intensive computations are migrated from a mobile device to the resource-rich cloud or server or nearby infrastructure. Computation offloading evolves from serving client-server paradigm to mobile systems and cloud computing. Client-server paradigm is still part of a cloud. However, cloud computing implies business, data stores, and other resources which are remotely hosted. Nonetheless, simply adopting the migration concept of client-server to cloud computing is not straightforward. Characteristics which are unique to cloud computing such as virtualization and elasticity of cloud resources need to be taken into consideration.

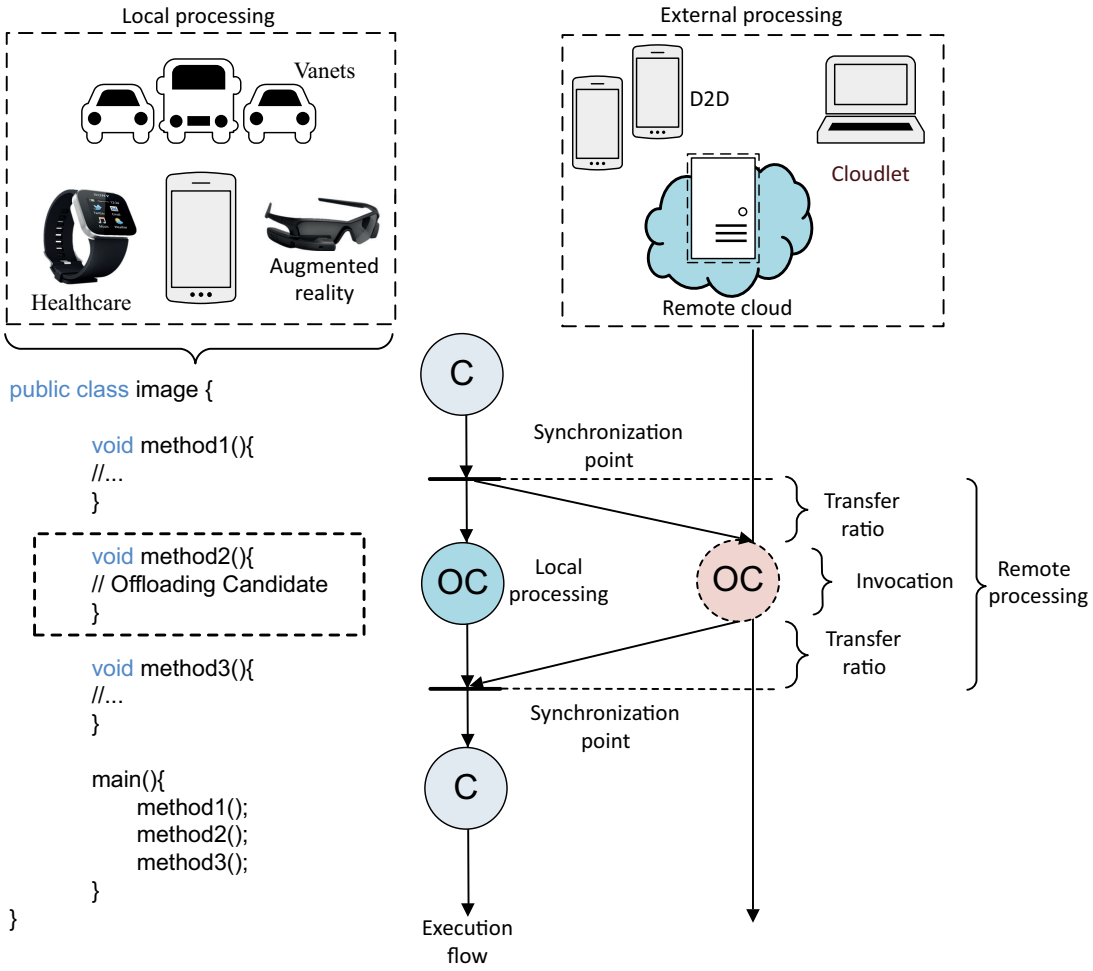


Figure 5 – Mobile cloud offloading.

The computational offloading architecture adapted from (FLORES et al., 2015) is shown in Figure 5. The architecture consists of two parts: a clients cluster and a servers cluster. In this approach, any mobile entity (connected cars, smartphone, smartwatch, smartglasses, tablet, and others) can outsource a computational task to any available cloud model, e.g. cloudlet server, remote cloud or ad-hoc cloudlet. This is also called D2D in other studies. Therefore, portions (C) of code (e.g. Method, Thread, or Class)

are identified as Offloading Candidates (OCs) by a software programmer, indicating that part of the code should be offloaded.

2.1.3 Offloading Mechanisms

The computational offloading technique can be categorized into static and dynamic, depending on when the offloading decision occurs.

The static offloading approach makes use of performance prediction models or offline profiling to estimate the performance. The application is then partitioned into client and server partitions which may subsequently be executed (KHAN, 2015). Static partitioning is done during development, i.e. the software developer explicitly select the code that should be offloaded using special static annotations (e.g., *@Offloadable*, *@Remote*) (FLORES et al., 2015).

The dynamic offloading strategy initially perform static analysis of the code and instrumentation in order to perform dynamic/online profiling during execution. Based on the information obtained from dynamic profiling, the application is partitioned into client and server partitions. The execution then continues with the updated configuration (KHAN, 2015). Dynamic partitioning is conducted during execution, i.e. an automated mechanism analyzes the code implicitly during runtime. Thus, once the application is installed in the device, the mechanism selects the code to be offloaded.

Table 2 – Static offloading VS Dynamic offloading.

Offloading strategy	**Advantage**	**Disadvantage**
Static	Low overhead during execution	Beneficial only when the parameters can be accurately predicted in advance
Dynamic	Adapt to different runtime conditions	High overhead during execution

The pros and cons between static and dynamic partitioning are highlighted in Table 2. Static partitioning could not adapt to varying network conditions efficiently and also places more responsibility on programmers. A partitioning mechanism which automatically computes estimated partitioning solution is more suitable. However, extra computing cost for dynamic partitioning is still an issue. Nonetheless, dynamic partitioning is a common method to incorporate adaptive feature in computation offloading framework (ENZAI; TANG, 2014).

2.1.4 Mobility Techniques used in MCC

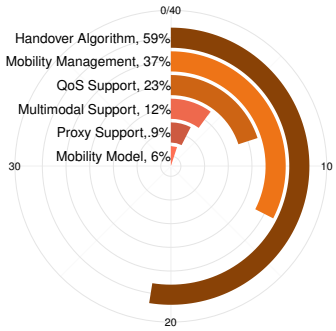
A significant amount of research has been performed on MCC connectivity. Aiming to conduct these studies, most of the researchers have adopted mobility techniques

to provide seamless connectivity and service, i.e., it refers to the uninterrupted service provisioning across different wireless access networks and optimal service delivery via the most appropriate cloud resource. However, there is no common list of which mobility techniques could be used in MCC research and a Systematic Mapping Study (SMS) could give important directions in this sense. A SMS is a type of investigation that has an evidence-based nature, applied in order to provide an overview of a research area by characterizing it (PETERSEN et al., 2008). Before presenting any mobility strategy that could solve the pursued objective of this thesis, we have applied a SMS, aiming to identify mobility techniques used in MCC. We have executed a SMS by means of analyzing four mobility's facets in MCC (mobility strategy, decision criteria, network technology, and cloud model). We synthesized implications for practicing, identifying research trends, open issues, and areas for improvement.

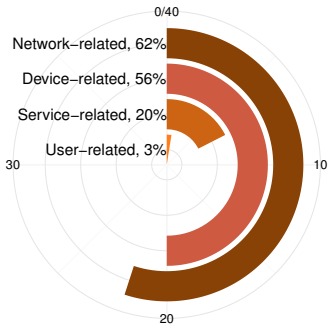
Starting from 736 papers, we filtered 36 studies that used mobility techniques for their proposals. Given the current state of MCC research, we judge that there are few studies with controlled experiments using real solutions. In our study, only 36 papers used mobility techniques in the their proposals, probably because this field is still relatively recent, with the first effectively primary study dating from 2012 (BIFULCO et al., 2012a). Following, we used circular bar charts to report the percentage of studies (i.e., number of papers) considering the corresponding facets. For instance, in Figure 6a the value "Handover Algorithm, 59%" means that means that 21 studies out of 36 use a handover algorithm as the mobility strategy, corresponding to 59%. Figure 6 summarizes our findings.

Mobility strategy, illustrated in Figure 6a, is characterized by identify the mobility mechanisms, models, and algorithms used to support seamless mobility and smart decision-making. We have observed that the majority of the solutions have adopted algorithms for the handover decision, followed by mobility management. Handover algorithm is characterized as steps/strategy with the aim of guiding the decision about triggering (or not) the handover from one PoA to another, taking into account many decision parameters, while mobility management is responsible for locating a mobile device, initializing handover, selecting a new network, registering the mobile device with the new network in the case of different vendors, access network or domain and executing the handover. Figure 6b shows that the most utilized decision criteria are network and device related, such as the Received Signal Strength Indication (RSSI), Round Trip Time (RTT), bandwidth, delay, cost/price, packet loss, battery power, location, and velocity. Among the network technologies used (Figure 6c), as expected, WiFi is the most envisioned technology, since among other motivations its high bandwidth and low latency are important features for delay-sensitive applications that require QoS. Regarding cloud model (Figure 6d), public cloud was the most employed model in MCC, with 28 studies. The decisions related to mobility strategy in this PhD research shall take into account these results, aiming to

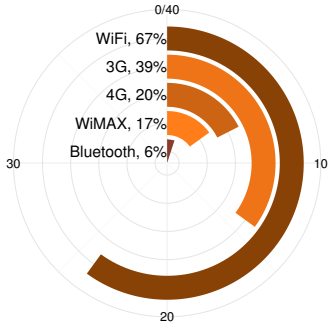
adopt similar test-beds and then make it possible to compare results with related studies.



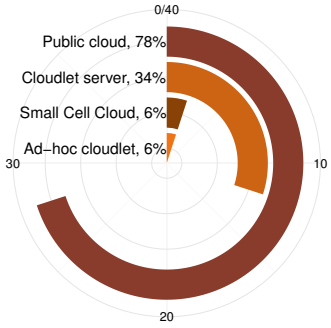
(a) Mobility strategy.



(b) Decision criteria.



(c) Network technology.



(d) Cloud model.

Figure 6 – Mapping by Quantity of Occurrences.

2.2 Software-Defined Networking

Modern Internet infrastructure consists of a set of networking devices with Application-Specific Integrated Circuits (ASICs) and chips that are used to achieve high throughput, thus realizing hardware-centric networking. However, the current hardware-centric Internet infrastructure suffers from several shortcomings, such as manageability, flexibility, and extensibility. Networking devices usually support a handful of commands and configurations based on a specific embedded Operating System (OS) or firmware. As a result, network administrators are limited to a set of pre-defined commands, even though it would be easier, simpler,and more efficient to support more protocols and applications if it were possible to program network controls in ways that are more responsive and flexible. In addition, researchers usually have to make their own testbeds or take advantage of simulations rather than real world implementation scenarios to realize their ideas. In other words, innovation and research is costly under the current condition of hardware-centric networking (FARHADY; LEE; NAKAO, 2015).

To overcome such limitations, the Software-Defined Networking (SDN) concept has been proposed. SDN can be defined as *"an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today's applications. This architecture decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services"* (FUNDATION, 2012). This definition is quite comprehensive, making it clear that the main advantage of the SDN paradigm is to allow different policies to be dynamically applied to the network by means of a logically centralized controller, which has a global view of the network and, thus, can quickly adapt the network configuration in response to changes (KIM; FEAMSTER, 2013). At the same time, it enables independent innovations in the now decoupled control and data planes, besides facilitating the network state visualization and the consolidation of several dedicated network appliances into a single software implementation (KREUTZ et al., 2015).

2.2.1 Control and Data Plane

Given that the separation between data and control planes is at the core of the SDN technology, it is important to discuss them in some detail. Figure 7 shows a simplified SDN architecture and its main components, showing that the data and control planes are connected via a well-defined programming interface between the switches and the SDN controller.

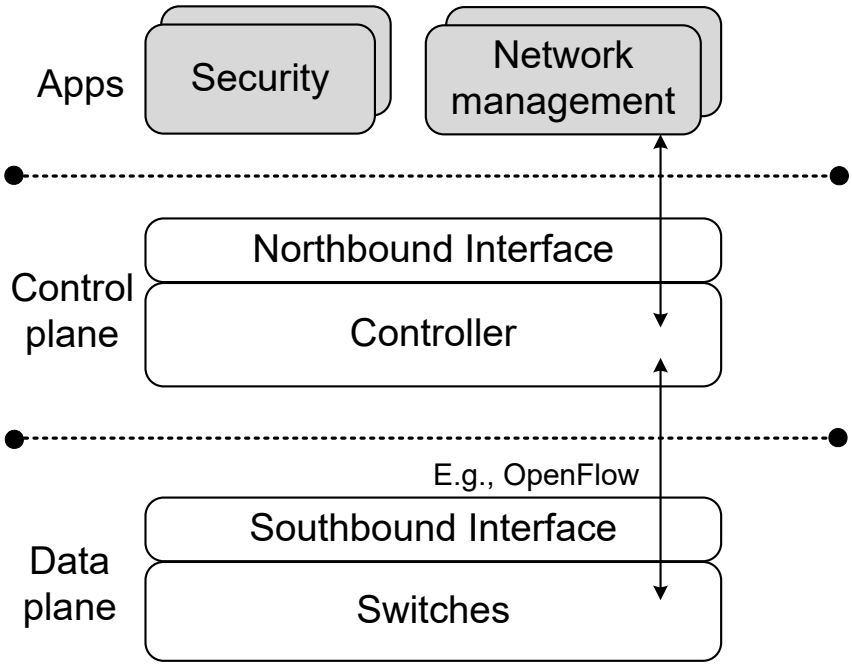


Figure 7 – Components of SDN.

The data plane corresponds to the switching circuitry that interconnects all devices

composing the network infrastructure, together with a set of rules that define which actions should be taken as soon as a packet arrives at one of the device's ports. Examples of common actions are to forward the packet to another port, rewrite (part of) its header, or even to discard the packet (FARHADY; LEE; NAKAO, 2015).

The control plane is the part that manipulates forwarding devices through a controller to achieve the specific goal of the target application. The controller uses the southbound interface of the SDN-enabled switch to connect to the data plane. This interface can be implemented using protocols such as OpenFlow 1.0 and 1.3 (OPENFLOW, 2009)(OPENFLOW, 2012), OVSDB (PFAFF; DAVIE, 2013), and NETCONF (ENNS; BJORKLUND; SCHOENWAELDER, 2011). The control plane concentrates, thus, the intelligence of the network, using information provided by the forwarding elements (e.g., traffic statistics and packet headers) to decide which actions should be taken by them (KREUTZ et al., 2015).

Finally, developers can take advantage of the protocols provided by the control plane through the northbound interfaces, which abstracts the low-level operations for controlling the hardware devices similarly to what is done by operating systems in computing devices such as desktops. These interfaces can be provided by Remote Procedure Calls (RPC), restful services and other cross-application interface models. This greatly facilitates the construction of different network applications that, by interacting with the control plane, can control and monitor the underlying network. This allows them to customize the behavior of the forwarding elements, defining policies for implementing functions such as firewalls, load balancers, intrusion detection, among others (MEDEIROS et al., 2015).

2.2.2 OpenFlow Protocol

The OpenFlow protocol is one of the most commonly used southbound interfaces, being widely supported both in software and hardware, and standardized by the Open Networking Foundation (ONF). It works with the concept of flows, defined as groups of packets matching a specific (albeit non-standard) header (MCKEOWN et al., 2008), which receive may be treated differently depending how the network is programmed. The controller manipulates the flow tables of the switch by adding, updating, and deleting flow entries. This happens reactively (when the controller receives a packet from the switch) or proactively according to the OpenFlow controller implementation. The controller communicates with the switches via a secure channel (see Figure 8). The OpenFlow switch supports the flow-based forwarding by keeping one or more flow tables. Flow tables are used to determine how the given packets are handled. Each flow table entry contains a set of packet fields to match (e.g., Ethernet addresses and IP addresses), and a set of actions to be applied upon a match (e.g., send-out-port, modify-field, or drop). Flow entries also maintain counters to collect statistics for particular types of flow (e.g., the number of transmitted bytes) (FARHADY; LEE; NAKAO, 2015).

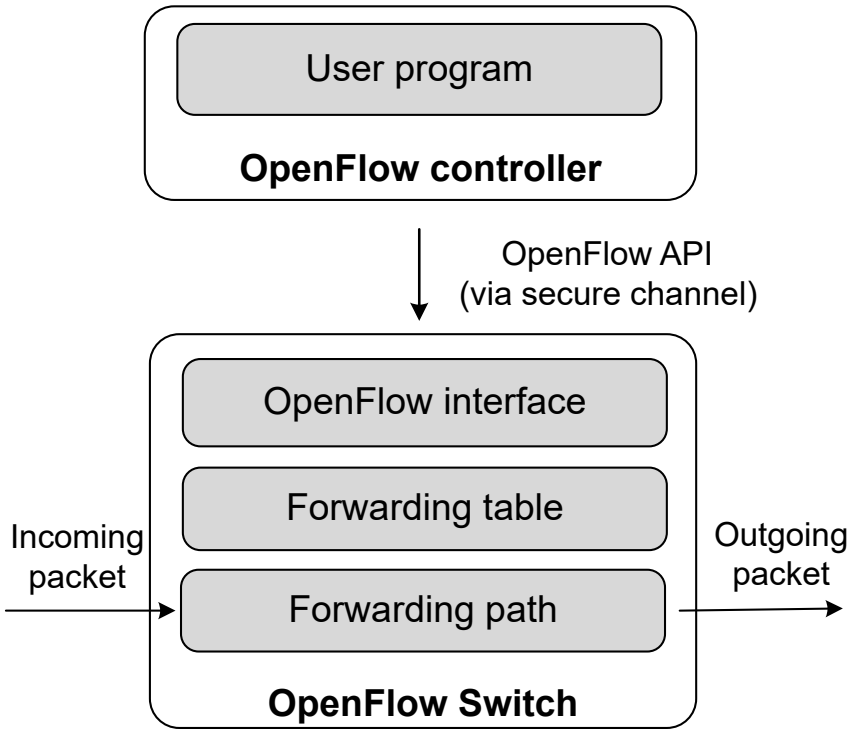


Figure 8 – OpenFlow switch.

When the switch receives a packet that causes a miss in the forwarding table matching process, the action to be taken depends on the table-miss flow entry. That packet can be forwarded to the controller, passed to the next flow table, or dropped. When the packet is forwarded to the controller, the controller then decides how to handle this packet. The controller can drop the packet, or it can add a flow entry that gives the switch direction on how to forward similar packets in the future (FARHADY; LEE; NAKAO, 2015).

2.3 Computational Context

The term context has been defined by many researchers. An example of context definition in the ubiquitous computing view was proposed by Abowd and Mynatt (ABOWD; MYNATT, 2000) which identified the five W's (Who, What, Where, When, Why) as the minimum information that is necessary to understand context. Schilit et al. (SCHILIT; ADAMS; WANT, 1994) and Pascoe (PASCOE, 1998) have also defined the term context. Dey claimed that these definitions were too specific and cannot be used to identify context in a broader sense and provided a definition for context as follows:

"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves"(ABOWD et al., 1999).

In addition, Sanchez et al. (SANCHEZ et al., 2006) explained the distinction between raw data and context information as follows: Raw data is unprocessed and retrieved di-

rectly from the data source, such as physical or logical sensors, while context information is generated by processing raw sensor data. For example, the logical monitor tracks the smartphone's CPU usage and the acquired values can be considered as raw data. Once we use logical monitor to produce statistical information about the health of smartphone hardware, we call it context information. Therefore in general, the raw data values produced by sensors can be considered as data. If this data can be used to generate context information, we identify these data as context. Therefore, mostly what we capture from sensors are data not the context information.

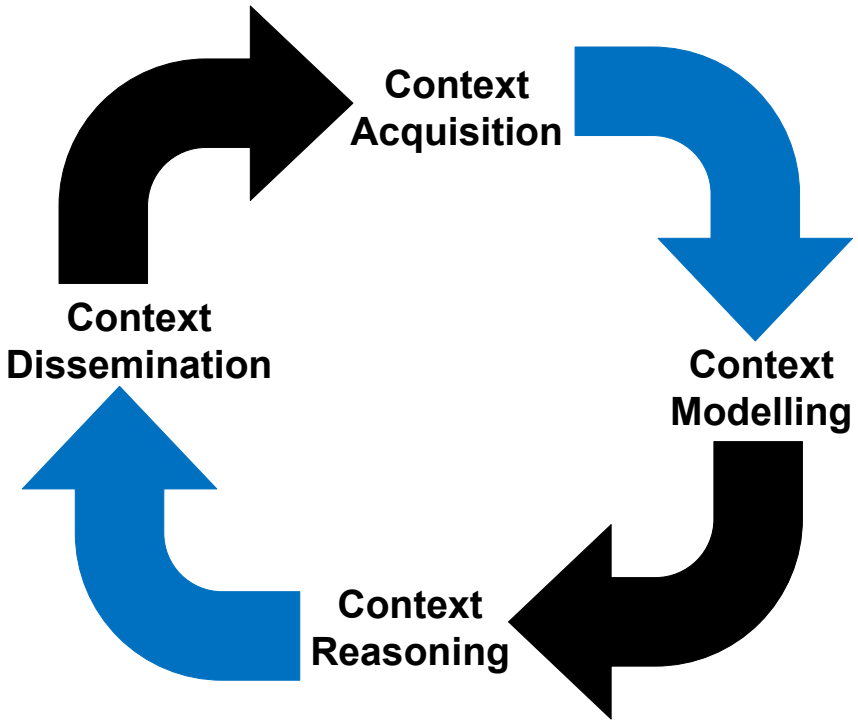


Figure 9 – Context life cycle.

A context life cycle shows how data moves from phase to phase in software systems (e.g. application, middleware). Specifically, it explains where the data is generated and where the data is consumed. According to Figure 9, this context life cycle consists of four phases. First, context needs to be acquired from various sources. The sources could be physical or virtual (context acquisition). Second, the collected data needs to be modelled and represented according to a meaningful manner (context modelling). Third, modelled data needs to be processed to derive high-level context information from low-level raw data (context reasoning). Finally, both high-level and low-level context needs to be distributed to the consumers who are interested in context (context dissemination) (PERERA et al., 2014).

2.3.1 Context-Aware Features

The term context-aware, also called sentient, refers to a system's awareness of its surrounding environment and it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task (BETTINI et al., 2010)

With this definition we can easily identify whether this system is a context-aware system or not. Context-awareness systems typically should support acquisition, representation, delivery, and reaction (DEY; ABOWD; SALBER, 2001). In addition, there are three main approaches that we can follow to build context-aware applications (HU; INDULSKA; ROBINSON, 2008).

- **No application-level context model:** Applications perform all the actions, such as context acquisition, pre-processing, storing, and reasoning within the application boundaries.
- **Implicit context model:** Applications uses libraries, frameworks, and toolkits to perform context acquisition, pre-processing, storing, and reasoning tasks. It provides a standard design to follow that makes it easier to build the applications quickly.
- **Explicit context model:** Applications use a context management infrastructure or middleware solution. Therefore, actions such as context acquisition, pre-processing, storing, and reasoning lie outside the application boundaries. Context management and application are clearly separated and can be developed and extend independently.

2.3.2 Context in Mobile Cloud

In MCC, a context-aware offloading system is one that is able to monitor, collect, select, process and share entity's context information, and which is involved in decision-making and the execution of computational offloading. It is used extensively in environments where an energy source or computing power is scarce. The aim of this kind of system is to be aware of all context information to thus offload only when cloud processing is better than local processing.

The offloading process could be improved by using contextual information to enable dynamic decision-making, in order to offload only when the context is advantageous, since static offloading is not beneficial in all contexts (HUANG; WANG; NIYATO, 2012). In MCC, an advantageous context is defined as a context where the set of information makes the offload processing cost less than the local execution of a task.

An example in a mobile healthcare environment is smart-band technology (WANG et al., 2013), which has numerous sensors whose function is monitoring a range of indicators, such as heart beats, the insulin level in blood, body temperature, besides others features. These smart-bands do not have enough battery or computational power to deal with

all kinds of tasks, considering there are tasks that requires high computational power and cannot be done locally. So, a context-aware offloading system is used in this case to analyze the context, with the system being aware of local processing capability, the number of sensors that the band has, the data size to be transfered, network bandwidth, and cloud capacity, with the purpose of choosing the best place to process all the raw data. Therefore, the smart-band will process data locally or in the cloud only when the context is advantageous for this to be done.

This kind of system can be also used in Vehicular Ad-Hoc Networks (VANETs) that aim to facilitate a more comfortable driving experience and a more enjoyable journey (WANG et al., 2017) (MERSHAD, 2012) through providing travel assistance and vehicle infotainment. In this way, cars and Roadside Units (RSUs) can share processing power - termed resource Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) respectively - since cars now have a lot of sensors that are used together to provide information about the vehicle, its immediate environment, and media content for the driver's and passengers' comfort. In VANETs, context informations can change abruptly and the connection to other vehicles or infrastructure can be intermittent(WANG et al., 2017). This means that all content (e.g., audio, picture or video) requested by the passengers, or data collected by sensors, has to be processed sometimes in VANETs and sometimes locally, according to data size, complexity of task, network bandwidth, the distance between cars/RSUs and the velocity of vehicle. It makes a context-aware offloading system necessary to decide when and where is the best place to process data periodically.

Another example is Mobile Social Networking (MSN). The MSN involves the interaction between users who have the same interests and/or objectives through their mobile devices within virtual social networks (HU et al., 2015). In this way, users' devices can share processing resources to be able to process data together, with the same objective. In one instance, for example, it could be an image processing or a text translation that is required, and the image size or text length are the features to consider in order to determine whether to offload to another device(s) if the task cannot be done by one device.

2.4 Machine Learning

Machine Learning (ML) can be considered as a subfield of Artificial Intelligence since those algorithms can be seen as building blocks to make computers learn to behave more intelligently by somehow generalizing rather that just storing and retrieving data items like a database system and other applications would do. ML is defined, among others by (KOTSIANTIS, 2007), as a system or an algorithm that can learn automatically based on past experience, which means determining a set of rules from instances (examples in a training set) (DOMINGOS, 2012) without any external assistance from a human (DAS; BEHERA, 2017). The core function of ML attempts is to tell computers how to automat-

ically find a good predictor based on past experiences and this job is done by a good classifier. Classification is the process of using a model to predict unknown values (output variables), using a number of known values (input variables).

In ML algorithms every instance of a particular dataset is represented by using the same set of features. The nature of these features could be continuous, categorical or binary. If instances are given with known labels (i.e., the corresponding correct outputs) then the learning scheme is known as supervised, while in unsupervised learning approach the instances are unlabeled. In this research, we will focus our attention on the methods which are being used for supervised learning, because the proposed system uses Supervised Machine Learning (SML) approaches.

2.4.1 Supervised Learning

A subdomain of the ML field is supervised learning. In this subdomain, the training is undertaken with labeled data, with inputs and desired outputs (ROKACH; MAIMON, 2005). In this way, classification algorithms whose function is to classify an instance based on its training set, can decide which output label the input belongs to. The process of applying SML to a real-world problem is described in below Figure 10.

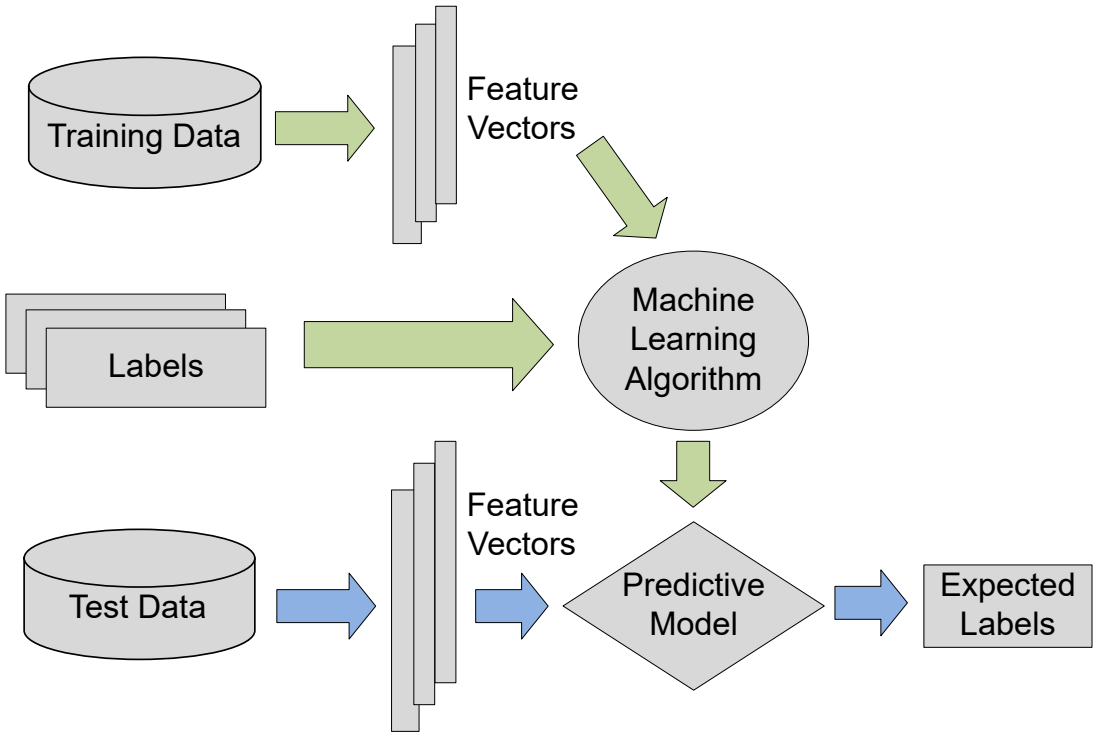


Figure 10 – Supervised Machine Learning Model.

In supervised learning the first step is dealing with dataset. In order to perform a better training on data set an appropriate expert could suggest better selection of features. If concerned expert is not in reach, then the other approach is “brute-force”, which means measuring everything available in the hope that the right (informative, relevant) features

can be isolated. However, a dataset collected by the "brute-force" method is not directly suitable for induction. Ultimately, in most cases it contains noise and missing feature values, and therefore requires significant preprocessing. In the next step, data preparation and preprocessing is a key function of researcher in SML (KOTSIANTIS, 2007).

Decision tree, k-nearest neighbor, rule-based, and naive bayes are ML reasoning techniques collectively called classification algorithms. They are important and widely used in pervasive and ubiquitous computing mainly due to their accuracy and relatively inexpensive computational cost. These four algorithms are described below (MUHAMMAD; YAN, 2015).

2.4.2 Decision Tree

A 'Decision Tree' is a supervised learning technique where the system builds a tree from a dataset that can be used to classify data. The tree is built using a top-down strategy, where the root is the feature that has the greatest information gain and the process continues recursively until all instances of the subset belong to the same label/class. The information gain of a feature is a score that indicates how important it is. The closer to the root, the more important is the attribute. It is made this way in order to produce the lowest quantity of rules.

In Artificial Intelligence field, Quinlan has contributed through his ID3 and C4.5 algorithms (QUINLAN, 2014). C4.5 is one of the most popular and the efficient method in decision tree-based approach (WU et al., 2008). It constructs the decision tree with a "divide and conquer" strategy. In C4.5, each node in a tree is associated with a set of cases. Also, cases are assigned weights to take into account unknown attribute values. At the beginning, only the root is present and associated with the whole training set and with all case weights equal to 1.0. At each node, the following "divide and conquer" algorithm (see Algorithm 1) is executed, trying to exploit the locally best choice, with no backtracking allowed.

Let T be the set of cases associated at the node. The weighted frequency $freq(C_i, T)$ is computed (Line 2) of cases in T whose class is C_i for $i \in [1, NClass]$ (RUGGIERI, 2002).

If all cases (Line 3) in T belong to a same class C_j (or the number of cases in T is less than a certain value), then the node is a leaf, with associated class C_j (respectively, the most frequent class). The classification error of the leaf is the weighted sum of the cases in T whose class is not C_j (respectively, the most frequent class) (RUGGIERI, 2002) (KOTSIANTIS, 2013).

If T contains cases belonging to two or more classes (Line 7), then the information gain of each attribute is calculated. For discrete attributes, the information gain is relative to the splitting of cases in T into sets with distinct attribute values. For continuous attributes, the information gain is relative to the splitting of T into two subsets, namely, cases with an attribute value not greater than and cases with an attribute value greater

than a certain local threshold, which is determined during information gain calculation (RUGGIERI, 2002) (KOTSIANTIS, 2013).

Algorithm 1 Pseudocode of the C4.5 Tree-Construction Algorithm

```
1:  procedure FormTree(T)
2:      ComputeClassFrequency(T)
3:      if OneClass ∨ FewCases then
4:          return a leaf
5:      end if
6:      create a decision node N
7:      for Attribute A do
8:          ComputeGain(A)
9:      end for
10:     N.test = AttributeWithBestGain
11:     if N.test is continuous then
12:         find Threshold
13:     end if
14:     for T' in the splitting of T do
15:         if T' is Empty then
16:             Child of N is a leaf
17:         else
18:             Child of N = FormTree(T')
19:         end if
20:     end for
21:     ComputeErros of N
22:     return N
23: end procedure
```

The attribute with the highest information gain (Line 10) is selected for the test at the node. Moreover, in case a continuous attribute is selected, the threshold is computed (Line 11) as the greatest value of the whole training set that is below the local threshold (KOTSIANTIS, 2013).

A decision node has s children if $T_1, ..., T_s$ are the sets of the splitting produced by the test on the selected attribute (Line 14). Obviously, $s = 2$ when the selected attribute is continuous, and $s = h$ for discrete attributes with h known values. For $i = [1, s]$, if T_i is empty, (Line 15) the child node is directly set to be a leaf, with associated class the most frequent class at the parent node and classification error 0 (RUGGIERI, 2002) (KOTSIANTIS, 2013).

If T_i is not empty, the "divide and conquer" approach consists of recursively applying the same operations (Line 18) on the set consisting of T_i plus those cases in T with an unknown value of the selected attribute. Note that cases with an unknown value of the selected attribute are replicated in each child with their weights proportional to the proportion of cases in T_i over cases in T with a known value of the selected attribute (KOTSIANTIS, 2013).

Finally, the classification error (Line 21) of the node is calculated as the sum of the errors of the child nodes. If the result is greater than the error of classifying all cases in T as belonging to the most frequent class in T, then the node is set to be a leaf and all subtrees are removed (RUGGIERI, 2002).

The information gain of an attribute a for a set of cases T is calculated as follows: If a is discrete, and $T_1, ..., T_s$ are the subsets of T consisting of cases with distinct known value for attribute a, then:

$$gain = info(T) - \sum_{i=1}^{s} \frac{|T_i|}{|T|} \times info(T_i), \tag{2.1}$$

Where

$$info(T) = -\sum_{j=1}^{NClass} \frac{freq(C_j, T)}{|T|} \times log_2\left(\frac{freq(C_j, T)}{[T]}\right) \tag{2.2}$$

is the entropy function. While having an option to select information gain, by default, however, C4.5 considers the information gain ratio of the splitting $T_1, ..., T_s$, which is the ratio of information gain to its split information:

$$Split(T) = -\sum_{i=1}^{s} \frac{|T_i|}{|T|} \times log_2\left(P\frac{|T_i|}{|T|}\right) \tag{2.3}$$

It is easy to see that if a discrete attribute has been selected at an ancestor node, then its gain and gain ratio are zero. Thus, C4.5 does not even compute the information gain of those attributes. If a is a continuous attribute, cases in T with a known attribute value are first ordered using a Quicksort ordering algorithm. Assume that the ordered values are $v_1, ..., v_m$. Consider for $i \in [1, m-1]$ the value $v = (v_i + v_{i+1})/2$ and the splitting:

$$T_1^v = \{v_j | v_j \leq v\} \quad T_2^v = \{v_j | v_j > v\} \tag{2.4}$$

For each value v, the information gain $gain_v$ is computed by considering the splitting above. The value v' for which $gain_{v'}$ is maximum is set to be the local threshold and the information gain for the attribute a is defined as $gain_{v'}$. By default, again, C4.5 considers (QUINLAN, 1996) the information gain ratio of the splitting $T_1^{v'}$, $T_2^{v'}$. Finally, note that, in case the attribute is selected at the node, the threshold is calculated (Line 11) by means of a linear search in the whole training set of the attribute value that best approximates the local threshold v' from below (i.e., which is not greater than v'). Such a value is set to be the threshold at the node.

Several advantages of using C4.5 are, amongst others: ease of viewing and understanding; the fact that a tree can be converted into a set of rules; that it is capable of handling both nominal or numerical (discrete) dataset input features together; and that it is able to handle instances of missing values from the database. On the other hand,

C4.5 requires that the target feature/attribute contains only discrete values. As decision trees use a "divide and conquer" methodology, C4.5 tends to perform well if a few highly relevant features are present, but less so if many complex interactions are present, causing replication problems where two or more subtrees are exactly the same (PAGALLO; HAUSSLER, 1990).

2.4.3 Rules

'Rule-Based' is the simplest and most straightforward method of reasoning. Rule-based reasoning are usually structure in an IF-THEN-ELSE format. It allows the generation of high level context information using low level context (PERERA et al., 2014) and represent each class by Disjunctive Normal Form (DNF). A statement is in DNF if it is a disjunction (sequence of ORs) consisting of one or more disjuncts, each of which is a conjunction (AND) of one or more literals. Below is an example of disjunctive normal forms.

A k-DNF expression is of the form: $((A_1 \wedge A_2 \wedge ...A_n) \vee (A_{n+1} \wedge A_{n+2} \wedge ...A_{2n}) \vee ...\vee)(A_{(k-1)n+1} \wedge A_{(k-1)n+2} \wedge ... \wedge A_{kn})$, where k is the number of disjunctions, n is the number of conjunctions in each disjunction, and A_n is defined over the alphabet $A_1, A_2, ..., A_j$. Here the objective is to build the smallest rule-set that is consistent with the training data (KOTSIANTIS, 2007). A good number of learned rules is usually a positive sign that the learning algorithm is attempting to remember the training set, instead of discovering the assumptions that govern it. A "separate-and-conquer" algorithm search for a rule that explains a part of its training instances, separates these instances and recursively conquers the remaining instances by learning more rules, until no instances remain (KOTSIANTIS, 2007). In below algorithm 2, a general pseudo-code for rule learners is presented.

Algorithm 2 General pseudo-code for rule learners

```
1: Initialize rule set to a default
2: Initialize examples to either all available examples or all examples not correctly handled by rule set
3: repeat
4:     Find the best rule with respect to examples
5:     if such a rule can be found then
6:         Add best to rule set
7:         Set examples to all examples not handled correctly by rule set
8:     end if
9: until no rule best can be found
```

The core difference between heuristics for rule-based algorithms and heuristics for decision trees algorithms is that the latter evaluate the average quality of a number of disjointed sets, while rule-based only evaluate the quality of the set of instances that is covered by the candidate rule (KOTSIANTIS, 2007). One of the most useful characteristic

of rule-based classifiers is their comprehensibility. In order to achieve better performance, even though some rule-based classifiers can deal with numerical features, some experts propose these features should be discredited before induction, so as to reduce training time and increase classification accuracy (MUHAMMAD; YAN, 2015).

The Repeated Incremental Pruning to Produce Error Reduction (RIPPER) algorithm (COHEN, 1995) works based on a set of rules. It builds a rule set by repeatedly adding rules to an empty dataset until all positive or negative examples converge to the same class (ENGG; SCIENCE, 2014). It also applies the reduced-error pruning method to determine whether a rule needs to be pruned. The reduced-error pruning method uses a validation set to estimate the generalization error of a classifier. To determine whether a rule should be pruned, RIPPER computes the following measure:

$$v_{RIPPER} = \frac{p - n}{p + n} \tag{2.5}$$

where p is the number of positive examples in the validation set covered by the rule, and n is the number of negative examples in the validation set covered by the rule.

The advantages of this classifier are flexibility, easy to extend, ability to add or modify new rules for new data, easily interpretable, and require for less resources (e.g., processing, storage) (LORENA et al., 2011). The disadvantages are that it is computationally expensive for large datasets, no validation or quality checking, and does not properly work in the presence of missing values (DUMA et al., 2010).

2.4.4 K-Nearest Neighbour

The K-Nearest Neighbor (K-NN) is a classification algorithm that also identifies the category of unknown instances based on the nearest neighbor whose class is already known. The choice of the best K changes depending on the dataset. The nearest neighbor is calculated on the basis of the value of K, i.e. how many neighbors are to be considered. The similarity is measured according to the distance from the K to the nearest instance. When given an unknown sample, a K-NN classifier searches the pattern space for the K training samples that are closest to the unknown sample. "Closeness" is defined in terms of Euclidean distance, where the Euclidean distance between two points, $X = (x_1, x_2, ..., x_n)$ and $Y = (y_1, y_2, ..., y_n)$ is:

$$d(x, y) = \sqrt{\sum_{k=1}^{n} (x_k - y_k)^2} \tag{2.6}$$

The unknown sample is assigned the most common class among its K nearest neighbors. When K=1, the unknown sample is assigned the class of the training sample that is closest to it in pattern space.

K-NN classifiers are instance-based or lazy learners in that they store all of the training samples and do not build a classifier until a new(unlabeled) sample needs to be

classified. This contrasts with eager learning methods, such a decision tree and backpropagation, which construct a generalization model before receiving new samples to classify.

The K-NN' algorithm (see Algorithm 3) is amongst the simplest of all machine learning algorithms. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common amongst its K nearest neighbors. K is a positive integer, typically small. If K=1, then the object is simply assigned to the class of its nearest neighbor. In binary (two class) classification problems, it is helpful to choose K to be an odd number as this avoids tied votes (PHYU, 2009).

Algorithm 3 Pseudo-code for instance-based learners

```
1: procedure InstanceBaseLearner(Testing Instances)
2:     for testing instance do
3:         Find the k most nearest instances of the training set according to a
4:         distance metric
5:         Resulting Class: most frequent class label of the k nearest instances
6:     end for
7: end procedure
```

It is clear, however, that K-NN has high computational complexity and memory requirements. Combined with these disadvantages, K-NN is easily fooled by irrelevant attributes. In contrast, K-NN training is very fast and can be effective depending on training data size, the algorithm is simple and easy to learn, and it is robust to noisy data (BHATIA; AUTHOR, 2010).

2.4.5 Naive Bayes

In a 'Naive Bayes' classifier, a probability function is used to define the class/label to which a given instance belongs (WU et al., 2008). It can predict class membership probabilities, such as the probability that a given sample belongs to a particular class. Bayesian classifier is based on Bayes' theorem, and assume that the effect of an attribute value on a given class is independent of the values of the other attributes. This assumption is called class conditional independence. It is made to simplify the computation involved and, in this sense, is considered "naive" (KANTARDZIC, 2011).

The probability is calculated using the following: the probability of a given instance belongs to one of the possible classes according to its measurement vector (features) and a function that means the conditional distribution of a given instance. The final probability is thus calculated by using these probabilities and functions distributed between the target classes. According to Bayes' theorem, the formula for calculating the conditional probability is expressed as

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)} \tag{2.7}$$

Where, $P(H)$ is the probability of hypothesis H being true. This is known as the prior probability. $P(X)$ is the probability of the evidence(regardless of the hypothesis). $P(X|H)$ is the probability of the evidence given that hypothesis is true, and $P(H|X)$ is the probability of the hypothesis given that the evidence is there. These probabilities may be estimated from the given data (HAN; PEI; KAMBER, 2011).

The Naive Bayes classifier works as follows:

1. Let T be a training set of samples, each with their class labels. There are k classes, $C_1, C_2, ..., C_k$. Each sample is represented by an n-dimensional vector, $X = x_1, x_2, ..., x_n$, depicting n measured values of the n attributes, $A_1, A_2, ..., A_n$, respectively.

2. Given a sample X, the classifier will predict that X belongs to the class having the highest a posteriori probability, conditioned on X. That is X is predicted to belong to the class C_i if and only if

$$P(C_i|X) > P(C_j|X) \quad \text{for } 1 \leq j \leq m, j \neq i. \tag{2.8}$$

   Thus we find the class that maximizes $P(C_i|X)$. The class C_i for which $P(C_i|X)$ is maximized is called the maximum posteriori hypothesis. By Bayes' theorem

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)} \tag{2.9}$$

3. As $P(X)$ is the same for all classes, only $P(X|C_i)P(C_i)$ need be maximized. If the class a priori probabilities, $P(C_i)$, are not known, then it is commonly assumed that the classes are equally likely, that is, $P(C_1) = P(C_2) = ... = P(C_k)$, and we would therefore maximize $P(X|C_i)$. Otherwise we maximize $P(X|C_i)P(C_i)$. Note that the class a priori probabilities may be estimated by $P(C_i) = freq(C_i, T)/|T|$.

4. Given data sets with many attributes, it would be computationally expensive to compute $P(X|C_i)$. In order to reduce computation in evaluating $P(X|C_i)P(C_i)$, the naive assumption of class conditional independence is made. This presumes that the values of the attributes are conditionally independent of one another, given the class label of the sample. Mathematically this means that

$$P(X|C_i) \approx \prod_{k=1}^{n} P(x_k|C_i) \tag{2.10}$$

   The probabilities $P(x_1|C_i), P(x_2|C_i), ..., P(x_n|C_i)$ can easily be estimated from the training set. Recall that here x_k refers to the value of attribute A_k for sample X.

   a) If A_k is categorical, then $P(x_k|C_i)$ is the number of samples of class C_i in T having the value x_k for attribute A_k, divided by $freq(C_i, T)$, the number of sample of class C_i in T.

b) If A_k is continuous-valued, then we typically assume that the values have a Gaussian distribution with a mean μ and standard deviation σ defined by

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\mu\sigma}} exp - \frac{(x-\mu)^2}{2\sigma^2}, \tag{2.11}$$

so that

$$p(x_k|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i}) \tag{2.12}$$

We need to compute μ_{C_i} and σ_{C_i}, which are the mean and standard deviation of values of attribute A_k for training samples of class C_i.

5. In order to predict the class label of X, $P(X|C_i)P(C_i)$ is evaluated for each class C_i. The classifier predicts that the class label of X is C_i if and only if it is the class that maximizes $P(X|C_i)P(C_i)$.

The advantages of using Naive Bayes are that: it is a simple way of calculating probability; it has fast training and testing; it can deal with numerical or nominal values; it requires low memory; irrelevant features are not considered; and it is short in computational time (ARCHANA; ELANGOVAN, 2014) (KEOGH, 2006). The disadvantages are that it requires strong feature independence assumptions to be made, which causes a loss of accuracy, and requires a huge number of records in the training dataset.

2.5 Summary

This chapter presented the theoretical fundamentals so that the reader can understand on the main topics that make up this thesis. The background about mobile cloud models and computational offloading, as well as software defined networking, allows the understanding of the application of such concepts in the design and development of mobile cloud infrastructures. Other topics covered in this chapter such as, computational context and machine learning guided the development of this work, and allowed the establishment of connections of these concepts with those applied in this research.

3 RELATED WORK

The related work is presented in two sections, which references the two core contributions of this thesis: context-aware offloading and mobile offloading system. The following analysis does not intend to provide an exhaustive view of published works on those topics, but rather to point out significant advances which go towards a similar direction as this research do, or give basis for future extensions.

3.1 Context-Aware Computational Offloading

Table 3 synthesizes the contributions of the most prominent works related to this part of the thesis. The references are categorized by tree aspects: (i) profiling sources, (ii) decision support, and (iii) accuracy.

Context sources refers to the physical and logical entities that provides relevant context information. The *Application* delivers data related to its components, methods, instructions, and input/output data, while *Device* is a information source about the local hardware, i.e., CPU and memory utilization, battery level, Global Positioning System (GPS), accelerometer and other sensors. *Wireless Network* delivers data related to the state and performance of the main components of a wireless network infrastructure, such as RTT, throughput, signal strength and connection status. Lastly, *Cloud/Cloudlet* delivers data related to access policies, availability, performance, and service cost. Information that can be monitored and captured are Virtual Central Processing Unit (vCPU) usage, virtual disk access time, and number of answered requests. According to Table 3, our proposal is the only one that implements all the context sources (details related to the profilers see the Section 4.1).

Decision support refers to technique that is used to assist the offloading decision, as well as to infer when offloading will improve performance. ThinkAir (KOSTA et al., 2012), MobiByte (KHAN et al., 2015), CADA (Ting-Yi Lin et al., 2013), and OMMC (GHASEMI-FALAVARJANI; NEMATBAKHSH; Shahgholi Ghahfarokhi, 2015) make offloading decisions considering the energy involved in computation and communication through reliable energy estimation models. On the other hand, MAUI (CUERVO et al., 2010) solves a 0-1 integer linear programming problem on the remote server to decide where each method must be executed and periodically updates the mobile device partition information. A primary requirement of linear programming is that the objective function and every constraint must be linear. However, in real world situations - just like in MCC environments - several heterogeneity and mobility problems are non-linear in nature.

Other solutions including EMCO (FLORES; SRIRAMA, 2013), MALMOS (EOM et al., 2015) and Majeed et al. (MAJEED et al., 2016) handle offloading decisions based on context

reasoning decision techniques, such as Fuzzy Logic, Instance-Based Learning (IBL), Perceptron, Naive Bayes, and Support Vector Machine (SVM). EMCO proposes the use of a fuzzy logic system to aggregate the profiling metrics and uses historical data for building an inference system that can be used by the mobile device to classify where the threads must be executed. The problem is that EMCO's results show only a few input parameters segregated by the fuzzy logic engine and consequently this ignores more complex scenarios.

Table 3 – Related work comparison - context-aware offloading.

Solutions	**Context Sources**				**Features**	
	App	**Device**	**Wireless Network**	**Cloud/ Cloudlet**	**Decision Support**	**Accuracy (%)**
MAUI	✓	✓	✓	✗	Integer linear programming	None
ThinkAir	✓	✓	✓	✗	Energy model	None
Mobibyte	✓	✓	✓	✗	Energy model	None
ARC	✗	✓	✓	✗	Naïve Bayes	None
Kwon et al.	✓	✓	✗	✗	Sparse Polynomial Regression	None
OMMC	✓	✓	✓	✗	TOPSIS and Energy model	None
mCloud	✓	✓	✓	✗	TOPSIS and Cost model	None
EMCO	✓	✗	✓	✓	Fuzzy logic	None
Rego et al.	✓	✓	✓	✗	Decision tree	None
MALMOS	✓	✗	✓	✗	IBL, Perceptron, and Naïve Bayes	86~93
CADA	✗	✓	✓	✗	Energy model	90
Majeed et al.	✓	✓	✓	✗	SVM	92
This Work	✓	✓	✓	✓	(K-NN, Rules, Naïve Bayes, and Decision Tree)	95

On the other hand, MALMOS provides an online training mechanism for the machine learning-based runtime scheduler such that it supports a policy that dynamically adapts scheduling decisions at runtime based upon the observation of previous offloading decisions and their correctness. The authors measured the scheduling accuracy of MALMOS by offloading each application to four different remote servers, while varying the network bandwidth and the input size. The system proposed by Majeed et al. uses SVM for accurately scheduling the component remotely or locally. The SVM classifier adapts its

decision according to external context (network bandwidth) and internal environmental data (e.g. memory usage, execution time and CPU utilization). Nevertheless, the solutions mentioned executes all the complex operations of the training and testing at regular intervals inside the mobile device, which can contribute to the overheads imposed on the system. Furthermore, the solutions completely fails to address the important aspect of the amount of energy used by the online training mechanisms.

Similar to our system, Rego et al. (REGO et al., 2017) uses a decision tree-based approach for handling offloading decisions through adaptive monitoring and historical data, while in the AnyRun Computing (ARC) system (FERRARI; GIORDANO; PUCCINELLI, 2016), the 'stup' component uses an inference engine based on a Naive Bayes decision model to assess the probability that offloading is advantageous compared to local execution. However, the first solution depends on the decision tree creation and concepts of entropy and information gain to identify the most relevant metrics for the offloading decision, which generates a high cost in terms of communication and computing. In addition, these two approaches do not evaluate their reasoning with regard to the perspective of energy consumption and application performance, unlike our work.

Knwon et al. (KWON et al., 2016) and mCloud (ZHOU et al., 2015) use other prediction techniques for beneficial offloading. For instance, mCloud is a code offloading framework that proposes a context-aware offloading decision algorithm aiming at providing code offloading decisions at runtime on the selection of wireless medium and appropriate cloud resources. The authors apply Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) (HWANG; LAI; LIU, 1993) for wireless medium selection by considering multiple criteria (e.g. wireless medium availability, network congestion, cost energy of the channel) and by using a cost estimation model that calculates the execution cost for each offloading request. Knwon et al. propose a feature-based prediction technique to overcome the input-sensitivity problem of mobile application performance. It is called *fMantis*, which generates a performance predictor for a mobile application that predicts whether or not a certain method will be executed according performance metrics including execution time, energy consumption, memory usage and state size.

Accuracy defines the percentage of the test dataset records that are correctly classified by the machine-learning classifier. Our work reached 95% accuracy with decision tree, rules-based, and K-NN algorithms (more details see Section 4.3.2).

Our proposal is the first offloading system developed and designed to handle raw context and transform them into appropriate context representation without human intervention. To the best of our knowledge, until now there has been no work in the literature that uses and evaluates multiple classifiers from experimental databases. The system is equipped with a decision engine that works with the main ML classification algorithms. Its historical database was built from experimental data, i.e. we first collected training examples (tasks execution locally and remotely by varying the context), then we labeled them

according to the processing time results (offloading or not). In sum, our work adopts a history-based prediction approach where we utilize the past profiled information as a basis for performance inference for future tasks. Unlike previous work, the system extrapolates features and experimental tests from benchmark applications that are easily configurable by the programmer to allow interaction with proposed middleware. It also improves the applications' performance as the offloading operations occur only in favorable contexts.

3.2 Mobility Mechanisms for Cloud-Based Application

Table 4 lists and classifies mobile offloading systems according to three aspects: (i) objective, (ii) data caching, and (iii) mobility management.

The *objective* defines the real benefit of using the associated solution. Ryu et al. (RYU; LEE; MUN, 2012), Qi et al. (QI et al., 2016), and DTSHM (YANG et al., 2014) are designed to achieve handover delay reduction, for instance. Ryu et al. is one of the first works to support the seamless handover for various wireless technologies in cloud computing. Its handover mechanism is based on the Fast Mobile IPv6 (FMIPv6), which considers the Probability Predictive Mode Failure (PPMF) to optimize the handover control in the IP layer. The Qi et al. mechanism supports multi-service soft handover, i.e., keeps multiple active cloud services and meets the QoS requirements, when mobile users roam. These benefits are ensured by the use of the Session Initial Protocol (SIP) to make all active services execute handover together. Lastly, DTSHM is a proactive service handover mechanism that uses the delaunay triangulation to build all the Access Point (AP) into a delaunay triangulation topology. In mobility scenarios, the cloud in advance authenticates and buffers the required data packets of a mobile device in standby APs, so it can quickly handover its service from a source to a target AP, when the handover process is triggered.

Similar to our work, Ravi et al. (RAVI; PEDDOJU, 2015), EMCO (FLORES; SRIRAMA, 2013), Cloudlet+Clone (MAGURAWALAGE et al., 2014), and StreamCloud (BACCARELLI et al., 2016) focuses on the mobile device's energy saving. Ravi et al. relies on a fuzzy vertical handover algorithm to trigger handover from a cloud resource to another when the device's energy consumption increases or the connection time with the resource decreases. In the Cloudlet+Clone system architecture, the offloading algorithm uses an energy model to estimate the energy consumption before offloading for a cloud; while, in the EMCO framework, the cloud side evidence analyzer is in charge of evaluating both mobile device and cloud variables so as to support the fuzzy decision engine for code offloading. The StreamCloud prototype relies on the CDroid module (BARBERA et al., 2014) in order to efficiently offload computation tasks. This module has a communication handling component that attains maximization of the per-client offloaded traffic rate at the minimum computing-plus-communication energy cost, by allowing each client to opportunistically select the more energy-efficient Radio Access Network (RAN) technology. The studies

EMCO, Cloudlet+Clone, and StreamCloud were not designed for supporting mobility in the MCC environment, unlike to our system, which addresses multiple goals (ensures performance and energy-efficiency simultaneously) and supports mobility for offloading operations.

Table 4 – Related work comparison - mobility-aware offloading.

Solutions	**Objetive**	**Data Caching**	**Mobility Management**
Ryu et al.	Reduce handover delay	None	FMIPv6 protocol
Qi et al.	Multi-service handover, Reduce handover delay	None	M-MIP protocol
M^2C^2	Multi-homing, Select best network and cloud	None	M-MIP protocol
Felemban et al.	Distributed cloud architecture	None	Active session record (ASR)
FMC	Transparent migration of services	None	OpenFlow rules
Cloudlet+Clone	Energy saving, Improve performance	Cloud	None
EMCO	Energy saving, Scalability (Multi-tenancy)	Mobile Device and Cloud	None
StreamCloud	Energy saving at both the mobile devices and data centers	Cloud	None
DTSHM	Reduce handover delay	Access Point	Delaunay triangulation and Master-Standby model
Ravi et al.	Energy saving, Service availability	None	Many Cloudlets/Clouds
This Work	Energy saving, Improve performance, Seamless offloading	Cloudlet and Cloud	OpenFlow rules and Network application

M^2C^2 (MITRA et al., 2015), Felemban et al. (FELEMBAN; BASALAMAH; GHAFOOR, 2013), and FMC (BIFULCO et al., 2012b) designed solutions for other purposes. For instance, M^2C^2 system supports multi-homing and establishes multi-paths between mobile devices and the cloud, while Felemban et al. propose distributed cloud architecture for multimedia services with integrated cloudlet and base station, which uses the cloudlet server to provide a proxy functionality and performs dynamic resource allocation. In the FMC, the OpenFlow protocol provides mobility features in a TCP/IP network for both users and services at the edge of the network. This technology provides both the ability to migrate network end-points and relatively relocate network services depending on users'

locations in order to guarantee adequate performance for the client-server communication. Unlike the FMC that uses global mobility and rewrite Internet Protocol (IP) address, our approach provides localized mobility, fewer signaling messages after handover, and keeps the same IP address.

Data caching is utilized to cache the results calculated by the remote cloud. This technique enables mobile data traffic reduction and improves the application response time. Cloudlet+Clone (MAGURAWALAGE et al., 2014), EMCO (FLORES; SRIRAMA, 2013), and StreamCloud (BACCARELLI et al., 2016) are examples of such technique, which usually involves identifying reusable results of generic code invocations that can be used to answer repeated requests from other applications. Similarly, Cloudlet+Clone introduces a data caching mechanism at the cloud, but the simulation environment does not allow understanding the real benefits of this mechanism according to the application's nature and input/output data size. In contrast to previous solutions, the DTSHM (YANG et al., 2014) mechanism adopts a cache scheme at the AP. The cloud service caches the device's required data packets in the target AP before handover occurs, and therefore, avoids retransmitting data packets from the source to the target AP. The remote caching technology in our system works on both at the network-edge with cloudlets and network-core in the public cloud, since that it is embedded in the middleware and it is capable of data caching for any mobile application. This technique makes the previous computation offloading available for a newly connected user that has not received the results due to a PoA handover.

Mobility management in wireless networks aims at seamlessly switching ongoing sessions between the network's PoA. In MCC, it ensures a seamless offloading operation while the mobile device changes its PoA, which means that ideally no delay in receiving results from the cloud should be perceived by the mobile applications' user. Solutions like DTSHM (YANG et al., 2014) and Ravi et al. (RAVI; PEDDOJU, 2015) supports handover for cloud service by adopting different techniques. DTSHM provides a proactive service handover mechanism using the delaunay triangulation. Thus, the authors developed a master-standby service model, which can accurately obtain the mobile device's target WiFi AP and minimize the handover delay. On the other hand, Ravi et al. relies on cloudlet and cloud interconnection to provide a seamless service to mobile users when they are moving from different APs. In the proposed work, the device's mobility is ensured with the discovery and communication modules on the cloudlet side and connection module on the device side. Felemban et al. (FELEMBAN; BASALAMAH; GHAFOOR, 2013) presented the connection handover mechanisms such as active session record and dynamic buffer allocation among cloudlets to manage handover calls and multimedia sessions. These mechanisms can reduce the impact of jitter delay on the multimedia services caused by the handover operation.

Qi et al. (QI et al., 2016) and M^2C^2 (MITRA et al., 2015) utilize the Multi-homed

Mobile IP (M-MIP) (ÅHLUND; BRÄNNSTRÖM; ZASLAVSKY, 2005) protocol to support seamless handover. M-MIP enables a mobile device to connect to several access networks simultaneously before initiating the handover process, i.e., a mobile device performs network discovery, network configuration, and network registration in advance for all available wireless networks. The mobile device periodically probes the registered network interfaces to select a target network for handover without disconnecting from the previous network interface, thereby minimizing network delay and packet losses during the handover process, but with the significant drawback of having high energy consumption. Ryu et al. (RYU; LEE; MUN, 2012) optimized the FMIPv6 in order to consider the PPMF that combines two operation modes (predictive and reactive) and is affected by the device's velocity, cell radius and the layer 2 trigger time. The proposed scheme reduces the overhead of the traditional FMIPv6 protocol.

Most of the aforementioned works focus on reducing the handover impacts on cloud services (e.g., Youtube, Twitter and Facebook). However, as far as we are concerned, no author has addressed computational offloading service continuity when users are connecting to different APs. Ravid et al. and FMC are the only solutions that tried to address the issue of cloud service discontinuation through mobility. Nevertheless, these proposals rely on a set of cloudlets and clouds to ensure service continuity for each access network. In the real world, these requirements would be expensive to deploy and maintain, since they need a significant amount of signaling messages in inter-cloud scenarios. Besides, DTSHM solution is adaptable to the high-speed movement environment, but more buffer space is required for AP caching thus increasing packet storage costs. Therefore, to the best of our knowledge, our system is the only one that was developed for supporting seamless offloading operations, allied to the high performance at a low cost, to improve both performance and energy saving.

3.3 Summary

This chapter highlighted the main related works that were found during the literature review on the topics mentioned. Although, it is important to emphasize that this is not an exhaustive view of published papers and related research papers. There may be other papers and theses which have made significant progress in this field, but to the best of our knowledge the combination of the features described in Tables 3 and 4 is one of the main factors that distinguish this work from the state-of-the-art.

4 CONTEXT-SENSITIVE SYSTEM

In this chapter, we first describe in detail our Context-Sensitive Offloading System (CSOS), its components and interactions (Section 4.1). Next, we discuss the system's development process and the implementation details (Section 4.2). Finally, we evaluate the proposed system in Sections 4.3 and 4.4.

4.1 Design Goals and Architecture

CSOS follows the standard client-server model. The CSOS Client components are located on the mobile device, while the CSOS Server components are located on the cloud or cloudlet. Figure 11 presents the overall architecture and provides the connections between the main components.

Within middleware, the architecture consists of three main components: a Decision Engine, Profilers, and a Proxy Handler. These components interact with each other to execute context-sensitive offloading. When a user initiates an application to process a task (e.g., an image), what are acquired are the most recent results of the profilers from the database for assisting the decision engine in making the correct inference. It can decide to execute the task locally or remotely based on the classification algorithm used (e.g., J48, JRIP). We define CSOS components as the following:

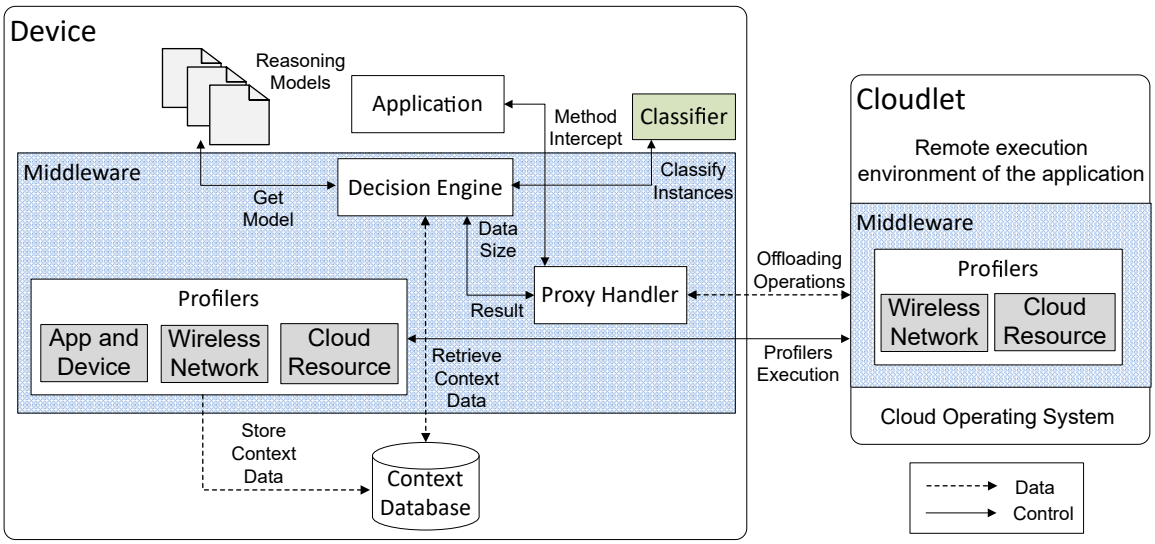


Figure 11 – CSOS Architecture.

- **Application**: this represents the three possible benchmarking applications (image editor, face detection, and an online game) that undertake resource-intensive computing. In the application interface, the users can choose the classification algorithm that they wish to use with CSOS.

- **Network Profiler**: this aims to capture wireless network information at runtime. The monitoring of network quality is critical in MCC environments since a poor network can cause packet loss and a delay in communication between device and cloud. We calculate the throughput by sending packets to the cloudlet server, which in turn estimates the device's upload rate. The server then sends packets to the smartphone to calculate the device's download rate.

- **Cloud/cloudlet profiler**: This is responsible for monitoring and collecting cloud/-cloudlet performance data in order to ensure that the cloud has larger processing capacity than the mobile device at a particular instant. This profiler calculates the vCPU load every second. The server's vCPU load is then encapsulated in the network profile to be sent back to the device.

- **App and device profiler**: this service monitors and collects application/hardware context data asynchronously at runtime. From the smartphone we gather the hardware values, more specifically the device's RAM, number of cores, and the maximum clock of each core. Beyond that, the smartphone's CPU load is also calculated from the application. We also capture the application name and calculate the data size to be processed. The data size is calculated by the *Proxy Handler* in the moments before the decision.

- **Context Database**: the profiling system runs every 35 seconds to gather raw context (such as application name, data size, smartphone CPU usage, cloud/cloudlet vCPU usage, upload/download rate, and smartphone hardware) and transforms it to high-level contextual information at runtime (for more details see Section 4.2). This information is saved in a database that always returns the most recent instance when asked to by the decision engine.

- **Decision Engine**: this component has three functions: (1) it is responsible for loading a context reasoning decision model (e.g., J48, JRIP, IBK) that is based on a training set to classify new instances; (2) it analyzes each attribute of the most recent instance in the database to collect information, such as the name, type, and possible values; (3) it uses the Weka library[1] to classify (local or remote) the latest instance stored in the database and sends the result to the proxy handler.

- **Proxy Handler**: its role is to intercept methods identified as offloading candidates with the *@remotable* markup. If the annotation is marked as static, the proxy handler sends the request for the method directly to the cloudlet or cloud, ignoring the decision engine. Otherwise, if it is marked as dynamic, it is responsible for calculating the current application data size and sends the corresponding value along with the

[1] Weka is a collection of ML algorithms. It permits the exportation of classification models to use them in personal Java code. http://www.cs.waikato.ac.nz/ml/weka/downloading.html

classification algorithm name to the decision engine. After that, it receives the result of the decision engine and runs either locally or remotely.

4.2 Configuration Process and Components Details

This section provides an overview of the steps a developer must follow to configure an application to use CSOS as well as providing more detailed information regarding the operation of its components.

4.2.1 Configuration Process

The configuration process is illustrated as follows in Figure 12. Each step is described below.

In the first step (i), the developer must define those factors that are important to be analyzed as relevant information for the offloading decision. Based on empirical evaluations, we identified six factors: network throughput, smartphone hardware, application category, data size, smartphone CPU, and cloudlet vCPU. Each one of these metrics can change independently of the others. In addition to some that periodically change over time, while others remain static. For example, a smartphone CPU usage can change every second, while the smartphone hardware retains the same configuration. Moreover, the combination of each metric value represents a profiler. For example, network bandwidth can assume congested, median, or free values, while the cloudlet vCPU can assume stressed, normal load, or relaxed values. Therefore, these various combinations can lead the decision engine to make different decisions based on the contextual information.

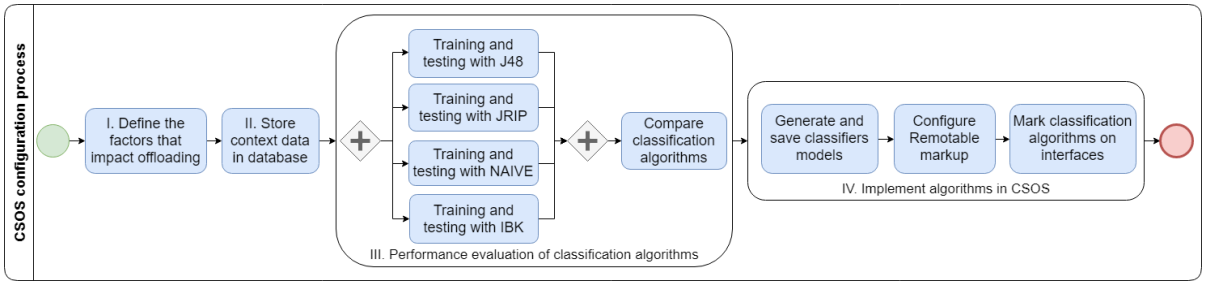


Figure 12 – CSOS configuration process.

The second step (ii) is the filling of a database with context information. In this phase, experiments are undertaken that change the contextual information and analyze the total processing time of each task between two configurations: static local, where the whole task is processed on smartphone; and static cloudlet, in which only the computing-intensive task is processed remotely in the cloudlet. At the end of this phase, the developer must compare the total runtime of each task (local and cloudlet) in each context, then

label with a "Yes" value those ones whose total runtime is shorter, and with a "No" value those whose total runtime is longer.

The next step (iii) covers the classifiers evaluation process. We developed a Java program[2] to automate the training and testing of classification algorithms. This program measures the performance of each classifier by means of its accuracy and others metrics (for more details see Section 4.3.1) in the test data by using 30 repetitions of a 10-fold cross-validation and varying the seed value in the range from 1 to 30. The results must be analyzed and compared by suitable statistical techniques such as a confusion matrix and performance metrics (e.g., specificity, sensitivity, precision, and accuracy).

Finally, in the last step (iv) the developer must generate the trained classifiers' models corresponding to those that obtained the highest accuracy. In addition, the files corresponding to the generated models must be saved in the CSOS project to allow the decision engine to load them at runtime. The next activity is to configure the *remotable* markup with a *dynamic* value in the methods identified as an offloading candidate. After that, the developer must specify which classifier to use on the same markup (for more details see section 5.3). All classifiers to be used by the developer must be declared in an enumeration class. Thus, each classifier is translated into the respective generated model.

After following the four steps of the development process, the mobile application is ready to use CSOS to enable the context-sensitive offloading of their methods/data.

4.2.2 Implementation Details

Next, we describe the technical details of how CSOS operates the decision engine, the *remotable* markup, and the transformation from the raw context to the high-level context.

In Algorithm 4, we present the pseudocode of the decision engine developed to handle the context data and classify the most recent instance in the database. The *isRemoteAdvantage* procedure (line 1) receives as arguments the application's data size and classifier name, respectively. Naturally, the data size metric must be captured by the app profiler. However, the value of this metric can only be known at the instant of time that the application's user selects the desired image resolution for processing. Since the profiling system runs every 35 seconds, it would be impracticable to accurately capture the value of this metric. Therefore, the proxy handler captures this value at runtime and passes it to the decision engine. Between lines 3 and 5, we check whether the object of the classification model corresponds to the specified classifier. If false, it instantiates a new object of the requested classification algorithm. Next, we check each attribute of the most recent record in the database to collect information, such as name, type, and possible values (between lines 6 and 9).

[2] An automation program for training and testing using a 10-fold cross-validation is available to the community at the following website: https://github.com/ehammo/algorithmCompare

Algorithm 4 Procedure for offloading decision with classifiers

```
1: procedure isRemoteAdvantage(InputSize, Classifier)
2:     response ← false
3:     if classifierModel ≠ Classifier ∨ classifierModel = Null then
4:         loadClassifier(Classifier)
5:     end if
6:     for all attribute A ∈ Database do
7:         Attributes[] atts ← getAttribute(A)
8:         Values[] values ← getValue(A)
9:     end for
10:    Instance ← createInstance(atts, atts.qtde)
11:    for i ← 0, atts.qtde do
12:        if atts.getName[i] = 'DataSize' then
13:            Instance.setValue[i] ← InputSize
14:        else
15:            Instance.setValue[i] ← values[i]
16:        end if
17:    end for
18:    result ← ClassifyInstance(Classifier, Instance)
19:    if result ≥ 0.7 then
20:        response ← true
21:        return response
22:    else
23:        return response
24:    end if
25: end procedure
```

To enable classification from the Weka library, the instance object needs to be created (line 10) along with the set of attributes and their quantity. After that, the instance receives the value corresponding to the input size if the attribute name is equal to 'DataSize'; otherwise it receives the other values (lines 11-17). *ClassifyInstance* (line 18) classifies an instance with probabilistic values. Thus, when the instance is rated above 70%, the procedure returns a "true" value, indicating that offloading is favorable; otherwise it returns a "false" value, indicating that it is unfavorable. The rate of 70% refers to probability for the "Yes" class, corresponding to remote processing. We define this threshold to reduce the impact on application execution due to a wrong decision by the algorithm, since in this case the application will be executed locally.

The following code examples illustrate the development of a face detection application that has two methods: *detectFaces()* and *getFaces()*. Firstly, we created an enumeration to list the possible classifiers to be used by the decision engine, and a variable to save the classifier that is going to be used to decide if this method is going to be offloaded or not. Second, we modified the *@remotable* markup to receive the J48 classifier (line 2 in Listing 4.1). When the *detectFaces* method execution (line 3) is intercepted by the proxy handler, the decision engine will decide whether the method must be executed locally or

outside of the mobile device, based on the J48 classifier.

When the CSOS is running, the user of the benchmarking application can choose which classifier to use. Therefore, our solution supports one interface for each classifier to allow the proxy handler to interpret the classifier specified by the user through markups (or Java annotation).

```
1 public interface DynamicDetectFacesJ48 extends DetectFaces {
2     @Remotable(value=Remotable.Offload.DYNAMIC, status=true, classifier=
          Remotable.Classifier.J48)
3     PropertiesFace detectFaces(String cascadeClassifier, byte[]
          originalImage);
4      [...]
5 }
```

Listing 4.1 – Android markup code for specifing the classifier.

To accurately decide whether to offload or not, we need profilers. We therefore made a task to run every 35 seconds, as a network profiler takes a long time to measure precisely the upload and download rate. This task gathers low-level context information or raw context, converts it to high-level context, and then saves it on a context database. The following is an example of this conversion. We can see in Listing 4.2 that the *getCPULabel* method receives the raw value of CPU usage in a percentage (line 1). If the o value is between 45 and 75, the variable "ret" receives the value "Normal_Load", indicating that the CPU is processing normally (lines 5 and 6).

```
1  public ResultTypes.ResultTypesCpu getCPULabel(float total) {
2   ResultTypes.ResultTypesCpu ret;
3   if (total < 45) {
4    ret = ResultTypes.ResultTypesCpu.Relax;
5   } else if (total >= 45 && total < 75) {
6    ret = ResultTypes.ResultTypesCpu.Normal_Load;
7   } else if(total == (-1)){
8    ret = ResultTypes.ResultTypesCpu.Unknown;
9   } else{
10   ret = ResultTypes.ResultTypesCpu.Stressed;
11  }
12  return ret;
13 }
```

Listing 4.2 – Transformation of low-level context to high-level context.

Table 5 shows the mapping of each low-level context to a high-level one. We performed several exploratory tests to define the range of values (low-level context) for the attributes of numbers 1,3, and 5; while the range of values of the attributes corresponding to the numbers 2 and 4 were the results of research by the authors. For instance, to define thresholds with respect to RAM memory and the clock speed of the *Phone(Hdw)*

attribute, we used a library that analyzes an Android device's specifications (RAM, CPU cores, and clock speed). This allowed the authors to modify its behavior based on the capabilities of the smartphone's hardware.

Regarding the *Data size* attribute, more specifically for the BenchImage and BenchFace applications, we calculated the size in Kilobyte (KB) of each picture and converted it to a megapixel (MP) unit. As can be seen in Table 5, we mapped the values 708KB to 2MP, 1186KB to 4MP, and 4413KB to 8MP. The calculation for CollisionBalls is a bit different. Since each ball has 44KB, our strategy is to calculate this value by the amount of balls. So, we mapped the first value of 649KB to 250 balls, 1938KB to 750 balls, and so on.

Table 5 – Attributes and contextual values.

No	Attribute name	Low-level context	High-level context
1	Bdw: Bandwidth	[up/down>20] [2<up/down<=20] [up/down<=2]	Free Moderate Congested
2	App/Data size	BenchImage [708-1186-4413] BenchFace [2075-3758-4717] CollisionBalls [649-1938-2583]	2MP-4MP-8MP 3MP-6MP-8MP 250-750-1500 Balls
3	Phone(CPU)	[CPU>=75] [40<CPU<=74] [CPU<=40]	Stressed Normal Load Relaxed
4	Phone(Hdw)	[2.3<RAM<=3 and FREQ<=1.8] [1.5<RAM<=2.3] [RAM<=1 and FREQ<1.3]	Advanced-intermediate Intermediate Weak
5	Cloud(vCPU)	[CPU>=75] [40<CPU<74] [CPU<=40]	Stressed Normal Load Relaxed

4.3 Performance analysis of classifiers

In this section, we present the evaluation and the experiment using CSOS. We evaluate and analyze two different aspects of the solution: (i) the performance analysis of the classification algorithms in section 4.3.2; (ii) mobile application runtime and energy consumption (lab testbed) in section 4.4.

4.3.1 Evaluation Setup

For this evaluation, four classification algorithms (C4.5, Rules, K-NN, and Naive Bayes) were compared to each other, considering the main evaluation metrics. The (Java) implementation of C4.5 in Weka (WITTEN et al., 2016) is referred to as J48, while the K-NN and Rules-Based are respectively referred to as IBK and JRIP. We have used these classifiers because they require less resource (e.g., processing, storage) than artificial neural networks, and are fairly accurate (PERERA et al., 2014). These features are relevant to the offloading decision in order to discover hidden knowledge in our own context database. In this regard, we benefited from the Weka library to develop a Java program and thus evaluate the algorithms.

A 10-fold cross-validation has been made for testing and evaluation of the results. For our purpose, the database is divided into two groups: training and testing, where 90% is for training and 10% is for testing. We have run each algorithm 30 times, varying the seed value in the range from 1 to 30, in order to obtain a sample of 120 results, which have been averaged for the data set. Thus the results, which will be further analyzed by the statistical techniques, correspond to the average accuracies in test data.

To investigate the performance of the four classification algorithms more accurately, we used the confusion matrix that can be seen in Table 6. With this matrix the amount of each indicator is calculated and then results are compared. The confusion matrix is a useful tool for analyzing how well a classifier can recognize multiples of classes different. The ideal situation is when the most relevant data are on the main diameter matrix and the rest matrix values are zero or near zero (ALIZADEH; GHAZANFARI; TEIMORPOUR, 2011).

Table 6 – Confusion matrix in our study.

Actual	**Predicted**	
	Offloading (Positive)	**No-Offloading (Negative)**
Positive	TP	FN
Negative	FP	TN

Various evaluation metrics - such as true negative rate (Specificity), true positive rate (Recall or Sensitivity), False Positive Rate (FPR), False Negative Rate (FNR), Precision and Accuracy (ACC) of assessment categories - are calculated according to the formulas (4.1)-(4.7). Each one is described below.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

$$F_1 = \frac{2TP}{(2TP + FP + FN)} \tag{4.2}$$

$$Sensitivity = TPR = \frac{TP}{TP + FN} \tag{4.3}$$

$$Specificity = TNR = \frac{TN}{TN + FP} \tag{4.4}$$

$$Precision = \frac{TP}{TP + FP} \tag{4.5}$$

$$FPR = \frac{FP}{FP + TN} = 1 - TNR \tag{4.6}$$

$$FNR = \frac{FN}{FN + TP} = 1 - TPR \tag{4.7}$$

Where:

TP = the number of positive examples correctly classified.
TN = the number of negative examples correctly classified.
FP = the number of positive examples misclassified as negative.
FN = the number of negative examples misclassified as positive.

The *accuracy* of a classifier for a given test dataset is indicated by the percentage of the test dataset records that are correctly classified by the classifier. The *sensitivity* (or *recall*) measures the fraction of positive examples correctly predicted by the classifier, while *specificity* is the proportion of negative records that are correctly identified. *Precision* determines the fraction of records that actually turn out to be positive in the group where the classifier has been declared to be a positive class. *Sensitivity* and *precision* are summarized into another metric known as the F_1 measure (see formula (4.2)). The classification error (4.6) is referred to as FPR, the proportion of negative records that are not correctly identified. The classification error (4.7) is referred to as FNR, the proportion of positive records that are not correctly identified (WENG; HUANG; HAN, 2016).

4.3.2 Classifiers Performance

Before evaluating the aforementioned classifiers, we have chosen the best K to K-NN algorithm (IBK), i.e. the K value with the most expressive *accuracy*. We decided to start with the value *K=1* and increase it up to the total number of records in our database (300 records). To do that, we have tested all possible combinations between *Ks* and *seeds*, then we have made an average of the accuracy to each K value with each *seed* aiming to choose the combination to obtain the best accuracy. The result is show in Figure 13. The best K is equal to 5, with *seed* equal to 12, and an *accuracy* rate of 96.35%.

By using the confusion matrix and the above formulas, the values of the aforementioned metrics for four algorithms can be specified. The results with the average are shown in Table 7. Figure 14b shows the comparative analysis of four classifiers in terms of specificity, sensitivity, precision, FPR, FNR and F1. From the graph, we can observe that the *specificity* is highest and similar for IBK and J48 i.e. 93.45% and 93.43% respectively,

and lowest for Naive Bayes with 89.27%. The *sensitivity* is maximum for JRIP with a rate of 98.08% and minimum for Naive Bayes at 93.89%. In summary, JRIP was the algorithm that most correctly predicted positive records (offloading favorable) as well as outperforming J48 by 0.74%. The *precision* with the J48 algorithm was 94.69%, slightly higher than IBK with a difference of 0.03%, and 0.48% compared to JRIP.

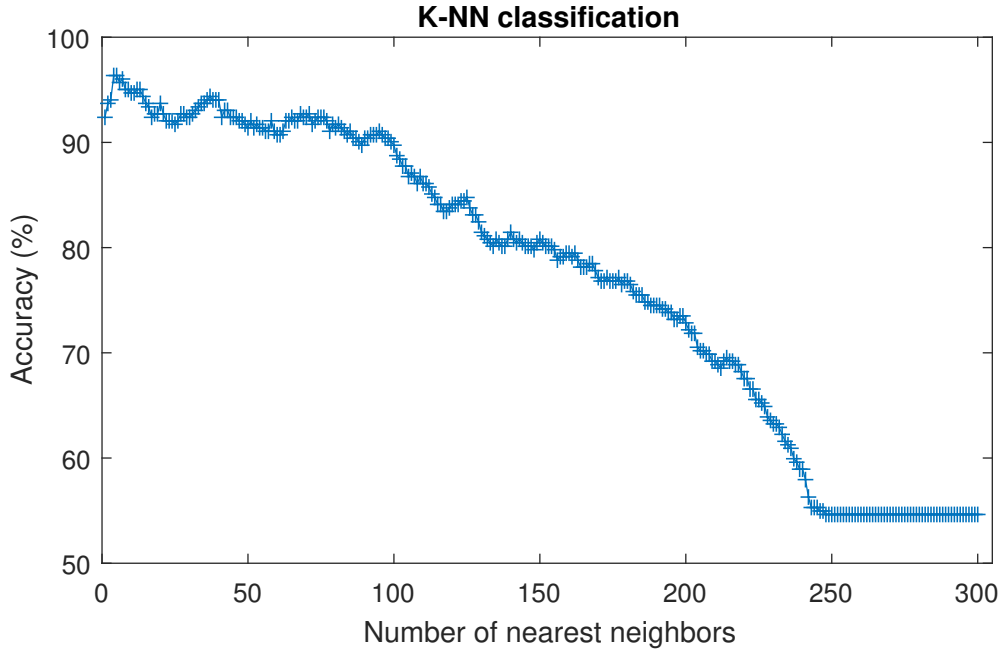


Figure 13 – Accuracy for different values of K.

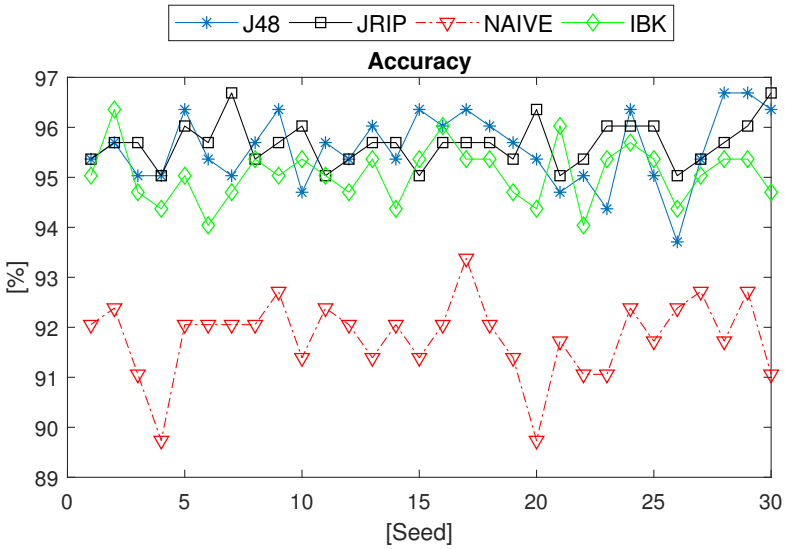
The FPR and FNR are referred to as the types of errors. Therefore, the first type of error is more important to mobile cloud solutions for determining the drawbacks of code offloading than the second one. According to Table 7, IBK and J48 have lower FPR (6.54% and 6.56%) compared to JRIP and Naive Bayes algorithms, while FNR is lowest for JRIP with a difference of 27.65% compared to J48 and 46.79% compared to IBK.

Table 7 – Average of each measured metric for algorithms (%).

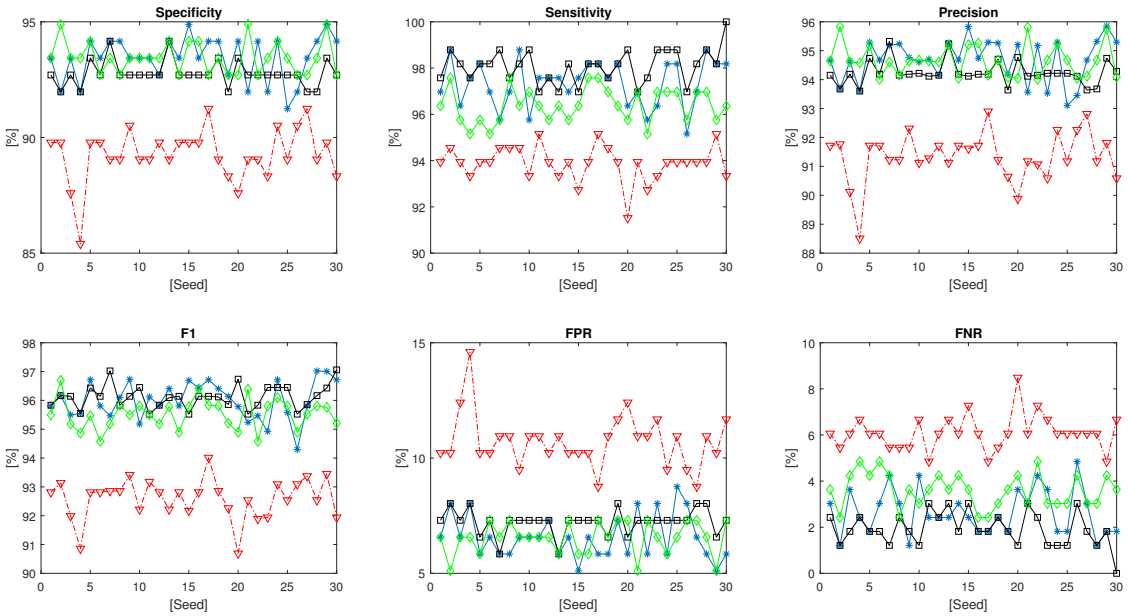
Algorithm	**Specificity**	**Sensitivity**	**Precision**	**FPR**	**FNR**	**F1**	**ACC**
Naive Bayes	89.27	93.89	91.33	10.72	6.10	92.59	91.79
IBK	93.45	96.40	94.66	6.54	3.59	95.52	95.06
J48	93.43	97.35	94.69	6.56	2.64	96.00	95.57
JRIP	92.77	98.08	94.23	7.22	1.91	96.11	95.67

Sensitivity and *precision* are two metrics widely employed in applications where the successful detection of one of the classes is considered more significant than that of the other classes. Unfortunately, they are in conflict with each other due trade-off between them, since if we want retrieve more relevant records (i.e., for increasing the sensitivity rate), consequently more non-relevant records will be retrieved as well (i.e., it will decrease

the precision rate). Therefore, F_1 is proposed to be means to achieve harmony between sensitivity and precision (WITTEN et al., 2016). Figure 14b clearly shows that the JRIP and J48 techniques have F_1 greater than the IBK and Naive Bayes with rates of 96.11% and 96.00%, respectively.



(a) Accuracy.



(b) Specificity, sensitivity, precision, F1, FPR, and FNR.

Figure 14 – Comparison between the classifiers using different indicators.

Figure 14a shows the comparative analysis in terms of accuracy. In MCC the high accuracy is very important due the adaptive and dynamic nature of mobile systems, which leads to an inaccurate decision that in turn leads to high energy consumption and degrades performance. Our results show that the rules generated by the JRIP algorithm have

an accuracy slightly higher compared to J48 and IBK amongst the contextual dataset. The accuracy of the JRIP algorithm is 95.67% comparable to the J48 algorithm, which achieves 95.57%. The Naive Bayes had the worst accuracy with 91.79% of records correctly classified.

The good classifiers noted from the above results are JRIP, J48, and IBK. It can be seen that in all cases, the Naive Bayes algorithm has shown to have the worst performance over our context database.

According to our evaluation results, the J48, JRIP, and IBK classifiers had acceptable and similar performance. To determine whether the differences between these algorithms in terms of accuracy are significant, we use the Friedman test with a confidence interval of 95% (FRIEDMAN, 1937; FRIEDMAN, 1940). The Friedman test is a non-parametric equivalent of the repeated-measures ANOVA (DEMŠAR, 2006). It ranks the algorithms for each cross-validation fold separately, with the top algorithm receiving the rank of 1, the second best receiving the rank of 2, and so on. Thus, the worst performing algorithm receives a rank equal to the number of algorithms (in our case 4). Average ranks are assigned in case of ties. The Friedman statistic is defined as:

$$X_F^2 = \frac{12N}{k(K+1)}\left[\sum_j R_j^2 \frac{k(k+1)^2}{4}\right] \tag{4.8}$$

where N is the number of folds (10 in our case), k is the number of algorithms (4 in our case), and R_j is the average rank of the jth of k algorithms. The average rank is defined as $R_j - \frac{1}{N}\sum_i^N r_i^j$ where r_i^j is the rank of the jth of k algorithms on the ith of N folds.

When the null hypothesis (all classifiers are equivalent) of the Friedman test is rejected we perform the Nemenyi post-hoc test to determine which classifiers are significantly different (NEMENYI, 1963). The performance of two classifiers is significantly different if the corresponding average ranks differ by at least the critical difference:

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}} \tag{4.9}$$

where the critical values q_α are based on the Studentized range statistic divided by $\sqrt{2}$, N is the number of folds and k is the number of algorithms to be compared (DEMŠAR, 2006). If the difference between the mean rankings of 2 classifiers is bigger than the Critical Difference (CD) of 0.8563, then the performance of these algorithms differs significantly.

Figure 15 shows the results for Multiple Comparison with Best (MCB) statistical test. The "Ha:Different" label means that at least one algorithm is different. Consequently, the null hypothesis was rejected, indicating that there is a significant difference between the accuracy of the classifiers. Thus the Nemenyi post-hoc test was performed to outline the algorithms rankings.

Table 8 displays the mean ranking of the classification algorithms along with the critical difference to clearly show any algorithms that are significantly different. It is

possible to note that the J48 has the best overall ranking position with 1.68. However, the difference between J48 and JRIP is less than the CD value (equivalent to 0.02), indicating that theses classifiers are not statistically different. On the other hand, both J48 and JRIP are significantly different from the IBK with 0.94 and 0.92, respectively. In a nutshell, the Friedman and Nemenyi statistical tests show that the J48 and JRIP classifiers outperform the IBK and Naive Bayes classifiers from our context database. Once that these classification algorithms obtained similar performance results, we implemented both in CSOS and used them in our real-world experiments (for more details see section 4.4).

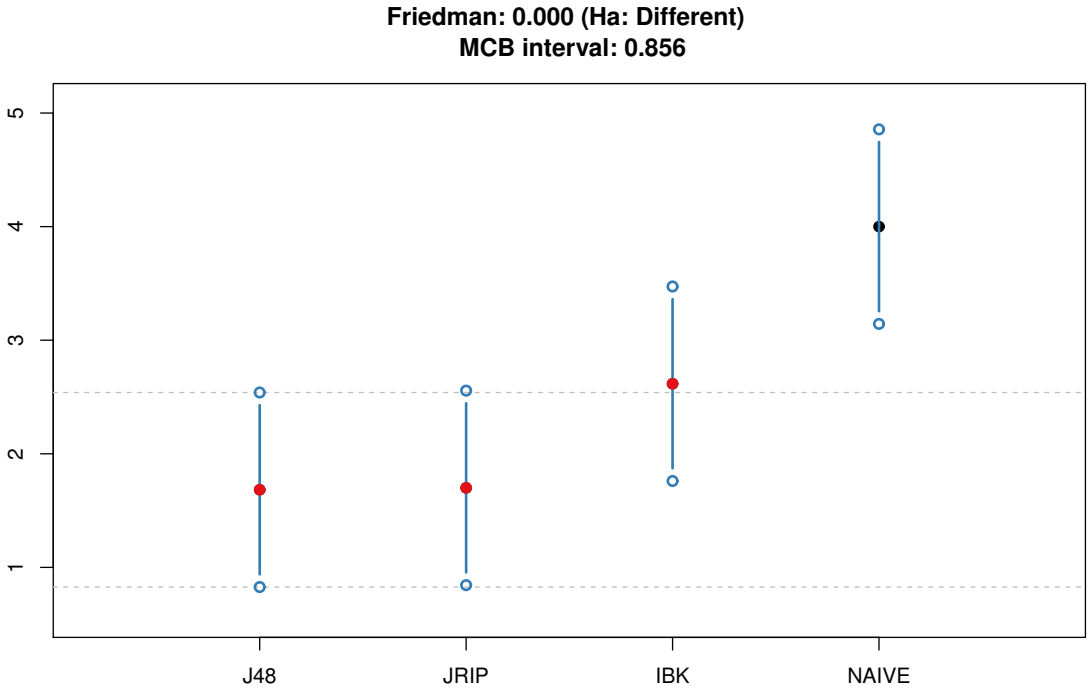


Figure 15 – Friedman and Nemenyi tests for comparisons between algorithms.

One of the key points in the rules generated by J48 and JRIP algorithms is that the rules are very simple. Therefore these rules can be used very easily by resource-constrained mobile devices, a critical issue in the MCC environment. The J48 algorithm uses a statistical property derived from information theory, called the information gain, that measures how well a given attribute separates the training instances according to their target classification.

Table 8 – Mean ranking of each classifier.

Prediction algorithm					
	J48	**JRIP**	**IBK**	**NAIVE**	**Nemenyi critical distance**
Position	1	2	3	4	-
Value	1.68	1.70	2.62	4.00	0.8563

Therefore, according to Figure 16, predictor importance for bandwidth, application, smartphone's hardware, smartphone's CPU, and cloud's vCPU are 0.4348073, 0.1538575, 0.0256408, 0.0060328 and 0.0000928, respectively. These data reveal how much the wireless network quality impacts on offloading operations.

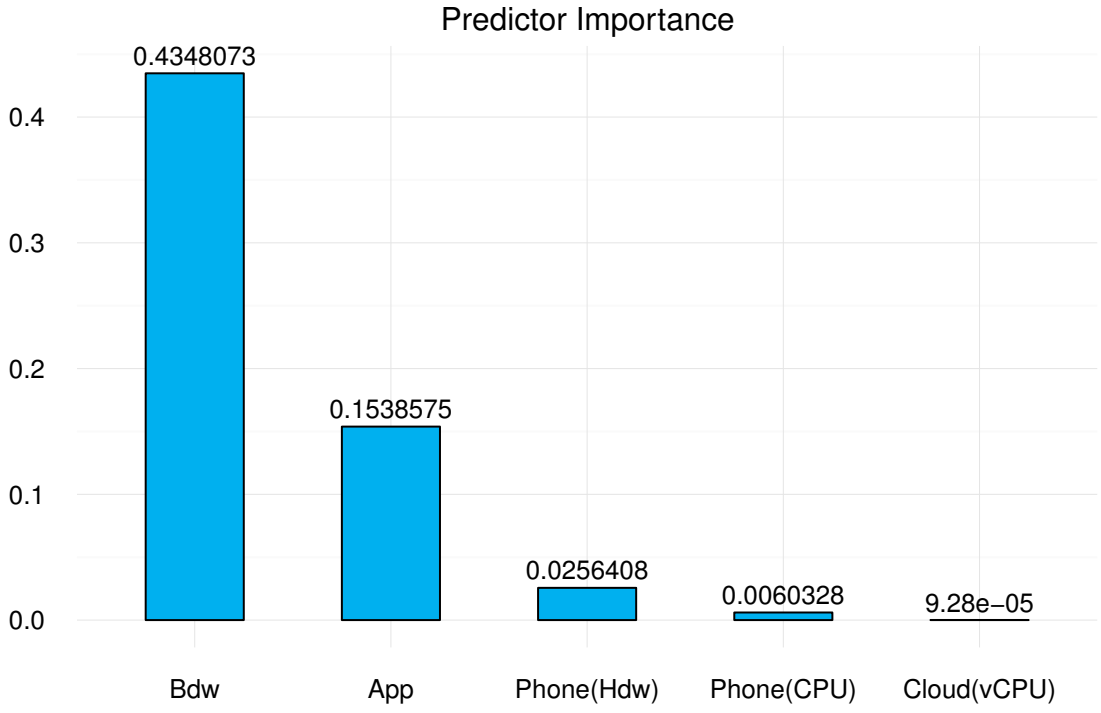


Figure 16 – Importance of the factors in the prediction of offloading by using info gain.

The rules generated by this algorithm are shown in Table 9. The depth of trees produced by this algorithm was 19, which is very different compared to the JRIP algorithm. It is also clear that nine rules have been generated for class *No* (class *No* is devoted to 'no offloading') and ten rules for class *Yes* (class *Yes* is devoted to 'do offloading'). The rules generated by the J48 algorithm shows that this algorithm is suitable for the offloading decisions, due to the balance in the class values (Yes/No).

According to Table 10, it can be seen that eight rules have been produced by the JRIP algorithm. It is also clear that seven rules have been generated for class *No* and only one rule for class *Yes*. With regard to the results and some rules, it can be deduced that JRIP can make incorrect decisions due to the omission of some rules for class *Yes*. Generally, by comparing our results and the Tables, it can be said that J48 and JRIP are more successful in identifying the right time to undertake offloading. But it should be noted that the rules produced by J48 have provided more detail about both classes.

Table 9 – Rules generated by the J48 algorithm.

Num	**Rules**
1	IF Bdw = Free AND App = BenchFace THEN class Yes
2	IF Bdw = Free AND App = BenchImage THEN class Yes
3	IF Bdw = Free AND App = CollisionBalls AND Phone(Hdw) = Adv. Interm. THEN class No
4	IF Bdw = Free AND App = CollisionBalls AND Phone(Hdw) = Weak AND DataSize <= 1186 THEN class No
5	IF Bdw = Free AND App = CollisionBalls AND Phone(Hdw) = Weak AND DataSize >1186 THEN class Yes
6	IF Bdw = Free AND App = CollisionBalls AND Phone(Hdw) = Intermediate THEN class No
7	IF Bdw = Mdt AND App = BenchFace AND Cloudlet(vCPU) = Relaxed THEN class Yes
8	IF Bdw = Mdt AND App = BenchFace AND Cloudlet(vCPU) = Stressed AND Phone(Hdw) = Adv. Interm. AND Phone(CPU) = Relaxed THEN class No
9	IF Bdw = Mdt AND App = BenchFace AND Cloudlet(vCPU) = Stressed AND Phone(Hdw) = Adv. Interm. AND Phone(CPU) = Normal Load THEN class Yes
10	IF Bdw = Mdt AND App = BenchFace AND Cloudlet(vCPU) = Stressed AND Phone(Hdw) = Adv. Interm. AND Phone(CPU) = Stressed THEN class Yes
11	IF Bdw = Mdt AND App = BenchFace AND Cloudlet(vCPU) = Stressed AND Phone(Hdw) = Weak THEN class Yes
12	IF Bdw = Mdt AND App = BenchFace AND Cloudlet(vCPU) = Stressed AND Phone(Hdw) = Intermediate THEN class Yes
13	IF Bdw = Mdt AND App = BenchImage THEN class Yes
14	IF Bdw = Mdt AND App = CollisionBalls AND Phone(Hdw) = Adv. Interm. THEN class No
15	IF Bdw = Mdt AND App = CollisionBalls AND Phone(Hdw) = Weak AND DataSize <= 1186 THEN class No
16	IF Bdw = Mdt AND App = CollisionBalls AND Phone(Hdw) = Weak AND DataSize >1186 AND DataSize <= 2583 THEN class Yes
17	IF Bdw = Mdt AND App = CollisionBalls AND Phone(Hdw) = Weak AND DataSize >1186 AND DataSize >2583 THEN class No
18	IF Bdw = Mdt AND App = CollisionBalls AND Phone(Hdw) = Intermediate THEN class No
19	IF Bdw = Cong THEN class No

The rules in Tables 9 and 10 are simple and understandable for researchers and developers, meaning that these findings can be useful as an appropriate solution to identify an opportunist offloading event in the real environment. In other words, by using the results of this study, more effective rules for beneficial computational offloading can be achieved.

Table 10 – Rules generated by the JRIP algorithm.

Num	**Rules**
1	IF Bdw = Cong THEN class No
2	IF App = CollisionBalls AND Phone(Hdw) = Adv. Interm. THEN class No
3	IF App = CollisionBalls AND DataSize <= 649 THEN class No
4	IF App = CollisionBalls AND DataSize >= 3871 AND Bdw = Mdt AND Phone(CPU) = Stressed THEN class No
5	IF Bdw = Mdt AND Cloudlet(vCPU) = Stressed AND DataSize >= 4717 AND Phone(Hdw) = Adv. Interm. THEN class No
6	IF App = CollisionBalls AND Cloudlet(vCPU) = Stressed AND DataSize >= 3871 AND Phone(CPU) = Relaxed THEN class No
7	IF Bdw = Mdt AND App = BenchFace AND Phone(CPU) = Relaxed AND Phone(Hdw) = Adv. Interm. AND Cloudlet(vCPU) = Stressed AND DataSize <= 3758 THEN class No
8	ELSE class Yes

4.4 Evaluation and Validation

In this section, we conducted several experiments to evaluate the two classifiers with better performance: JRIP and J48. Section 4.4.1 describes the experimental environment, while the other sections (4.4.2, 4.4.3, 4.4.4) discuss the results.

4.4.1 Experimental Setup

Table 11 shows all the parameters used in the experiment. We benchmarked the three different computation-intensive applications, named BenchFace, BenchImage and CollisionBalls.

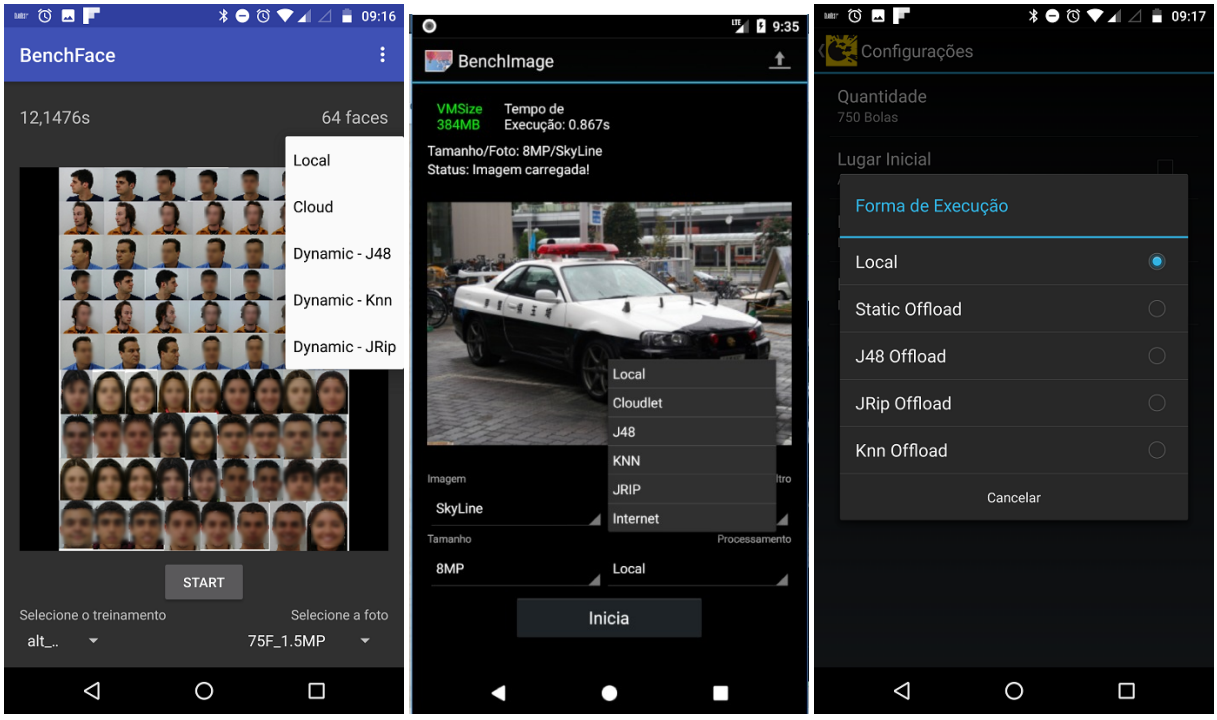
The BenchFace[3] application was developed by the author for face detection and uses *Haar features* based on cascading classifiers, a method proposed by the researchers in (VIOLA; JONES, 2001). The algorithm for face detection uses a ML approach. It trains cascade functions with a set of positive images (images containing faces) and negative images (images that do not have faces). The application consists of a single image with 78

[3] Android implementation of BenchFace, available in https://github.com/ehammo/HideTheFaceAndroid.

faces at different angles (see Figure 17a). The user can change the same image to different resolutions and cascading classifier algorithms.

According to (REGO et al., 2015), BenchImage is an image processing application that allows users to apply filters on pictures with different resolutions (see Figure 17b). The application provides the filters Sharpen, Cartoonizer, and Red Tone, which have different computation requirements and therefore different execution times. In addition, BenchImage provides an option to execute a benchmark procedure, in which the Cartoonizer filter is executed for each picture resolution (8MP, 4MP, and 2MP).

Rego et al. (REGO et al., 2015) define CollisionBalls[4] as an application that simulates several balls bouncing around the screen of the mobile device. The application detects when the balls touch the edge of the screen or when the balls touch each other, and then calculates the new direction of the balls (see Figure 17c). The amount of balls can be defined by the user as well as the type of serialization used: built-in Java and C# serialization or manual serialization. It is a real-time application.



(a) BenchFace application. (b) BenchImage application. (c) CollisionBalls application.

Figure 17 – Benchmark applications.

For BenchFace and BenchImage we varied the data size every 10 tasks. But we didn't vary the CollisionBall's data size, since the amount of balls couldn't be changed dynamically. The other big difference is in the amount of decisions. If the real time application asked the decision engine about the context after every new frame, it would be

[4] Android implementation of BenchImage and CollisionBalls, available in https://github.com/ufc-great/mpos/tree/master/android.

very detrimental to the application, since this delay would lower the frame per second rate. So, the decision was made once every 10 seconds, and its result will apply to all marked methods until the next decision.

Regarding smartphone power consumption metering, this work will focus on software-based power measurement. Trepn Profiler is an Android application that can display the real-time power consumption on a smartphone or tablet. According to (HOQUE et al., 2015), it is the only application that reports accurate real-time power consumption.

Table 11 – Parameters used in the context-sensitive experiment.

Apps	**Parameters**	**Value**
BenchFace	Detection algorithm	Alt_tree
	Size (MP)	3, 6, and 8
	Number of faces	77
	Tasks number	30
BenchImage	Filter	Cartoonizer
	Size (MP)	2, 4, and 8
	Image	SkyLine
	Tasks number	30
CollisionBalls	Serialization	Java built-in
	Size (Kb)	1938
	Number of balls	750
	Tasks number	10

We used the equipments described in Table 12. We have used three smartphones from different manufacturers and with different hardware. In our experiment, the Moto X Style, Galaxy S4, and LG K3 are mapped to advanced intermediate, intermediate, and weak smartphones, respectively. The communication between mobile device and cloudlet was performed using a wireless network (devices directly connected to the WiFi access point attached to the cloudlet), with one TP-LINK AC1200 (802.11ac) access point.

Table 12 – Technical specification of the devices in the context-sensitive experiment.

Units, equipment, and platform	**CPU (GHz)**	**RAM (GB)**
1, Moto X Style, Android	Hexa-core 1.6	3
1, Samsung Galaxy S4, Android	Quad-core 1.9	2
1, LG K3, Android	Quad-core 1.1	1
1, Laptop cloudlet, Openstack Kilo (Ubuntu 14.04 LTS)	Quad-core 2.7	8

In contexts where the bandwidth must be *moderate* (Mdt) or *congested* (Cong), we generate UDP background traffic from the Iperf (IPERF, 2017) tool for 2000 seconds and

report the result every 3 seconds. Besides that, we configure the CpuRun (CPURUN, 2017) and CpuBurn (CPUBURN, 2017) tools to utilize all available cores of the smartphone's CPU and cloudlet's vCPU respectively, aiming to achieve the context information, *stressed* and *normal load* for our lab testbed. To properly test we first picked which contexts are going to be used in the tests. As shown in Table 13 we picked five favorable contexts, where the cloudlet is going to be a better option than local execution, and five unfavorable contexts where the opposite is true. Last, we selected five unknown (Unkn) contexts, which means that these contexts represent new instances of the our database that have not participated in the training and testing process during the performance evaluation of the classification algorithms.

Table 13 – Context dataset for the experiments.

Decis.	**Id**	**Bdw**	**App**	**Cloud(vCPU)**	**Phone(CPU)**	**Phone(Hdw)**
Yes	C1	Free	BenchImage	Relaxed	Relaxed	Adv. Interm.
	C2	Free	BenchFace	Relaxed	Relaxed	Adv. Interm.
	C3	Mdt	BenchFace	Relaxed	Stressed	Adv. Interm.
	C4	Mdt	BenchFace	Stressed	Relaxed	Weak
	C5	Free	CollisionBalls	Relaxed	Normal Load	Weak
No	C6	Free	CollisionBalls	Relaxed	Relaxed	Adv. Interm.
	C7	Cong	BenchImage	Relaxed	Normal Load	Weak
	C8	Cong	BenchImage	Relaxed	Stressed	Intermediate
	C9	Mdt	BenchFace	Stressed	Relaxed	Adv. Interm.
	C10	Mdt	CollisionBalls	Relaxed	Relaxed	Adv. Interm.
Unkn	C11	Mdt	BenchFace	Stressed	Normal Load	Adv. Interm.
	C12	Mdt	BenchFace	Stressed	Relaxed	Intermediate
	C13	Mdt	BenchFace	Normal Load	Relaxed	Adv. Interm.
	C14	Free	CollisionBalls	Relaxed	Normal Load	Intermediate
	C15	Mdt	CollisionBalls	Relaxed	Normal Load	Intermediate

The objective of the experiments' related favorable and unfavorable contexts is to evaluate the performance of the proposed CSOS and the classifiers implemented upon it, and upon prior knowledge that our system must decide correctly. On the other hand, the aim of the experiments with related unknown context, is to know if our system can decide correctly or not in real time, and what the consequences are in terms of performance and energy.

All experiments utilized four strategies:

- Dynamic J48 (labeled as **J48**): this provides the entire CSOS system, such as the decision engine, the Weka library, discovery/deployment services and profilers. The

application relies on CSOS's decision engine to decide where to process offloading candidates methods (locally or remotely) based on the J48 classifier.

- Dynamic JRIP (labeled as **JRIP**): this provides the entire CSOS system, i.e. it includes the decision engine, Weka library, discovery/deployment services and profilers. The application relies on CSOS's decision engine to decide where to process offloading candidates methods (locally or remotely) based on the JRIP classifier.
- Static Cloudlet (labeled as **Cloudlet**): this provides the partial CSOS system, i.e. it includes discovery/deployment services and profilers. All applications are executed remotely on the cloudlet server to acquire a performance baseline.
- Static Local (labeled as **Local**): this does not use the CSOS system, and all processing is done on the smartphone, i.e. all applications must be executed locally to acquire a performance baseline.

Finally, to enhance the reader's comprehension, only 9 contexts are going to be plotted instead of all 15. From each situation we obtained 3 graphs, showing runtime, energy and context. The runtime graph represents the amount of time spent on each task, while the energy graph represents how much energy the smartphone consumed over the whole 30 tasks. The context graph shows the frequency of values for the current context's attributes. This configuration remains true for the BenchFace and BenchImage applications, but for CollisionBalls there is a little difference since it is a real-time application. The runtime graph is exchanged for a Frame Per Second (FPS) graph, which represents the amount of FPS the application has at the moment of the decision.

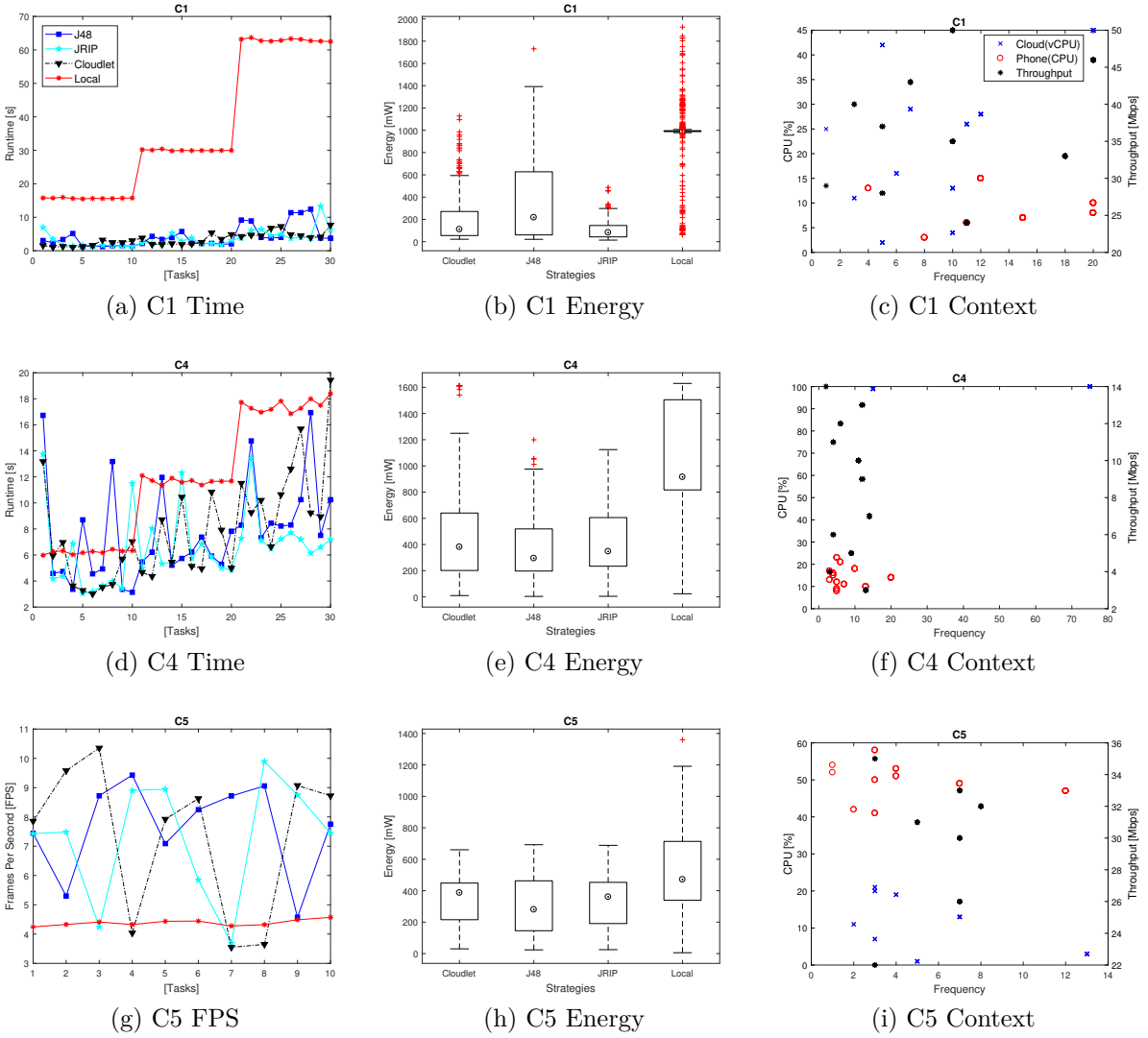
4.4.2 Results - Favorable Context

Figure 18 (a)-(i) shows the results of the three set of favorable contexts chosen: C1, C4, and C5. Each line on the graph shows a different metric for the same context, e.g. for the context labeled C1, we show the results corresponding to the runtime, consumed energy, and frequency of the context data (smartphone's CPU, cloudlet's vCPU, and throughput). With regard to the context graph, it is possible to identify the CPU usage by smartphone and cloudlet (both on the left y-axis). The network profiling between smartphone and cloudlet through the throughput (right y-axis) is also depicted.

Specifically for the BenchImage and BenchFace applications, the runtime and energy metrics were measured during the execution of a total of 30 tasks (see x-axis). The smartphone executed the first 10 tasks with the same image size (e.g., 2MP), and the next 10 tasks for a higher image size (e.g., 4MP), and so on.

Figure 18a presents the runtime in seconds for C1 context with the four strategies: Local, J48, JRIP, and Cloudlet. The results using the last three show that the runtime remained mostly below 13 seconds, even with the variation of the images size, indicating

that the device did not perform substantial processing, since the dynamic strategies, J48 and JRip, had a similar result to the cloudlet strategy. Therefore, the result using the local strategy presented runtime equal to 15 seconds with 2MP, 30 seconds with 4MP, and 62 seconds with 8MP, due local processing capacity being less than the cloudlet's.



(a) C1 Time (b) C1 Energy (c) C1 Context

(d) C4 Time (e) C4 Energy (f) C4 Context

(g) C5 FPS (h) C5 Energy (i) C5 Context

Figure 18 – Results with favorable offloading: C1, C4, and C5.

Figure 18b illustrates the distribution of the energy consumption in Miliwatts (mW) of the BenchImage application on Moto X Style corresponding to the C1 context, while Table 14 offers a comparison of the mean, median, minimum, and maximum for each context. The results indicate that the local strategy drains more energy than all other strategies, which means that processing efforts of the Moto X Style are higher due to the overhead caused by the intensive processing of the filters in the images. The energy consumption corresponding to the JRIP strategy is 47.80% and 71.46% more efficient than the cloudlet and J48 strategies, respectively. The J48 consumed more energy than cloudlet and JRIP because it has a greater number of rules regarding JRIP, and thus

demands a longer processing time for decision-making.

Table 14 – Comparative energy consumption to contexts C1, C4, and C5.

Context	Strategy	Min.	1st. Qu.	Median	Mean	3rd Qu.	Max.
C1	J48	20.99	60.84	217.70	392.20	626.20	1731
	JRIP	14.03	42.77	82.40	111.90	144.50	487.30
	Cloudlet	21.64	55.16	110.50	214.40	270.40	1128
	Local	62.39	988.30	990.90	978.10	996.20	1926
C4	J48	3.779	197.80	294.70	374.90	518.40	1199
	JRIP	4.638	235.60	347.70	413.50	604.50	1124
	Cloudlet	10.22	201.70	381.10	467.00	640.20	1611
	Local	24.06	818.70	915.30	1068	1505	1630
C5	J48	23.64	147.20	280.00	319.40	459.10	692.50
	JRIP	24.92	195.60	360.60	328.00	451.10	688.40
	Cloudlet	30.07	216.80	386.70	343.40	447.10	660.30
	Local	5.496	340.70	471.20	534	704.50	1360

The main difference between the C4 and C1 contexts are the cloudlet's vCPU, the smartphone's hardware, the network bandwidth, and the benchmarking application (see Table 13). The results of Figure 18d, corresponding to the J48, JRIP, and cloudlet strategies, show that the runtime suffers large variations as the image size increases, due to values assigned to the C4 context (see Figure 18f), but remained below 6 seconds with a 3MP image, 12 seconds with 6MP, and 17 seconds with 8MP, while the local strategy peaks up to 18.40 seconds in the runtime values with 8MP images. Figure 18e presents the distribution of the energy consumption for the same context. It is notable that the local strategy drains 56.27%, 61.28%, and 64.89% more energy than cloudlet, JRIP and J48 strategies, respectively, while the three last had similar energy drains.

Figure 18g shows the strategies performance by the FPS metric, since the C5 context makes use of CollisionBalls, a real-time application. The value of this metric depends on the time spent in calculating the new position of the balls. According to (HUGHES; FOLEY, 2014), a graphic computing application should run above 30 FPS, i.e. taking a maximum of 33.34 milliseconds (ms) to produce each frame, so that animation can run seamlessly from the application user's perspective. The results indicate that for the 750 balls scenario the local execution presents a lower FPS than computational offloading using any strategy, which means that given the amount of computation needed a resource-poor smartphone requires the use of offloading. Besides that, for the JRIP, J48 and cloudlet strategies, the offloading improves CollisionBalls' performance by 1.65, 1.74, and 1.67 times, respectively. This explains the fact that CSOS - even using decision-making and running profilers before offloading to cloudlet - does not impair the FPS quality, because the lowest values of FPS

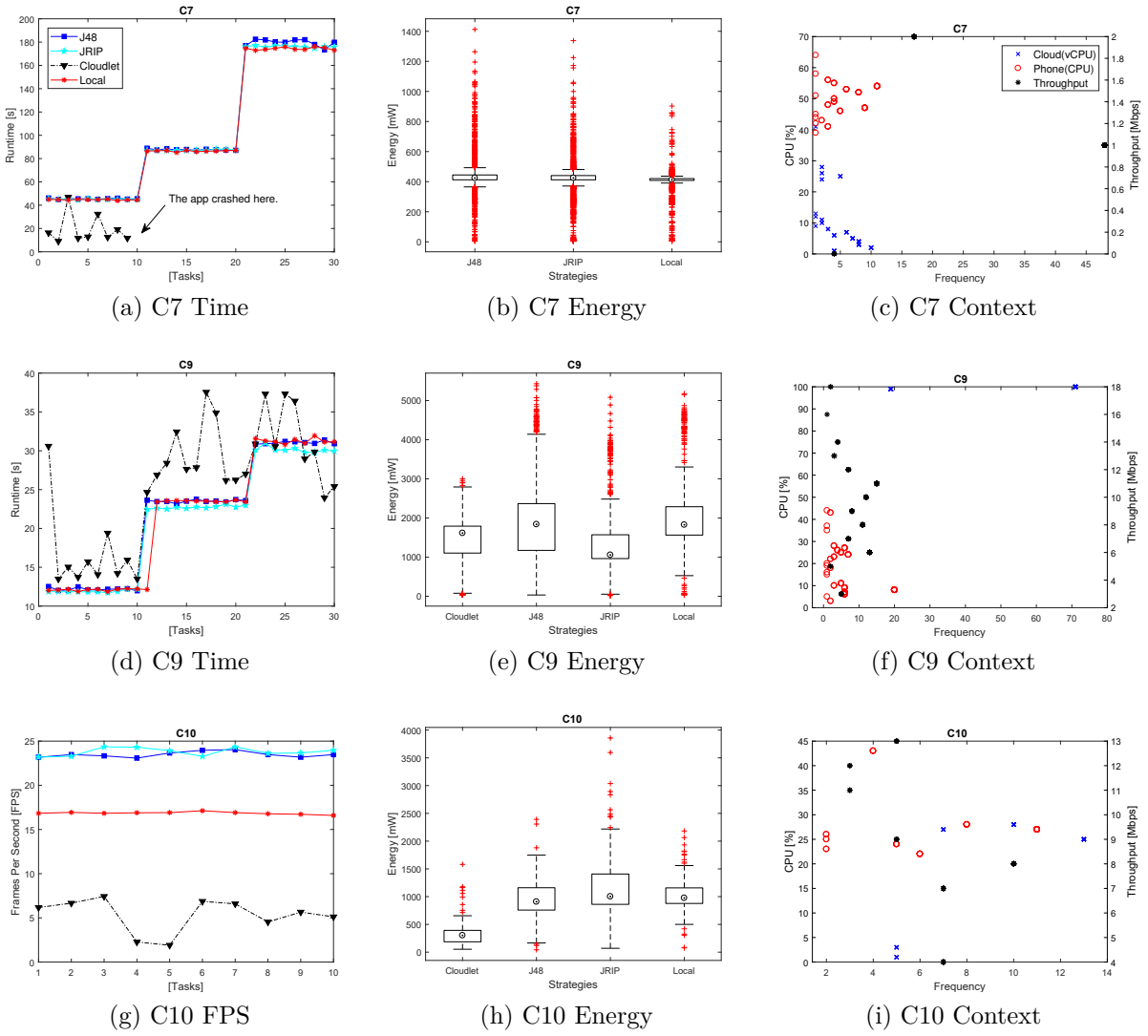
with CSOS are similar to those in the local strategy (see 3 and 9 tasks). Regarding the results of Figure 18h we can see that local processing provides the largest battery discharge when compared to other strategies. This is due to the low processing capacity of the LG K3 (categorized as weak hardware) to perform a lot of computing locally.

4.4.3 Results - Unfavorable context

Figure 19 (a)-(i) shows the results of the three sets of unfavorable contexts chosen: C7, C9, and C10. Regarding Figure 19a, the results show a mean runtime equal to 45 seconds with 2MP, 87 seconds with 4MP, and 176 seconds with 8MP, indicating that the device performed substantial processing, once the dynamic strategies, J48 and JRIP, had a similar result to the local strategy. On the other hand, the cloudlet strategy couldn't be completed, because the network was so congested that the connection between smartphone and cloudlet was lost, causing the application to crash. Thus, the result using the cloudlet strategy was worse. Figure 19b has similar results to the three strategies that executed locally and presented a high number of outliers (lower and upper), justified by the high energy drain. Cloudlet didn't have any energy results to show since it couldn't reach the 30 tasks.

According to Figure 19f, the cloudlet's vCPU in a stressed state has a direct influence on the runtime when we use the cloudlet strategy (see Figure 19d), since it will always run the application remotely, even when wireless network quality is not good. Consequently, the runtime with cloudlet is approximately 1.37, 1.22, and 1.02 times greater than the other strategies for 2MP, 4MP, and 8MP images, respectively. It is important to highlight the last four tasks for runtime corresponding to cloudlet with values below the other strategies. This situation occurs because the download and upload rate of the background traffic reaches the value near the maximum threshold (e.g., 18 Mbps) for a moderate bandwidth. This is why the runtime considers network fluctuations when an application's methods are executed remotely.

Figure 19g presents the FPS for the C10 context with the four strategies: J48, JRIP, cloudlet, and local. The results for cloudlet were the worst in this context. The wireless network in a moderate state decreases the FPS quality of the application (see Figure 19i). The mean FPS in the cloudlet was 68.48% and 77.67% lower than the local strategy and dynamic strategies, respectively. The J48 and JRIP strategies maintained good FPS quality by deciding that the best choice was to process locally. As we can see in Figure 19h, the cloudlet strategy saved approximately 70.85% of the mobile device's energy when compared to JRIP. In this context, even with the cloudlet spending less of a device's energy, it had a loss of FPS quality with an average of 5.31 frames per second.



(a) C7 Time (b) C7 Energy (c) C7 Context

(d) C9 Time (e) C9 Energy (f) C9 Context

(g) C10 FPS (h) C10 Energy (i) C10 Context

Figure 19 – Results with unfavorable offloading: C7, C9, and C10.

Table 15 presents the mean and median values related to energy consumption for the four strategies. In the C9 context, mean energy reveals low power consumption by the JRIP strategy, despite concentrating a high number of outliers (see Figure 19e). It consumes 1344 milliwatts while the cloudlet strategy consumes 1438 milliwatts, a difference of 6.53%, which means that both processing and connectivity efforts of the Moto X Style are higher due to the overhead caused by background traffic (in the C9 context the bandwidth must be moderate) and the offloading operations. In contrast, the energy draining with J48 strategy is 1.36 times higher than the cloudlet. In general, the JRIP and J48 strategies classify the C9 context with the '*No*' value, i.e. it is better to run it locally. As a result, CSOS ensures a lower runtime as well as delivering an energy drain similar to the cloudlet strategy.

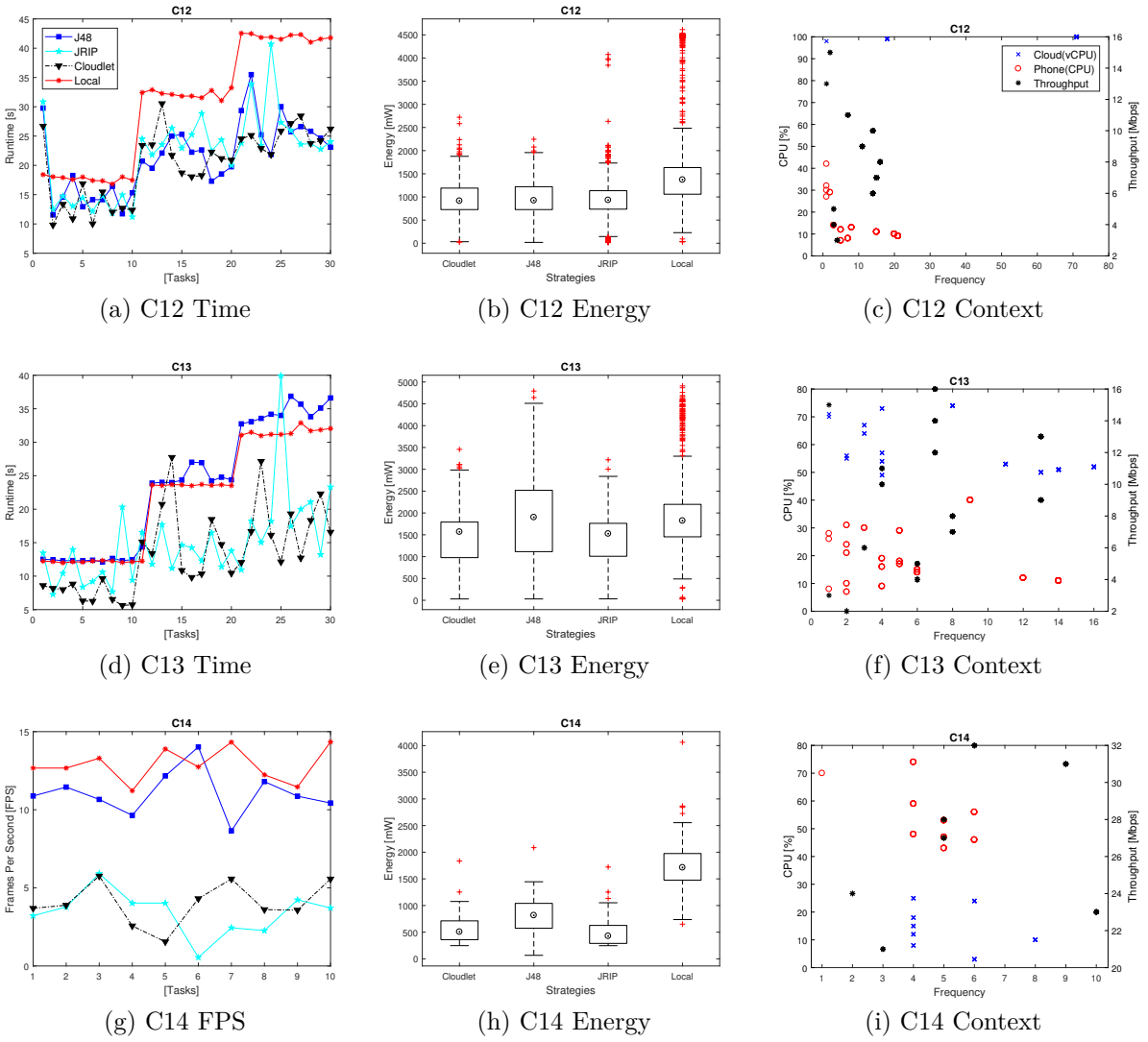
Table 15 – Comparative energy consumption to contexts C7, C9, and C10.

Context	Strategy	Min.	1st. Qu.	Median	Mean	3rd Qu.	Max.
C7	J48	4.208	412.40	424.50	437.60	444.60	1413
	JRIP	4.208	412.40	424.50	417.80	440.60	1338
	Cloudlet	-	-	-	-	-	-
	Local	5.067	408.40	412.40	409.10	420.40	903.20
C9	J48	33.19	1175	1838	1956	2364	5430
	JRIP	20.12	963.50	1056	1344	1571	5083
	Cloudlet	31.44	1106	1615	1438	1791	3003
	Local	31.44	1564	1826	2001	2284	5176
C10	J48	42.15	760.80	908.30	960.30	1151	2392
	JRIP	69.76	864.40	1003	1210	1387	3857
	Cloudlet	52.46	187.60	302.10	352.60	389.90	1581
	Local	76.69	879	975.70	1040	1158	2182

4.4.4 Results - Unknown context

Figure 20 (a)-(i) shows the results of the three set of unknown contexts chosen: C12, C13, and C14. Figure 20a presents the runtime in seconds for the C12 context with the four strategies: local, J48, JRIP and cloudlet. We can see that the two algorithms classified the C12 context correctly, since the offloading operations reduced the runtime by 29.24% for both J48 and JRIP strategies, even when the cloudlet's vCPU was stressed, i.e. 100% vCPU usage with a frequency of 71 times in Figure 20c. Such a performance gain of the cloudlet server compared to intermediate smartphone (corresponding to Galaxy S4) may be explained by the increase of computational overhead on the smartphone in terms of the runtime of the BenchFace application. Figure 20b illustrates the distribution of the energy consumption for C12 context. The results show that the energy consumption with cloudlet, JRIP and J48 strategies is 40.34% lower than when using the local strategy, since they require less computational effort, which leads the smartphone to save energy.

According to Figure 20d the results show that by using computation offloading with the JRIP strategy, it is possible to reduce method execution time by up to 1.57 times when compared to J48. Additionally, we can see that energy consumption using a JRIP strategy is 30% lower than using J48 and 29.43% lower than a local strategy (see Fig 20e). In a nutshell, the best choice is to offload, i.e. JRIP classified the C13 context correctly, while the J48 classified it incorrectly.



(a) C12 Time (b) C12 Energy (c) C12 Context

(d) C13 Time (e) C13 Energy (f) C13 Context

(g) C14 FPS (h) C14 Energy (i) C14 Context

Figure 20 – Results with unknown offloading: C12, C13, and C14.

The Figure 20g presents the FPS during CollisionBalls' execution for the C14 Context. The results show that for the 750 balls scenario the local execution related J48 strategy presents a higher FPS than offloading to a cloudlet server using the JRIP strategy. For instance, the local execution runs with approximately 69.14% more FPS than the cloudlet server. Such a result indicates that depending on the context (more specifically the smartphone's hardware), computational offloading cannot improve CollisionBalls' performance. Regarding the energy, Figure 20h and Table 16 show that the JRIP strategy saves 38.96% more energy than the J48 strategy by using offloading operations with bandwidth free and cloudlet's vCPU relaxed (see Table 13). Even when the application is executed locally using a J48 strategy, the smartphone consumes 2.12 times less energy than the local strategy. This situation is justified because the workload generated to stress the CPU reaches a value near the minimum threshold (i.e. 45%) for smartphone's CPU attribute mapped as the normal load (see Figure 20i).

Table 16 – Comparative energy consumption to contexts C12, C13, and C14

Context	Strategy	Min.	1st. Qu.	Median	Mean	3rd Qu.	Max.
C12	J48	18.61	727.30	923.90	950	1220	2246
	JRIP	14.26	736.60	932	936	1135	4076
	Cloudlet	25.58	728.30	915.30	931.30	1191	2720
	Local	30.21	1059	1371	1574	1634	4615
C13	J48	30.57	1115	1901	1984	2518	4793
	JRIP	32.32	1009	1525	1388	1762	3218
	Cloudlet	30.57	977.30	1568	1407	1790	3457
	Local	30.57	1454	1823	1967	2196	4910
C14	J48	65.92	576.80	816.40	821.60	1040	2086
	JRIP	246.8	292.0	427.6	501.5	626.4	1723
	Cloudlet	247.6	363.5	506.6	555.4	712.1	1835
	Local	650.3	1482	1713	1748	1971	4062

In summary, the set of experiments to unknown contexts provided insights to the benefits of using the offloading technique only when beneficial with the J48 and JRIP strategies of the CSOS. The J48 classifier in particular is a promising solution to handle different application categories, since the rules produced by it are more detailed and the set of classes are balanced.

These results show that the proposed CSOS outperforms the baseline in all cases. In some contexts (e.g., C4 and C12 contexts), the CSOS achieves performance and energy efficiency at the same time with a fair tradeoff between required objectives. Nonetheless, there are contexts (e.g., C7 and C10 contexts) where CSOS achieves a single objective and may sacrifice energy efficiency or performance.

4.5 Summary

This chapter presented CSOS, a system able to perform context-sensitive computational offloading that uses machine-learning classifiers to ensure high accuracy in offloading decisions. The proposed solution relies on our context database for the training and testing of four classification algorithms previously selected. These are J48, JRIP, IBK and Naive Bayes, of which two are chosen for implementation in CSOS - J48 and JRIP - since they had the best performance over our database.

In order to evaluate the proposed solution, we developed a decision engine that adopts the J48 and JRIP classifiers for decision-making and profilers that transforms raw context elements to high-level context information at runtime. We then conducted experiments using three benchmarking applications that perform intensive computing. The first

experiment compared the runtime, FPS, and energy consumption with the CSOS enabled and disabled for favorable contexts, where the remote execution will be a better option than local execution. The second experiment is similar to the first, with the difference that it uses unfavorable contexts, where the opposite is true. The last experiment is similar to previous two, with the difference that it uses unknown contexts, i.e. the author does not know where the execution of code/data will require less computational effort (remote or local).

The results show that CSOS proves to be more effective compared to related works and baselines. CSOS achieved an accuracy of 95%, which in turn helped in taking the right offloading decision, thus saving energy consumption and improving runtime.

5 MOBILITY-AWARE OFFLOADING

This chapter presents the Mobile Context-Sensitive Offloading System (mCSOS) architecture as well as the interaction between its components (section 5.1). In addition, we present a step-by-step signaling overview (section 5.2) and describe the implementation details (section 5.3). Finally, the system is evaluated in the test-bed scenario with three different experimental environments (section 5.4).

5.1 Design Goals and Architecture

By applying a set of mobility-specific extensions, the CSOS becomes "mobility CSOS" or mCSOS. These extensions are related to the network controller that consists of a mobility management application, able to maintain the device's IP network address and track their location, and a remote caching technique to speeds up the offloading response time between PoAs.

The mCSOS is divided into three layers (see Figure 21). The 'mobile environment' layer consists of constrained-resource devices (e.g., smartphones and smartwatches), whether in processing or energy autonomy. When running heavy processing due to some demanding computational power application, those devices may not perform as expected, harming the user experience.

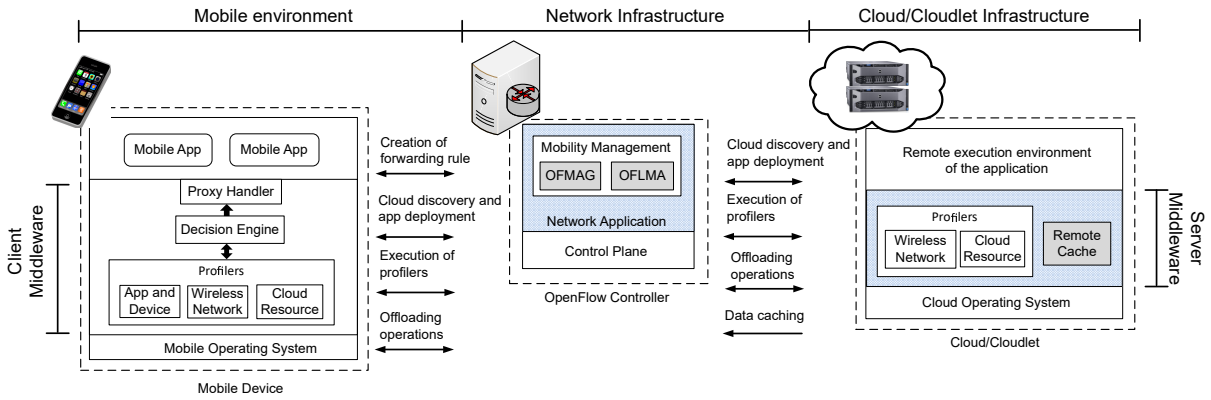


Figure 21 – mCSOS Architecture.

The 'network infrastructure' layer consists of network elements (e.g., switch, router, access point). Our mobility management application that run on top of the OpenFlow controller, programmatically controls these elements. It keeps handover execution transparent to the offloading operations. Hence, the device state is maintained regardless the cloud-side running service. For instance, the acquired IP address in the first wireless router by the mobile device is not changed, even if it switches to a nearby wireless network. Thus, it avoids network layer reconfiguration. The OpenFlow controller can be deployed

on low-cost commodity hardware and provides a global view of the network elements. Lastly, the 'cloud infrastructure' layer represents public cloud providers (e.g., Amazon AWS, RackSpace, Windows Azure) with its Service Level Agreement (SLA) and cloudlet servers that contain, for instance, a virtualized environment with VM.

There are two types of interactions regarding the proposed architecture. The first set relates to the communication between the mobile device and OpenFlow controller. When the mobile device is within a wireless coverage, it requests connection to the AP and such action, together with the controller, creates routing rules to assist the packet forwarding process towards cloudlet/cloud resources. Then, the middleware on mobile device side executes the cloud discovery and application deployment services. The objective of the discovery service is to discover a remote execution environment of the application that is running on the same wireless network that the mobile device is connected to (e.g., cloudlet). On the other hand, the deployment service is responsible to deploy application services on the cloud resource. When the discovery service does not detect an application service for a specific application running on the cloud, the deployment service is used to send all dependencies (e.g., binaries, libs, data) from the mobile application to the cloud in order to start the application service for such application. These services use UDP-based multicast in order to advertise mobile device presence within the same wireless coverage where the cloud resource is located. After the association between the mobile device and cloud resource, a profiling service (application, device, network, and cloud) captures context data at runtime to assist the decision engine that decides if the annotated method should be offloaded or not, as described in the previous chapter.

The second interaction entities are the OpenFlow controller and cloud resource. In the discovery and deployment services, middleware server listens to the announcement of the mobile device on the wireless network and responds indicating that it is available to receive requests. As mentioned, if there is no corresponding service to the application, it will be deployed. The profiler service monitors and collects data from the network and cloud, while the remote execution environment of the application (on cloud-side) receives code/data from the mobile device, then it execute them, and sends the result to mobile device continue the execution flow of the application. At last, the unidirectional interaction is the data caching, considering the previous processing at the same cloud resource, the remote caching technique only returns the processing result to the requester mobile device.

Since all the components of the mobile device and cloud/cloudlet were described in the previous chapter (see Chapter 4, Section 4.1), we will describe only the network controller components, as well as remote cache component.

- **Remote Caching:** It is responsible for storing and retrieving application data received by cloud resource. After data is processed, a copy is temporarily stored in the cache, and if an application requests the same data, the cache returns the

result of the offloaded execution, thus avoiding extra processing time in the cloud and speeding up response time.

- **OFMAG:** The OpenFlow Mobility Access Gateway (OFMAG) represents the wireless routers, which aims to monitor the user's mobility and to request an IPv4 prefix to the network controller. If any mobile device comes out of its coverage, it must notify the OFLMA by informing the occurrence of this event. The OFMAG sends a Proxy Binding Update (PBU) message to the OFLMA, to associate its address with a mobile device identification, e.g., the Media Access Control (MAC). Then it receives a Proxy Binding Ack (PBA) message from OFLMA, which contains network prefix to be allocated to the mobile device.

- **OFLMA:** The OpenFlow Local Mobility Anchor (OFLMA) represents the routers or switches. OFLMA has functions such as: creating rules on OpenFlow switches to meet the mobility needs of OFMAG's clients, receiving and handling PBU messages, and sending PBA messages to the mobile device. Besides, it manages the OFMAGs, i.e., controls all traffic coming out of the mobile device and arriving at the cloud resource. Furthermore, it maintains data structures that allow knowing if the user is moving between different OFMAGs (handover) and creates a Binding Cache Entry (BCE) to register device's information (e.g., device identifier).

Both OFMAG and OFLMA components were inspired by the Proxy Mobile IPv6 (PMIPv6) protocol (GUNDAVELLI et al., 2008). It allows the mobile device to be exempt from any signaling during mobility session while other entities, such as MAG and LMA, undertakes the signaling on behalf of the mobile device.

5.2 Signaling Details

This section presents the mCSOS's signaling, as well as a more detailed description regarding the messaging between its components. The two signaling processes will be described below and are only possible with the use of the infrastructure composed by OFMAGs, OFLMA, OpenFlow Controller (in Figure 22 and Figure 23 is represented by the acronym OF-C), and cloudlet server. The signaling procedure described in Figure 24 utilizes conventional network infrastructure and cloudlet server.

5.2.1 Signaling with Mobility Management

Figure 22 shows a scenario where the mobile device is connecting to the OFMAG1 router, then obtaining an IP address, and with the established connection, identifies the presence of a cloudlet server, hence performing the cloud discovery and application deployment.

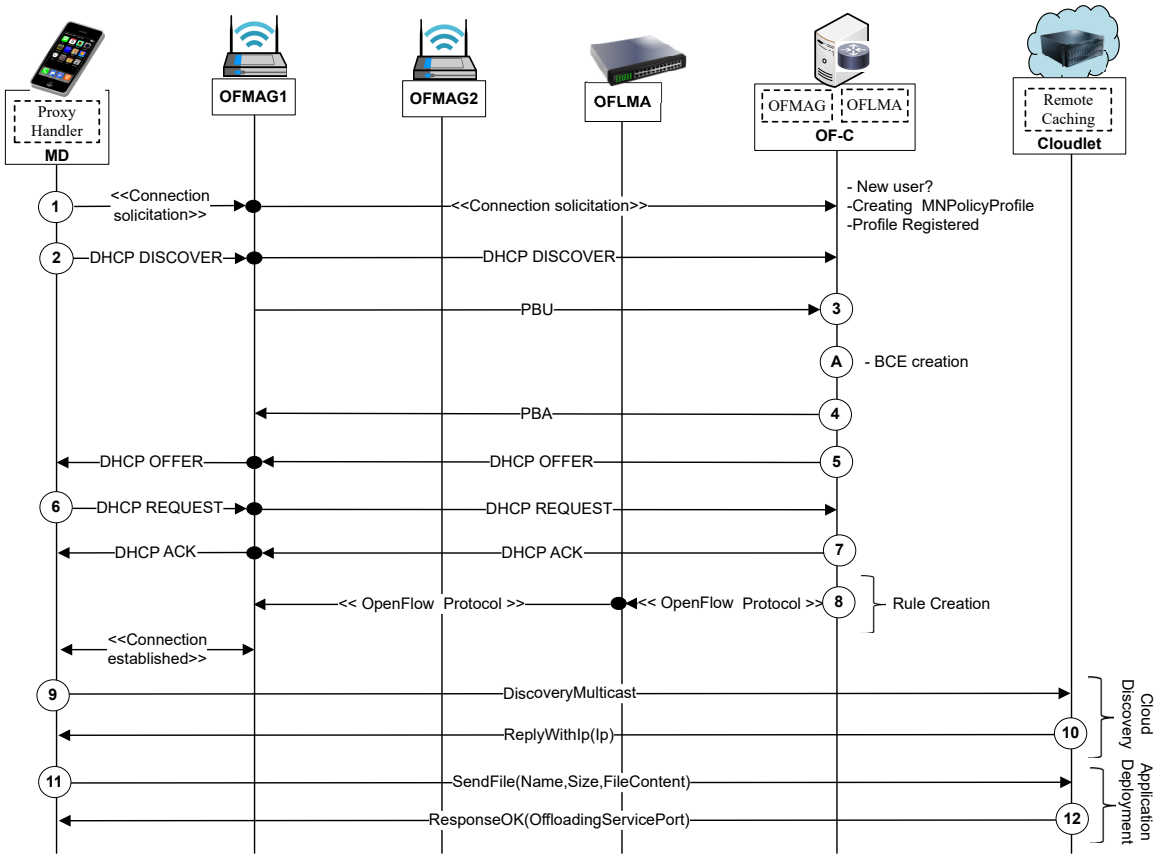


Figure 22 – Connection signaling with mobility management.

The Mobile Device (MD) initiates the admission process requesting a connection to OFMAG1, which immediately informs the OF-C to register the MD's profile on the *MN-PolicyProfile* ①. The MD requests an IP address by sending a DHCP DISCOVER packet ②. OFMAG1 forwards the request to the OF-C and sends a PBU message requesting the assignment of an address interface ③. At the OF-C (instance that represents the OFLMA class), an BCE is created for registering MD information (Event A). The OF-C responds with a PBA message creating forwarding rules for the OFMAG1 router for further defining the new path for the following mobile device packets in order to receive an IP address ④. The OF-C also has the DHCP server and thereby offers an IP address to the MD ⑤, which responds to accept the offered IP address ⑥, and the OF-C confirms the IP allocation, concluding the IP assignment process ⑦. Then, specific routing and forwarding rules to the IP and MD's interface are created in the OFLMA and OFMAG1 ⑧. Upon connection, the cloudlet discovery service starts. Differently, the MD's discovery service announces its presence using User Datagram Protocol (UDP) multicast ⑨. After detecting this announcement, the same service on the cloudlet-side responds by sending its IP through the UDP unicast ⑩. As this is the first access to the cloudlet, the corresponding service for the application does not exist yet. Therefore, the service sends all dependencies (binary files and libraries) following the pattern of the first file: name, file size, and finally its content ⑪. After receiving all the files, the service on cloudlet-side

sends the port number where the application service is running (12). From this moment on, the *proxy handler* can trigger the offloading operations.

The signaling scenario described in Figure 23 depicts offloading operation (to the cloudlet server) being performed by the *proxy handler* through OFMAG1 when the MD moves into the OFMAG2's coverage area, enabling handover and then receiving the processing result.

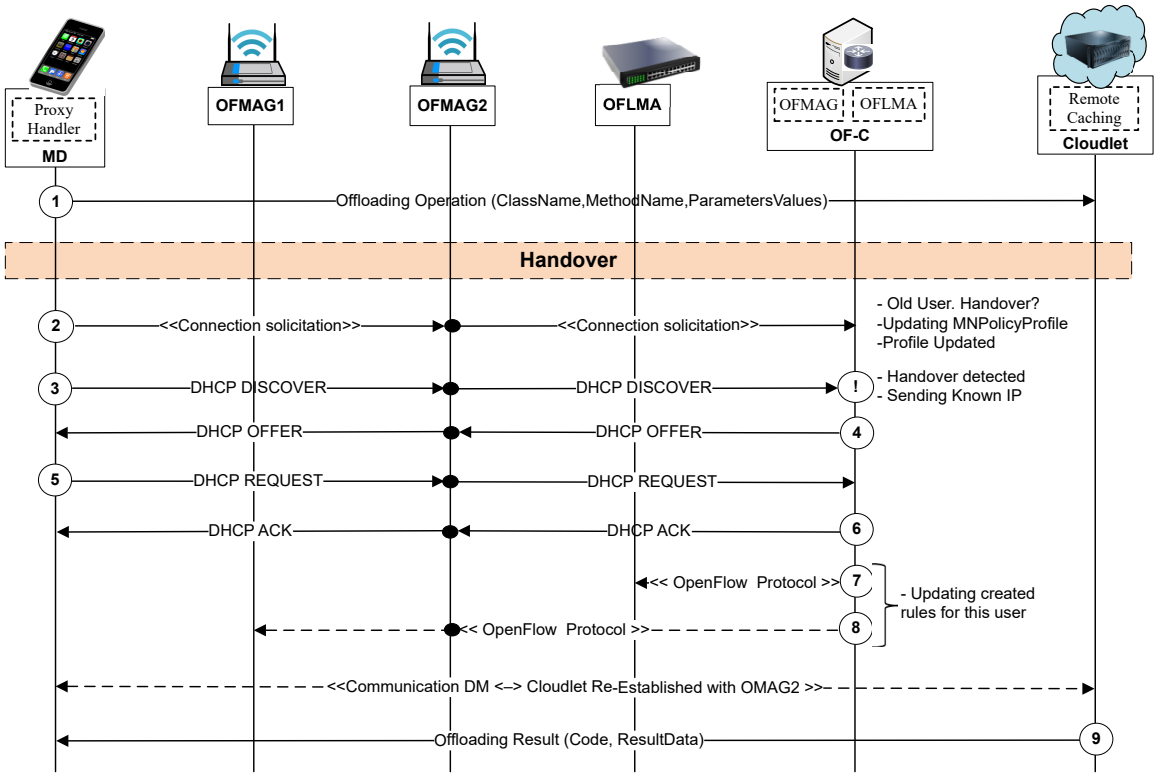


Figure 23 – Handover signaling with mobility management.

Once the connection is established and the middleware services finished, the *proxy handler* triggers offloading operations by sending the class and method names, as well as the parameter values for remote processing, being followed by a handover operation (1). Due to network change, the MD requests a new connection to the OFMAG2 router, which redirects to the OF-C. Upon receiving the request, OF-C checks the registered profiles and detects a handover. Therefore, it updates the *MNPolicyProfile* with the new MD location (2). The MD sends an IP request to the DHCP server (3) and therefore the OF-C becomes aware of the handover, providing the same device's IP (4). The MD accepts the offered IP, and the OF-C confirms, finishing IP address assignment (5) (6). The OFLMA receives the new rules to update the device's location and defines new routes (7). Then, the OF-C configures the new forwarding rules on OFMAG1 and OFMAG2 using OpenFlow protocol, redirecting the traffic to the new MD's location (8). With the reestablished communication, the remote execution environment of the application returns the result to the MD (9).

5.2.2 Signaling with Remote Caching

This scenario aims to illustrate unavailability of an SDN network infrastructure controlled by a mobility management application, leaving the remote caching responsibility to speed up delivery of the results to the mobile device when it suffers handover. It is similar to the previous handover situation, differing only by the OFMAGs, OFLMA and OF-C infrastructures, which are not utilized. On the contrary, only the device's proxy handler and cloudlet server's remote caching inside a default network infrastructure are implemented.

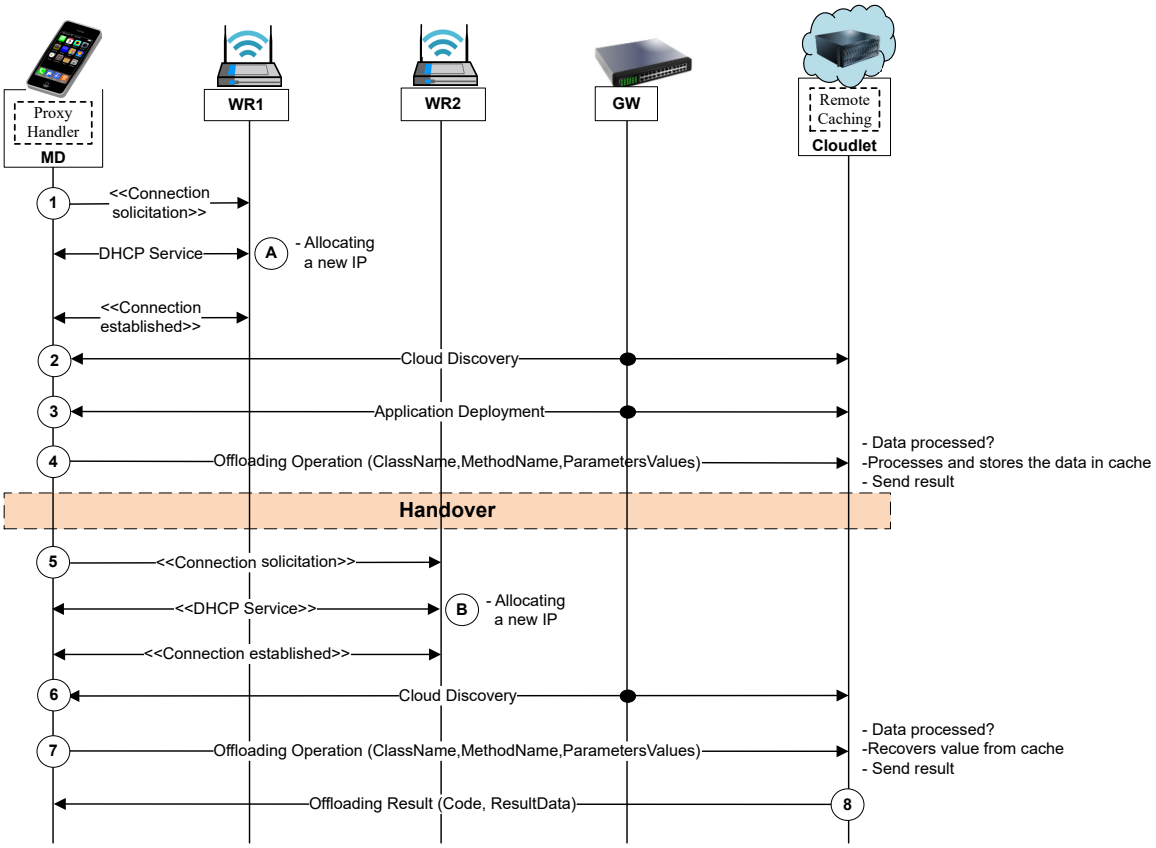


Figure 24 – Handover signaling with remote caching.

In Figure 24, the MD requests the establishment of a new connection with the wireless router 1 (WR1) ①. Next, the MD requests an IP address to the DHCP server in WR1, which offers a new IP address to the MD (Event A). The device accepts the address while WR1 confirms its reservation. Upon connection, the discovery and deployment services are executed and finalized ② ③. The offloading operations for cloudlet are then triggered ④. The remote caching logic checks if the requested data has been processed. Since this is the first entry, the cloudlet processes, stores a copy of the resulting data in a cache and returns the processed data to the MD. However, before returning a result, a handover occurs. The cloudlet will try to deliver the result by up to three times, without success. After the handover operation, the MD requests again the establishment of a new connection with WR2 ⑤, then it requests a new IP address to the DHCP server in WR2

(Event B). After establishing the connection, only the discovery service is activated, because the corresponding service has been deployed earlier in the same cloudlet server ⑥. The offloading operation is performed again for the same data ⑦ and as it was processed previously, the cloudlet recovers it from the cache and returns the result ⑧.

5.3 Implementation of the System

mCSOS is written in Java (middleware, mobility management application, and benchmark applications) and works in any cloud resource with Android/Java platform. The mobility management application was deployed with the Model-Driven Service Abstraction Layer (MD-SAL) together with Open Services Gateway Initiative (OSGI) framework (OSGI, 2016). This application runs on OpenDaylight (Matthieu Lemay and Luis Gomez, 2015) SDN controller that uses the Maven (MAVEN, 2016) for easier build automation.

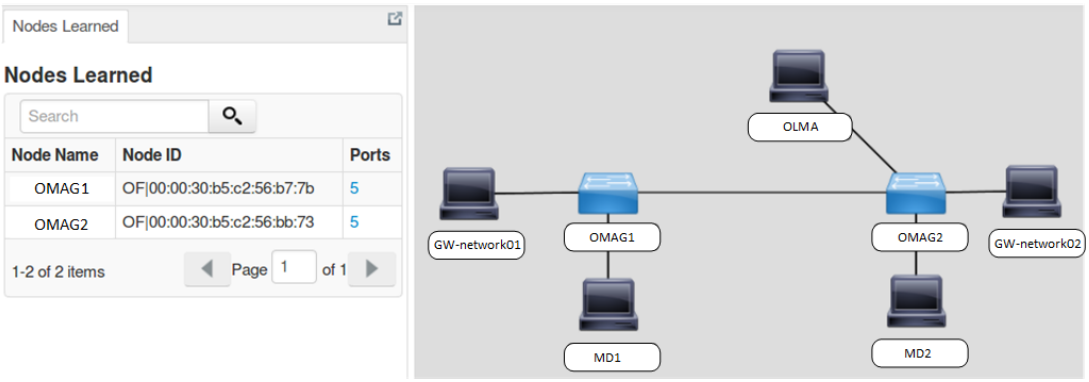


Figure 25 – Web-dashboard ODL with smartphones connected to the network.

OpenDaylight (ODL) uses Link Layer Discovery Protocol (LLDP) messages to discover the topology of the devices connected to the OpenFlow network. The ODL topology Web-dashboard (see Figure 25) provides a holistic view of the links between hosts and switches. The two OFMAGs routers are represented by switches, in contrast, the OFLMA is depicted as a host rather than a switch, physically being a switch but with the gateway function. The OFLMA is seen as the central connection point to the other networks and not as a network switch. Thus, each OFMAG has its wireless network, i.e., different IP ranges, but works both as a gateway and as a switch to mobile devices. Each OFMAG has an IP linked to one port so that the hosts connected to each network can also communicate with other networks, being named **GW-network** in Figure 25. Finally, **MD1** and **MD2** hosts representing the mobile device connected to the wireless network.

As illustrated in Figure 26, the `OFLMA` and `OFMAG` classes extend the `Thread` class, which allows managing more than one physical element of a wireless network infrastructure. We deploy our own DHCP server through the `DHCPServer` class in the controller. Therefore, the DHCP server exchanges messages with the mobile device through the

`sendOfferMessage` and `sendAckMessage` methods for IP address assignment by the `OFMAG` class.

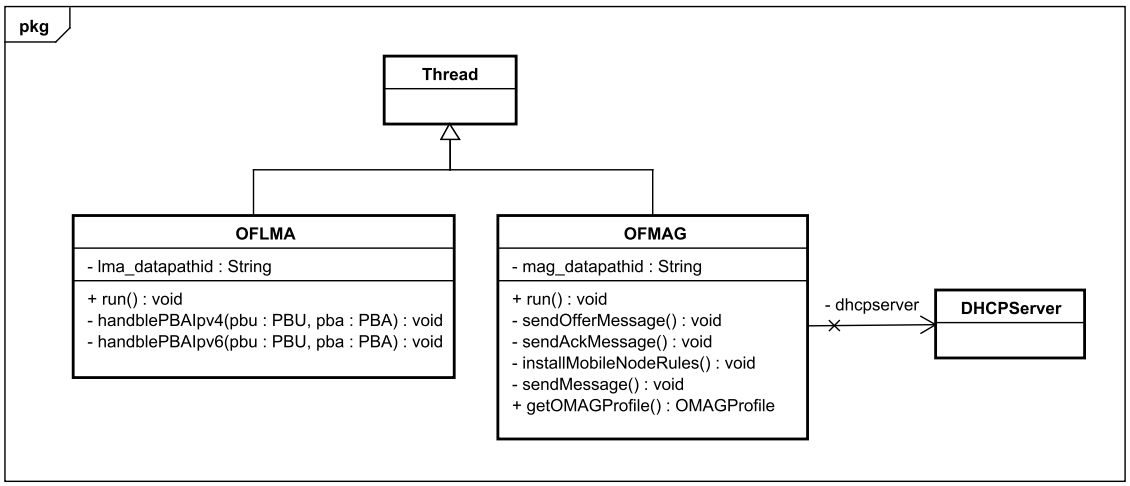


Figure 26 – Class diagram of the OFLMA and OFMAG.

The `getOMAGProfile()` feeds the OFMAG profile (i.e., status, port, identifier) that is transferred to the `MNPolicyProfile` class instance, which creates a profile for each mobile device connected to the corresponding OFMAG. Therefore, it is possible to know if the connection deals with a handover or not. If handover occurs, the `run()` method from `OFLMA` class checks the identifier (ID) used in the first connection of the mobile device and authorizes its migration. During the handover operation, the IP address is the same utilized by the previous network, which maintains the connection in the upper layers. Finally, the `installMobileNodeRule()` method is responsible for installing the routing and forwarding Openflow rules in OFMAGs to restore the communication between device and application service in the cloudlet.

In mCSOS, the developer needs to initiate the client middleware inside the `OnCreate()` from the `MainActivity`. The `OnCreate()` is usually the first function to be executed inside an activity, and it is responsible for loading XML files and other initial operations. It's also necessary to label a method candidate to be offloaded as `@Offloadable`, which indicates a computation-intensive method of a specific application.

Figure 27 illustrates the class diagram of the Math application that was developed to evaluate the mCSOS. It has two methods that apply different sorting algorithms (selection sort and insertion sort) and one method that uses the Fibonacci recursive algorithm. In the experiments of section 5.4, they are marked with the `@Offloadable`, since they require intensive processing. These marked methods must receive one or more parameters, and they must return some result.

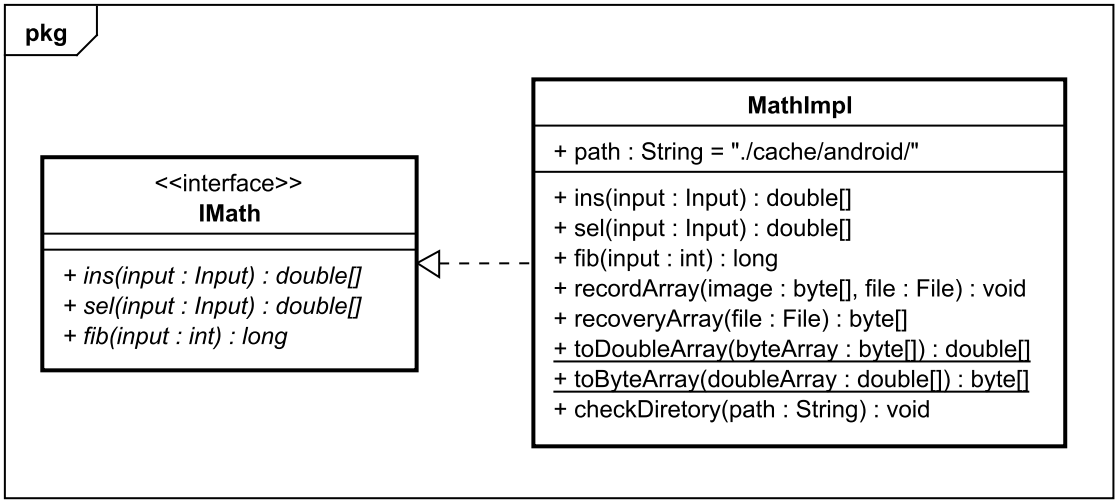


Figure 27 – Class diagram of the Math application with remote caching.

The `recordArray()` and `recoveryArray()` methods respectively stores and retrieves cache-processed data, considering that it is out of the storage directory. As there are many data types and each application may have its own remote caching mechanism, it is necessary to perform the conversion to accelerate the recovery process. For instance, to speed up the registration process, the `toByteArray()` is called before of the `recordArray()`, so as to convert the result of the numbered list in byte chain, and after to store in the cloudlet. In the case of recovery of number list (processing result), the `toDoubleArray()` is invoked after the `recoveryArray()`, to convert the byte chain back to the number list. This list is the result that must be sent to the corresponding application on the mobile device.

5.4 Performance Evaluation

In order to evaluate the proposed solution, the author has developed a benchmark application called 'Math' and employed the BenchImage application, both running on Android platform. These will be used as proof of concept to demonstrate our system's seamless mobility offloading feature.

Math application sorts a list of numeric values using two sorting algorithms, selection sort and insertion sort. The user sets both the value of list size and the algorithm to be used. The *proxy handler* sends the values that correspond to the list size to the application service on the cloud/cloudlet. Then, the application service sorts a list of unordered numbers. Finally, on the device side, the cloud/cloudlet returns the ordered list.

In all experiments, the Trepn Profiler[5] application monitors the device's battery consumption, so it is possible to collect statistics from its database.

[5] Available in https://developer.qualcomm.com/software/trepn-power-profiler.

5.4.1 Experimental Setup

Figure 28 shows an indoor mobility scenario for evaluating offloading operations with continuous handovers. Each action is described below.

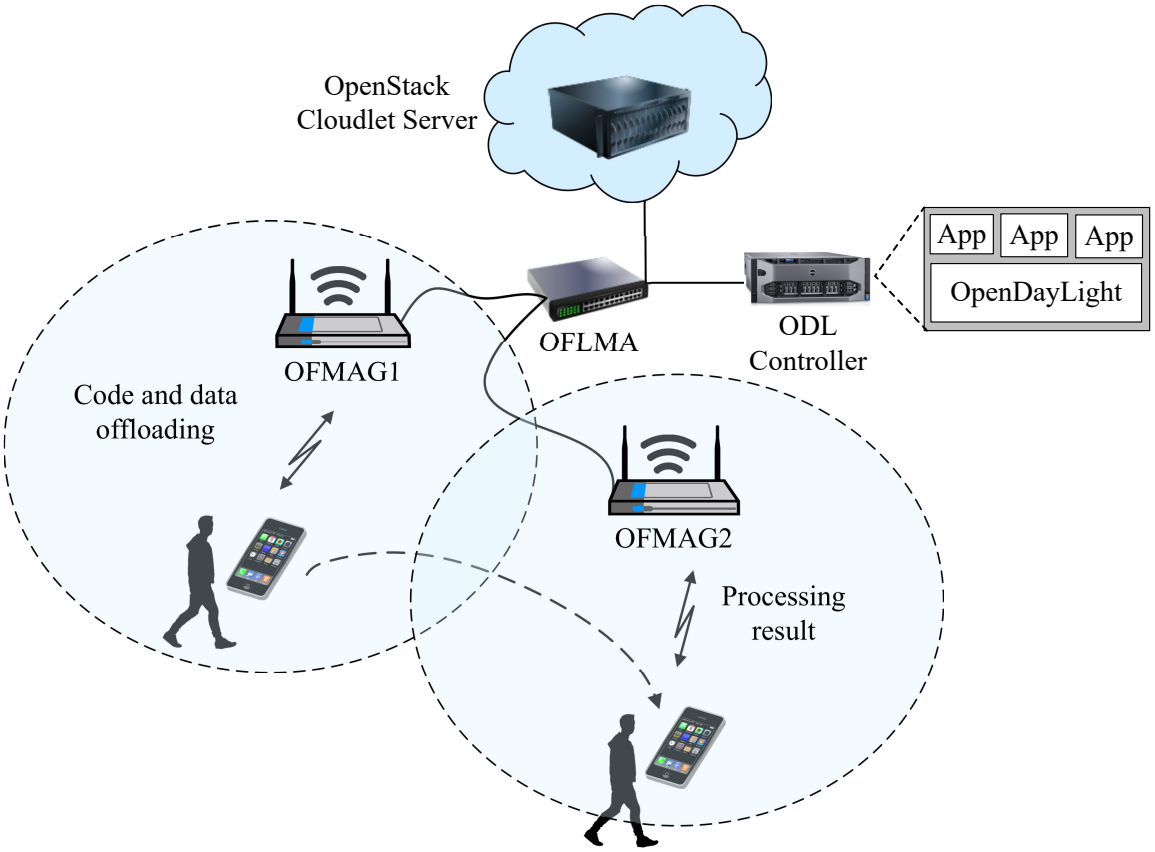


Figure 28 – Testbed scenario.

A smartphone moves linearly between two access points (OFMAGs). Initially, the *proxy handler* running on the mobile device sends data (e.g., images or numerical values) to the cloudlet server using the OFMAG1 router. Next, the mobile device starts moving towards OFMAG2 router's coverage area, which forces the device to execute a handover. So, there are two possible outcomes: if mobility management is supported, the device will wait for the processing response at the new access point; otherwise, the *proxy handler* must resend data to the cloudlet from the new access point. In both cases, after receiving the processing result, a new trip starts activating the *proxy handler* to send data to the cloudlet server through the OFMAG2 router. Then, the mobile device moves back to the OFMAG1 router's coverage area, and the aforementioned outcomes may occur depending on the adopted strategy. This scenario is repeated 30 times, i.e., 15 times in the OMAG1 and OMAG2 routers' coverage, respectively.

Table 17 presents the configuration (hardware and software) of all equipment used in our experiments. We have used three smartphones from different manufacturers and with hardware differences. In our experiment, the Handset A, Handset B, and Handset C

are high, medium, and low cost smartphones, respectively.

Table 17 – Summary of hardware and software configurations.

Equipament	**Configuration**
Mobile Device	Handset A/Android 5.1.1, Dual-core 1.8 GHz Cortex-A57+Quad-core 1.44 GHz Cortex-A53, 3GB RAM and 3000 mAh Handset B/Android 4.3, Quad-core 1.4 GHz ARM Cortex-A9, 2GB RAM and 2100 mAh Handset C/Android 4.4.2, Dual-core 1.3 GHz ARM Cortex-A7, 512 MB RAM, 1900 mAh
Cloudlet Server	Controller node/CentOS 7.1, Quad-core Intel® Xeon® E5649 2.53GHz, 6GB RAM Computer node/CentOS 7.1, Hexa-Core Intel® Xeon® E5649 2.53GHz, 12 GB RAM/Instance with Ubuntu 14.04 LTS, 6 vCPUS and 6 GB RAM
OpenFlow Controller	Ubuntu 14.04 LTS, OpenDaylight Lithium 0.3.1 SR1/Quad-core Intel i7 and 8GB RAM
Wireless Router	TP-LINK AC1200/OpenWRT Chaos Calmer 15.05, Open-VSwitch 2.3.9, 300Mbps, 2.4GHz and 802.11ac Standard
OpenFlow Switch	Switch Extreme X440-24p-10G/ExtremeXOS Release 15.7

The configuration steps of the hardware and software mentioned in Table 17 are described below: (i) **cloudlet server installation**: OpenStack kilo (OpenStack Kilo, 2016) was configured on two servers (node controller and node computer) with the CentOS 7.1 operating system, being one of the servers only for hosting the VMs (called instances); (ii) **OpenFlow controller installation**: OpenDaylight Lithium (OPENDAYLIGHT, 2016) was configured on a desktop with the Ubuntu 14.04 operating system. Then, mobility management application was configured to work with OpenDaylight; (iii) **OpenFlow switch installation**: SDN support has been enabled on the Switch Extreme X440-24p-10G and it was configured to receive rules from the OpenFlow controller; (iv) **wireless routers installation**: TP-LINK AC1200 equipment's had the default firmware replaced by OpenWRT and they was configured to receive rules from the OpenFlow controller; (v) **setting up a VM in cloudlet**: an instance was created from the Ubuntu 14.04 image, in which the server-side middleware was deployed; and (vi) **setting up a mobile device**: benchmark applications have been installed and the client-side middleware was deployed.

Table 18 shows all the parameters used in the experiment. We execute the BenchImage application 30 times for each image size (4MP and 8MP), applying the Red Ton filter. Math application is also executed 30 times for each sorting algorithm (Insertion Sort and Selection Sort) taking as input, the numerical list size 50000 and 40000, respectively.

Table 18 – Parameters used in the experiment

Strategies	**Parameters**		**Value**
Mobility Management	BenchImage	Filter	Red Ton
		Size	4MP and 8 MP
		Image	FAB Show
Remote Caching		Executions Number	30 times
Default Mobility	Math	Calculation Type	Numerical Sorting
		Sorting Algorithm	Insertion and Selection
		List Size	50000 and 40000
		Executions Number	30 times

All experiments utilized computational offloading operations with three strategies:

- Mobility management (labeled as **Mobility**): provides the entire mCSOS infrastructure (i.e., wireless router as OFMAG, switch OpenFlow as OFLMA, and OpenDaylight controller), mobility management application, discovery/deployment services and proxy handler.
- Remote caching (labeled as **Caching**):provides common network infrastructure (i.e., wireless router with DHCP server and simple switch), discovery/deployment services, proxy handler, and remote caching.
- Default mobility (labeled as **Default**):provides common network infrastructure (similar to **Caching** but without remote caching support), discovery/deployment services, and proxy handler.

Finally, to enhance the reader's comprehension, only six images will be shown (results from the 8MP image and SEL algorithm) instead of all twelve images that correspond instance vCPU and smartphone CPU for each image (4MP and 8MP) and sorting algorithm (INS and SEL).

5.4.2 Calculation Methodology of the Metrics

Processing time and energy consumption are important metrics to be considered in any mobile cloud scenario. These metrics have been carefully designed, as they have direct impact in measuring the behavior of mobile hardware. While energy required for processing and communication with cloudlet have direct relation with energy consumption, the total execution time is related to the wireless link quality and remote processing capability. Subsequent subsections describe the calculation methodology for above mentioned metrics.

5.4.2.1 Total Execution Time

The Total Execution Time (TET) metric is computed to estimate the total time for offloading computing during mobility. The TET comprises of transmitting, processing, and receiving time at the cloud. The metric is computed as follows:

$$T_{t_i} = T_{t1_i} + K_i * (T_{p_i} + T_{r_i}) + (1 - K_i) * (T_{t2_i} + T_{p_i} + T_{r_i}) \tag{5.1}$$

$$K_i \in \{0, 1\} = \begin{cases} K_i = 0 & \text{if first transmission failed} \\ K_i = 1 & \text{if first transmission is successful} \end{cases}$$

where, T_{t_i} is the total execution time of the i-th execution up to 30 repetitions, which is a summation of transmitting time (T_{t1_i}), processing time (T_{p_i}), and receiving time (T_{r_i}). Since there is a possibility of retransmit data due to handover with the adoption of remote caching and default mobility strategies, the variable K_i was added. The variable K_i receives the value "0" when the first attempt to transmit the data fails, requiring its retransmission. On the other hand, when the first transmission is successful (with the adoption of mobility management), K_i receives the value "1".

The mean time is computed through of the summation of the T_{t_i} values divided by the total number of repetitions. This metric is highest when the default mobility strategy is adopted.

$$T_m = \frac{\sum_i^n T_{t_i}}{n} \tag{5.2}$$

where, T_m represents the mean of the total execution time, and n denotes the total number of repetitions.

5.4.2.2 Energy Consumption

The energy consumption metric represents the energy consumed by mobile device for computational offloading. This metric depends of the total execution time used for calculate speed of remote computing. Power estimation takes into account the frequency and load of every CPU core, Graphics Processing Unit (GPU), and brightness of the screen. Since it was not published the model that Trepn Profiler uses to estimate power, the mathematical representation is out of scope of the paper.

$$E_{c_i} = P_{e_i} * T_{t_i} \tag{5.3}$$

$$P_{e_i} = Voltage_i * Current_i \tag{5.4}$$

where, E_{c_i} represents the energy consumption for the mobile device perform a computational offloading to cloud on the i-th execution up to 30 repetitions, which is a product of electric power (P_{e_i}) and total execution time (T_{t_i}) required to process.

The mean energy is computed through of the summation of the E_{c_i} values divided by the total number of repetitions. This metric is highest when the remote caching and default mobility strategies are adopted.

$$E_m = \frac{\sum_i^n E_{c_i}}{n} \tag{5.5}$$

where, E_m represents the mean of the energy consumption, and n denotes the total number of repetitions. In the following sections, we present and discuss the results of the experiments for the three strategies.

5.4.3 Case Study One - Handset A

Figure 29 (a)-(f) shows results obtained in the experiment using the Handset A device. Each graph shows only the first five results of up to 30 executions. A black line circle identifies the first five handovers for each evaluated environment. It is possible to identify the amount of CPU processing consumed by the Handset A and the vCPU usage by a cloudlet's instance (both at left y-axis). Besides, the network profiling between Handset A and cloudlet through the RTT (right y-axis) is depicted (red lines). The network profiler works in background and was configured to send an Internet Control Message Protocol (ICMP) packet (echo request) every one second and await the response (echo reply) by one-second timeout.

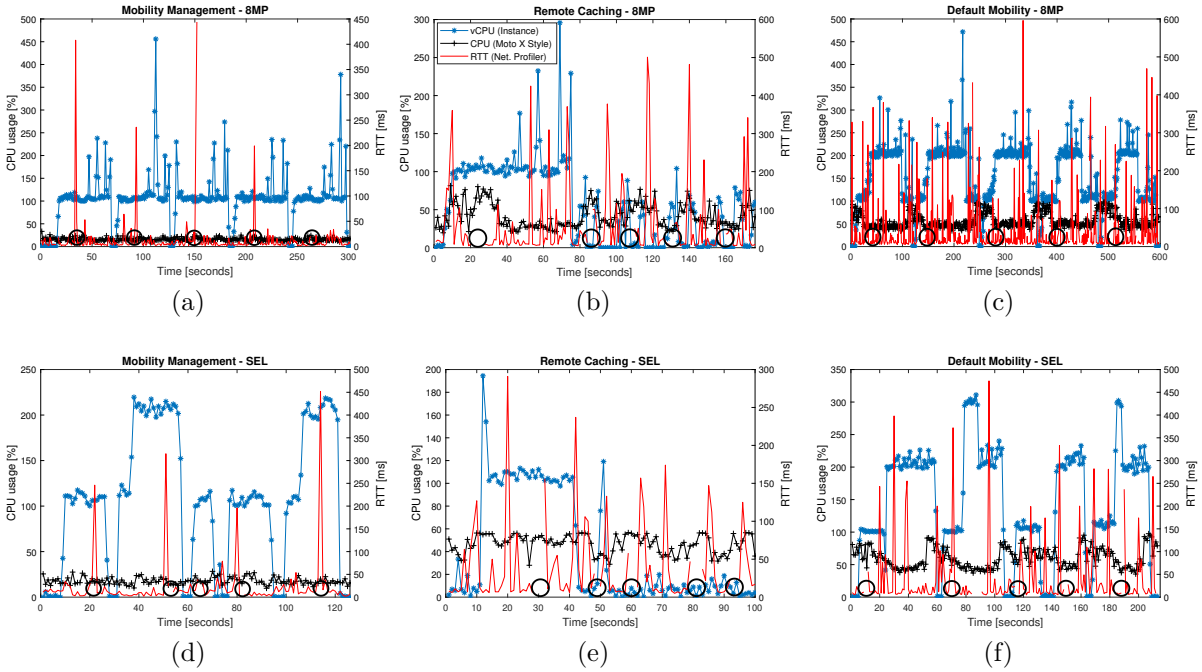


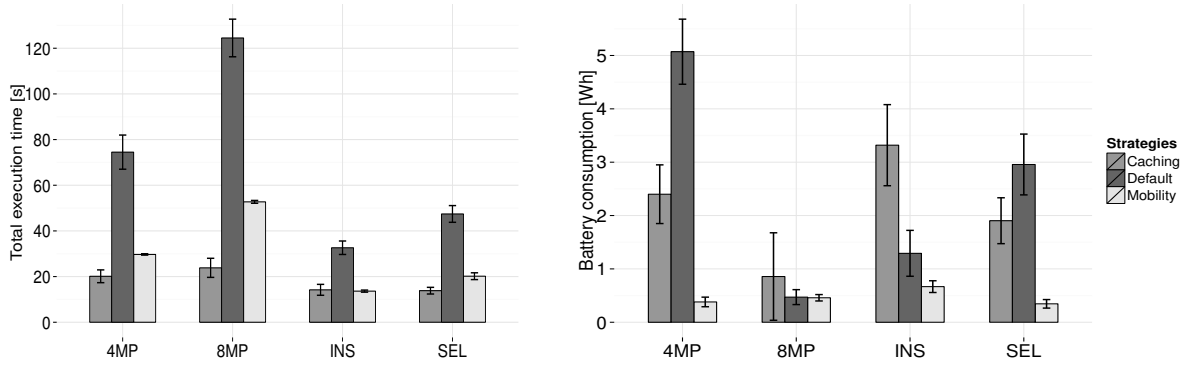
Figure 29 – Behavior of the Instance (VM) and Moto X Style CPU with five handovers.

For both applications, the RTT is kept low when using SDN-based mobility management, (see Figure 29a and 29d) while it increases and becomes highly unstable for

remote caching and default mobility scenarios. This behavior occurs due to low priority ICMP packets that suffer delay by increased network traffic during TCP packets exchange, required to repeat offloading operations and cloudlet discovery service since handover implies broken connections.

The results show that the smartphone's CPU remained mostly below 90%, indicating that the device did not perform substantial processing. However, the two results using the default strategy presented few CPU utilization peaks (see Figure 29c and 29f), due to the need to establish a new connection with the cloudlet and resend data (images or numerical values) to be processed by the cloudlet. Moreover, the instance's vCPU remains at 200% of usage in most of the time, indicating that the server is processing both first and second image (or first and second unsorted list) simultaneously; as a consequence of the device resending data due to handover. The instance can deliver to the smartphone only the second processed data, since the first data was discarded due to application port change and IP address renewal. Consequently, this may have caused an unnecessary use of instance's vCPU.

Regarding the mobility strategy, handover occurs during the data processing by cloud instance, i.e., the device requests the filter application (or algorithm processing) to the instance, next the handover occurs, and with a short extra time, the instance delivers the result to the device. In the caching strategy, instance's vCPU goes up to 100% on the first offloading. From the second offloading, the caching logic just compares the image (or the sorting result) and returns it processed from the cache. During this comparison, the handover happens and becomes necessary to carry out the second offloading, which is answered quickly by the cloudlet (see Figure 29b and 29e). The fact of not having to process the same data allows shortening the time between the second and fifth handovers.



(a) (b)

Figure 30 – Total time and battery consumption in Handset A.

Figure 30 (a)-(b) illustrates the total execution time and energy consumption of the BenchImage (4MP and 8MP) and Math (INS and SEL) applications on Handset A, while Table 19 offers a comparison of the mean and median with 95% confidence intervals. The

total time is comprised by sending data to the cloudlet, execute remote processing and return the result to the mobile device, when we use the mobility strategy (more details read Section 5.4.2). The handover also directly influences in the default strategy total time, since after the handover, the mobile device needs to reallocate a new IP (DHCP service), enable the discovery service and perform the three aforementioned steps. This leads to the results illustrated in Figure 30a, where the total execution time is longer in all four experiments.

The total time considering the mobility strategy is 60.13% and 57.64% smaller than the default strategy to the 4MP and 8MP images respectively, and approximately 32.22% and 54.8% greater than the caching strategy to the same images. Similar to the default strategy, the caching strategy needs to execute the same procedures after a handover, except to reprocess the same data, that is, it only needs to return the processing result; thus, the caching strategy seems to be appropriate for large amounts of data (4MP and 8MP images). The result corresponding to the INS algorithm shows that the mobility and caching strategies obtained a 13 second median time. Moreover, the caching strategy presented a high number of outliers in all experiments, justified by the instability of the first offloading that has a longer time than the following offloads.

Table 19 – Comparative total time and battery consumption in Handset A.

Strategies	**Param.**	**Mean time**	**Med. time**	**Mean energy**	**Med. energy**
	4 MP	29.70 ± 0.30	29.00	0.3805 ± 0.09	0.0726
Mobility	8 MP	52.73 ± 0.62	52.00	0.4589 ± 0.06	0.18110
Management	INS	13.63 ± 0.46	13.00	0.6675 ± 0.11	0.4571
	SEL	20.17 ± 1.49	18.00	0.3453 ± 0.08	0.03126
	4 MP	20.13 ± 2.80	18.50	2.4000 ± 0.55	1.036
Remote	8 MP	23.83 ± 4.18	20.00	0.8569 ± 0.82	0.2601
Caching	INS	14.20 ± 2.39	13.00	3.3190 ± 0.76	1.08700
	SEL	13.83 ± 1.46	13.00	1.9030 ± 0.43	0.63980
	4 MP	74.50 ± 7.48	75.00	5.0710 ± 0.61	3.7330
Default	8 MP	124.5 ± 8.24	120.50	0.4710 ± 0.14	0.0000
Mobility	INS	32.63 ± 2.95	33.50	1.2920 ± 0.43	0.1505
	SEL	47.43 ± 3.66	47.00	2.9570 ± 0.57	1.3490

On the Figure 30b, the results indicate that the default and caching strategies drain more energy on all four experiments, which means that both processing and connectivity efforts of the Handset A are higher due to the overhead caused by signaling and offloading operations. The energy consumption corresponding to the 4MP image shows that the mobility strategy is 84.14% and 92.49% more efficient than the caching and default strategies, respectively. The corresponding results for SEL algorithm show that the energy

consumption in mobility strategy is 81.85% and 88.32% smaller than caching and default strategies, respectively. Regarding the 8MP image results, the mean energy consumption in the mobility strategy is equal to 0.4589W/h, which is smaller compared with the caching and default strategies. Such difference between the three strategies occurred because the waiting time for receiving the results is much greater in the 8MP image experiment.

5.4.4 Case Study Two - Handset B

In the second case study, a Handset B smartphone was used to evaluate the mCSOS performance on the same testbed scenario. Figure 31a and 31d presents a more stable CPU usage in the mobility strategy when compared with the caching and default strategies.

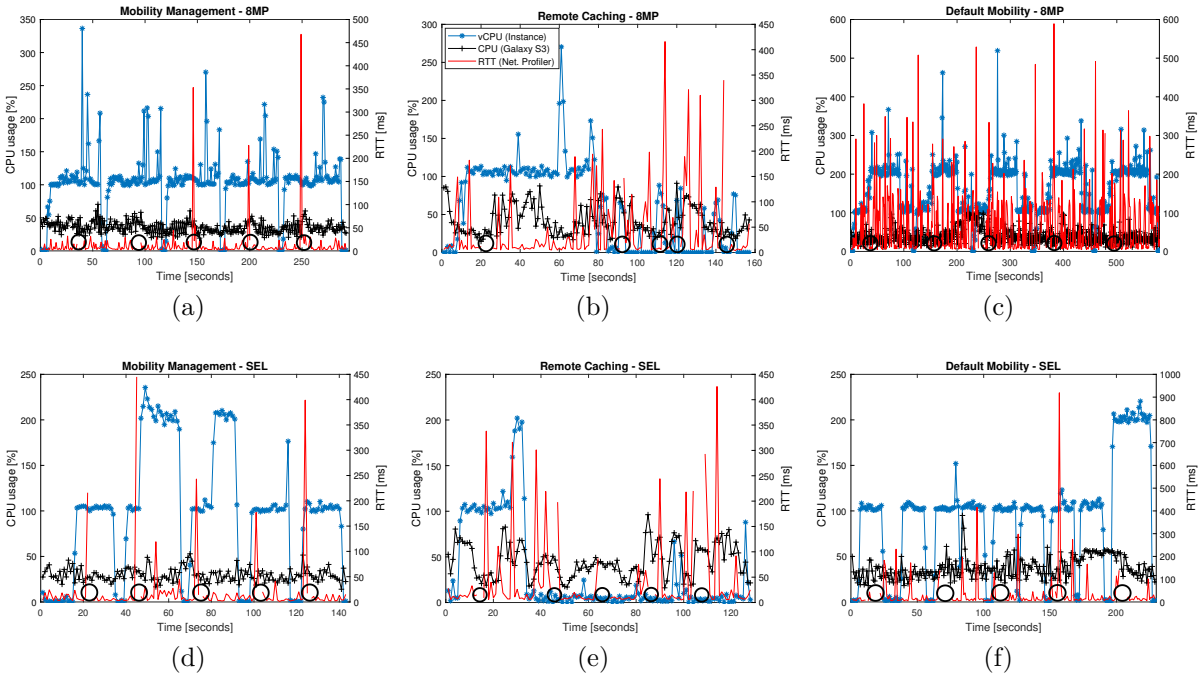
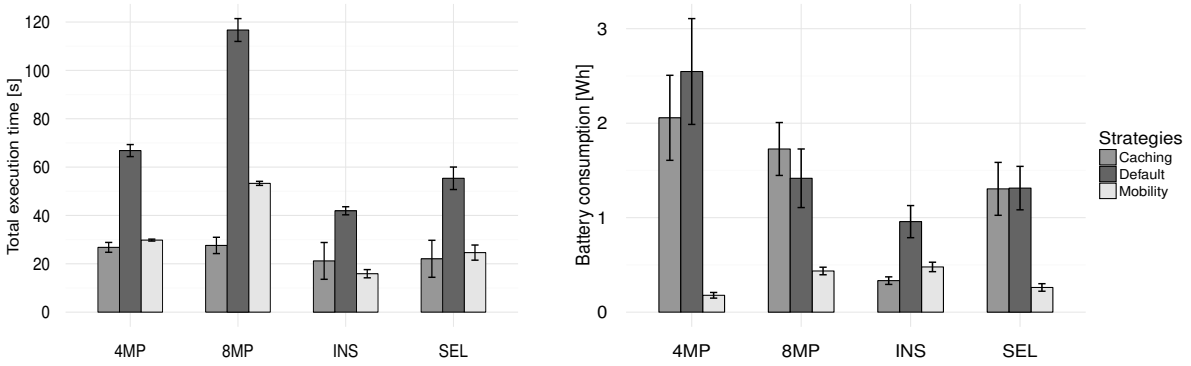


Figure 31 – Behavior of the Instance (VM) and Galaxy S3 CPU with five handovers.

We can observe many RTT variations in both remote caching and default mobility strategies due to the low priority ICMP packets when there is a significant number of offloading operations and discovery service messages in a short period. In both results corresponding to default mobility, the smartphone's CPU processes with more intensity, because data amounts affect the processing duration as a whole. The time of the first five handovers with the 8MP image reaches 600 seconds with default strategy, while that the mobility strategy reaches below 300 seconds, i.e., it is less than half the time. The long time of the default strategy occurs because the instance is processing two offloading requests at the same time, since only the second result is delivered to the smartphone. On the other hand, the short period in the mobility strategy occurs because the instance processes only one offloading request for each handover.



(a) (b)

Figure 32 – Total time and battery consumption in Handset B.

Figure 32 presents the total time (on the left) and energy consumption (on the right) on Handset B. For the 8MP image, the total execution time with default strategy is approximately 2.2 times greater than the mobility strategy and 4.46 times greater than the caching strategy. Table 20 shows that the default strategy has the worst result towards the execution time with constant handovers as it requires double processing in the cloudlet, double DHCP signaling, and discovery services.

Table 20 – Comparative total time and battery consumption in Handset B.

Strategies	**Param.**	**Mean time**	**Med. time**	**Mean energy**	**Med. energy**
	4 MP	29.80 ± 0.40	30.00	0.1784 ± 0.03	0.04696
Mobility	8 MP	53.27 ± 0.84	52.50	0.4357 ± 0.04	0.28490
Management	INS	15.90 ± 1.70	13.00	0.4785 ± 0.05	0.42940
	SEL	24.63 ± 3.13	19.50	0.2613 ± 0.04	0.17490
	4 MP	26.80 ± 2.04	26.00	2.0570 ± 0.45	1.4230
Remote	8 MP	27.60 ± 3.37	26.00	1.7270 ± 0.28	1.4310
Caching	INS	21.20 ± 7.61	16.00	0.3336 ± 0.04	0.23570
	SEL	22.07 ± 7.65	17.00	1.305 ± 0.28	1.051
	4 MP	66.83 ± 2.49	67.00	2.5470 ± 0.56	1.4400
Default	8 MP	116.7 ± 4.72	116.0	1.4170 ± 0.31	0.8079
Mobility	INS	41.93 ± 1.67	41.50	0.9582 ± 0.17	0.5344
	SEL	55.37 ± 4.65	54.50	1.3130 ± 0.23	0.76650

Besides that, for the INS and SEL algorithms and 4MP image, the performance difference between the mobility and caching strategies is relatively small. This result shows that although the caching strategy accelerates the result delivery, it requires higher smartphone processing and network signaling. Nevertheless, the caching strategy is 10.06% and 10.39% faster than mobility strategy to the 4MP image and SEL algorithm, respectively.

Regarding energy consumption, the results indicate that the mobility strategy significantly reduces the energy consumption to the 4MP/8MP images and SEL algorithm, which proves to be a promising solution concerning energy saving environments with constant handovers and offloading operations. Although the mobility strategy has the lowest execution time for the INS algorithm, it consumes more energy than the caching strategy (see Figure 32b), i.e., the energy consumption with caching strategy is 30.28% lower than when using the mobility strategy and 65.18% lower than default strategy. This solution can be a good option for energy saving with applications that use small number of I/O operations, such as Math application.

5.4.5 Case Study Three - Handset C

Figure 33 (a)-(f) shows the results for the Handset C device, which is classified as a low-cost smartphone. Similarly to the Handset A and Handset B results, the Handset C has a significant gain with the mobility strategy. The CPU usage remains close to 50% when the device is inside the mobility management environment, but remains at 100% usage when in the remote caching and default mobility environments. Unlike other smartphones, the Handset C requires more processing because it has very limited hardware (see Table 17).

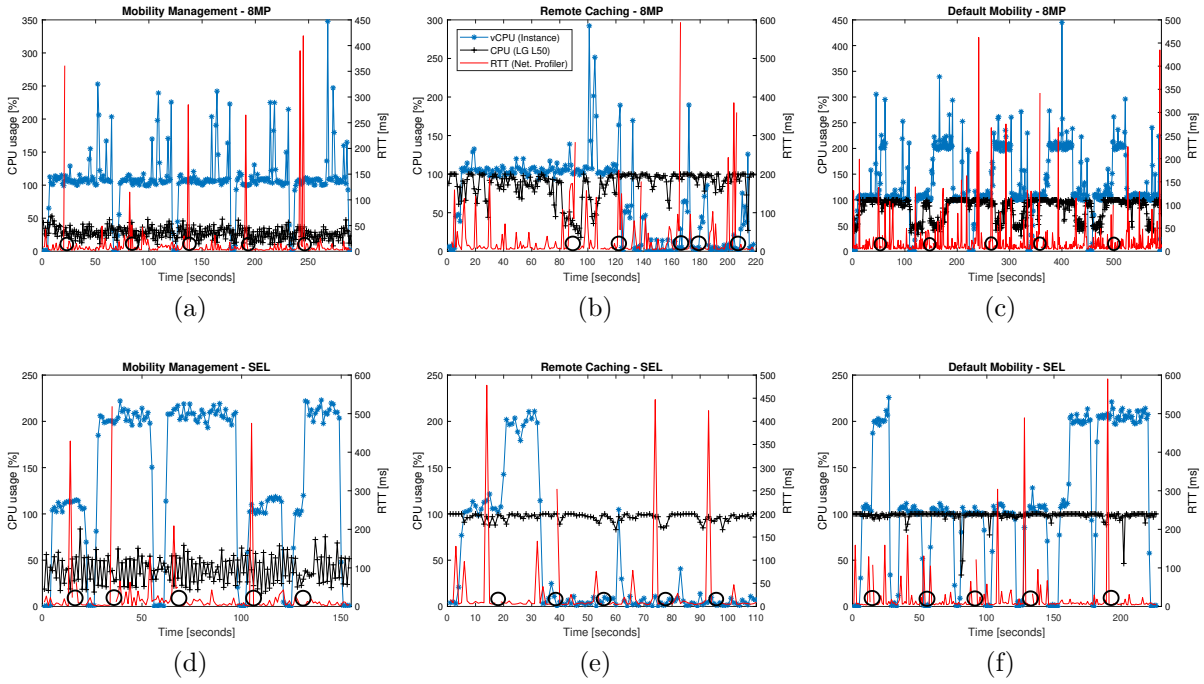
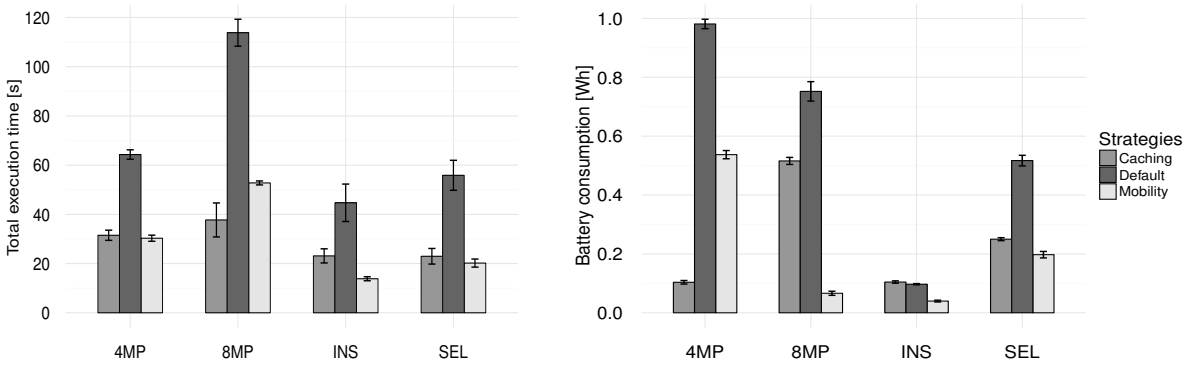


Figure 33 – Behavior of the Instance (VM) and LG L50 CPU with five handovers.

Another mobility strategy feature is to ensure the loss of only a few packets. Both results for the 8MP image and SEL algorithm present peaks in the RTT values during handover. However, even with few breaks in the RTT line, the handover was conducted

in less than one second. On the other hand, the caching strategy takes a greater time compared to other strategies, since before returning the result, the middleware needs to process and cache data. Therefore, in all results with 8MP images (Handset A, Handset B, and Handset C), the first processing happens at intervals of 10 up to 80 seconds in the caching strategy, while the mobility strategy takes of 5 up to 60 seconds.

Figure 34 illustrates the results of the total execution time and energy consumption on Handset C smartphone. The mobility strategy slightly outperforms the caching strategy with INS and SEL algorithms. For instance, it is just 12.05% faster than caching strategy to the SEL algorithm. However, the caching strategy is just 1.39 times faster than mobility strategy concerning the 8MP image. The key difference between results from mobility and caching strategies is the mobile application's category, where the mobility strategy reduces the runtime for mathematical operations (e.g., Math application), and caching strategy also reduces the runtime for image operations (e.g., BenchImage application).



(a) (b)

Figure 34 – Total time and battery consumption in Handset C.

Table 21 presents the mean and median values for the three strategies, where the median energy reveals the single favorable result when we apply the caching strategy to the 4MP image, i.e., it consumes 0.11W/h while the mobility strategy consumes 0.56W/h. Differently, the mobility strategy provides better results in most experiments, for instance, the difference between the mobility and caching strategies is 87.14% for the 8MP image.

In general, the Handset C has lower battery consumption when using the mobility management environment, since it does not require to offload twice and therefore is the most efficient using the mCSOS together with the mobility management application.

Table 21 – Comparative total time and battery consumption in Handset C.

Strategies	**Param.**	**Mean time**	**Med. time**	**Mean energy**	**Med. energy**
	4 MP	30.30 ± 1.24	29.00	0.5371 ± 0.014	0.55670
Mobility	8 MP	52.77 ± 0.81	52.00	0.0663 ± 0.007	0.048500
Management	INS	13.83 ± 0.81	13.00	0.0398 ± 0.003	0.04157
	SEL	20.20 ± 1.64	18.00	0.1974 ± 0.011	0.2119
	4 MP	31.50 ± 2.07	30.50	0.1037 ± 0.006	0.1102
Remote	8 MP	37.73 ± 6.90	34.00	0.5158 ± 0.012	0.54420
Caching	INS	23.13 ± 2.87	22.00	0.1045 ± 0.004	0.10390
	SEL	22.97 ± 3.17	21.00	0.2501 ± 0.005	0.2553
	4 MP	64.30 ± 1.92	63.50	0.9809 ± 0.016	1.0150
Default	8 MP	113.8 ± 5.47	113.00	0.7520 ± 0.033	0.8112
Mobility	INS	44.70 ± 7.60	41.00	0.0972 ± 0.002	0.10190
	SEL	55.87 ± 6.09	47.00	0.5169 ± 0.018	0.5341

5.4.6 Hypothesis Testing with ANOVA

In experimental environments with one or more treatments, one of the most widely used statistical methods is Analysis of Variance (ANOVA). The ANOVA involves the analysis of data sampled from more then one population or data from experiments with more than two treatments.

When we are conducting an analysis of variance, the null hypothesis considered is that there is no difference in treatments mean, so once rejected the null hypothesis, the question is what treatment differ. In the case of the experiments described previously, we have two treatments. The first refers to the strategy (mobility management, remote caching, and default mobility) and the second to the mobile device (Handset A, Handset B, and Handset C). Consequently we have two factors, the strategy and the mobile device.

The first objective is to verify if the tested strategies have significant differences in the improvements related to the total execution time and battery consumption, since in some cases the strategy with remote caching presented better results and in others the mobility management was superior, which leads us to believe that there may be similarity in the performance of both, i.e., the hypothesis that there is a difference between them is null. The second objective is to verify if the hardware configuration of mobile devices presents differences in total execution time and battery consumption when applied in the three strategies.

Table 22 corresponding to analysis of variance shows the P-value for the strategies and devices. The P-value is the probability of effect (or difference) observed between the treatments/categories be due to chance, and not to the factors being studied. In other words, it is the probability of equality of performance between strategies and devices. The

closer this value is to 1, probability of similarity between factors is high, and the closer to zero, lower the probability of two or more factors being equal.

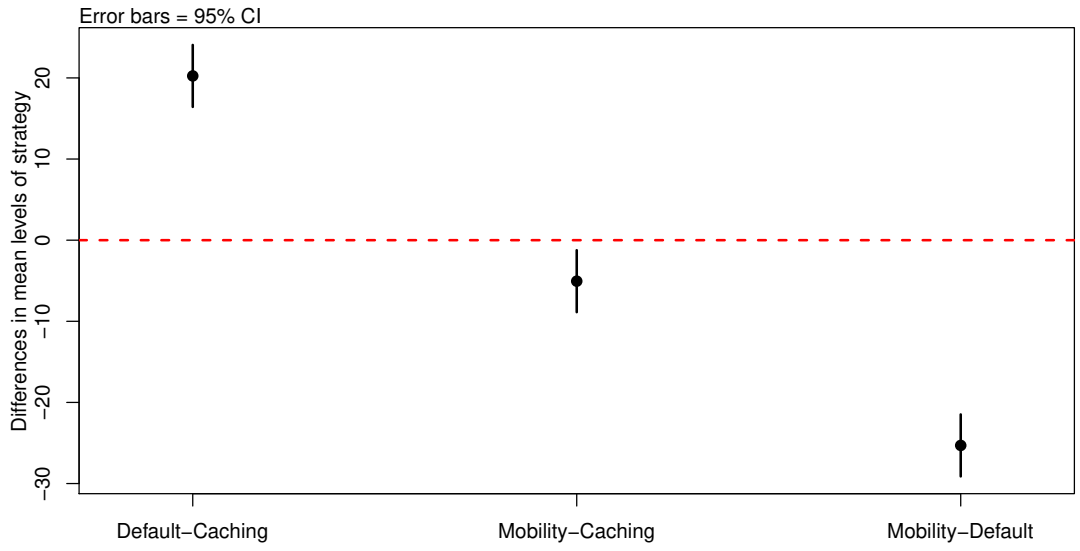
Table 22 – Comparison of P-value with ANOVA.

Analysis of Variance		
	P-value (Total Time)	**P-value (Battery Consumption)**
Strategy	<2e-16	<2.2e-16
Device	1.97e-05	<2.2e-16
Tukey's Test		
	P-value (Total Time)	**P-value (Battery Consumption)**
Mobility-Caching	0.0057065	0
Default-Caching	0	0.0051504
Default-Mobility	0	0
MotoX-L50	0.0000554	0
S3-L50	0.8509157	0
S3-MotoX	0.0004911	0

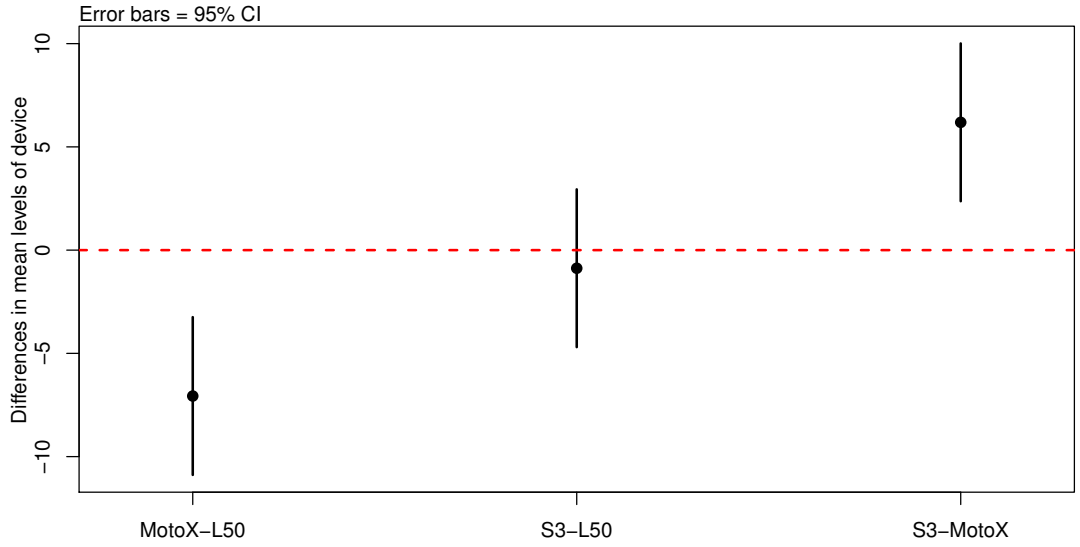
Thus, the P-value for the strategy and device is <2e-16 and 1.97e-05 corresponding to total time, respectively, and the same value <2.2e-16 corresponding to battery consumption. The results indicate that there is significant difference between strategies, as well as between mobile devices.

Once ANOVA identified a difference between treatments, the Tukey's test verifies where there are differences, comparing one with the other in the peer-to-peer format. According to the same Table 22 related to the Tukey's test, the P-value (total time) corresponding to the strategies, indicates that the differences Default-Caching and Default-Mobility are highly significant, while the difference Mobility-Caching is significant. On the other hand, the P-value (total time) corresponding to the devices, indicates that the differences MotoX-L50 and S3-MotoX are significant, while the difference S3-L50 is not significant.

A more easy way to interpret these outputs is visualizing the confidence intervals for the mean differences. Figure 35 shows the confidence interval for the means differences in relation to the total execution time. One can see that only the confidence interval for S3-L50 contain 0. Thus, it appears that the smartphones Handset B and Handset C do not differ among themselves, but are different from Handset A (see Figure 35b), i.e., when considering a significance level of 5%, we do not reject the equality hypothesis in the influence that strategies have on the performance of the Handset B and Handset C smartphones. This result is acceptable, since these smartphones are able to achieve similar performance for different strategies.



(a) Mean differences for strategy.



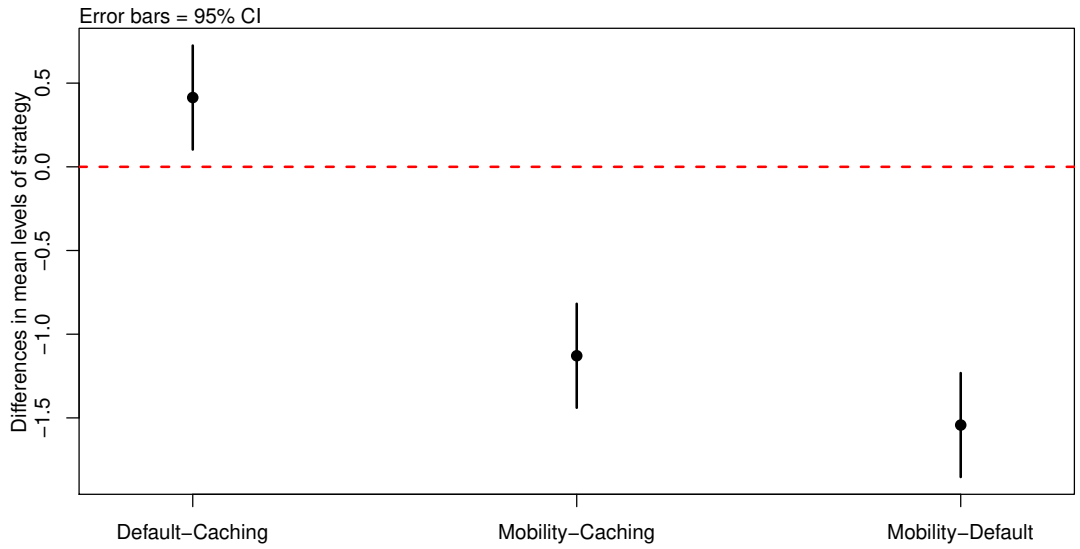
(b) Mean differences for device.

Figure 35 – Confidence level for total execution time.

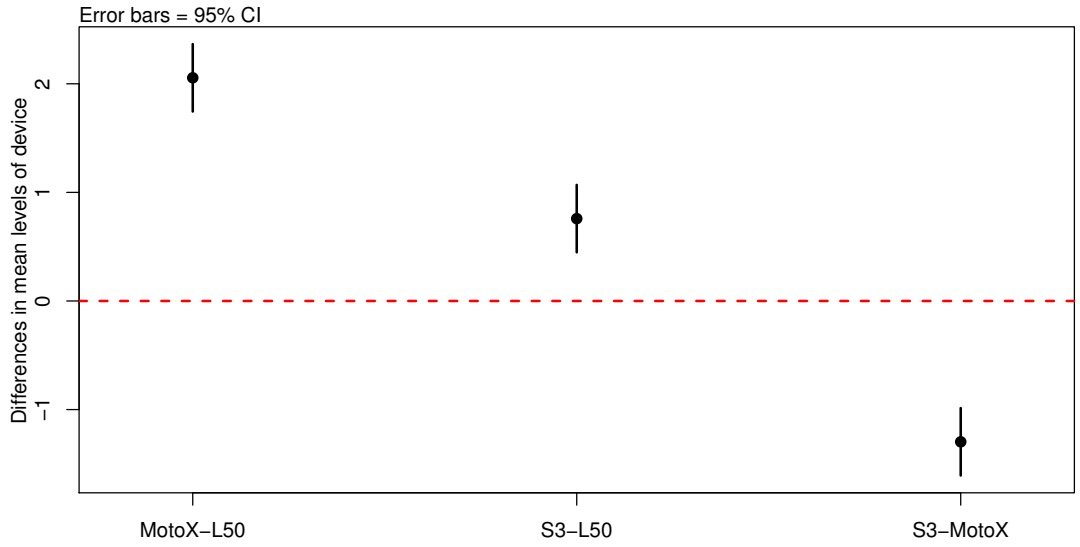
Regarding strategies (see Figure 35a), remote caching and mobility management are very different from default mobility, confirming significant improvements in application performance. These two strategies are also different, but with near performance, which allows the use of one, in case of unavailability of the other.

The P-value (battery consumption) corresponding to the strategies (see Table 22), indicates that the differences Mobility-Caching and Default-Mobility are highly signifi-

cant, while the difference Default-Caching is significant. On the other hand, the P-value (battery consumption) corresponding to the devices, indicates that the differences MotoX-L50, S3-L50, and S3-MotoX are highly significant.



(a) Mean differences for strategy.



(b) Mean differences for device.

Figure 36 – Confidence level for battery consumption.

Figure 36 shows the confidence interval for the means differences in relation to the battery consumption. Mobility management proved to be very different from remote

caching and default mobility, which proves its superiority against the other strategies when it is wanted to save energy. Regarding the devices, there is a difference in the influence that the three strategies have on the battery consumption of the three smartphones. This result is desirable, since mobility management is expected to provide specific improvements in energy savings for different hardware configurations.

5.4.7 Results Discussion

The set of experiments provided insights on the benefits of using the offloading technique with mobility and caching strategies. Particularly, the caching strategy is a promising solution for processing large data amounts, which can be applied to different smartphone categories. It is also effective when applied to situations where a user may switch to an off-line mode for a time, but resumes communication requesting an offloading operation.

On the contrary, the mobility strategy is efficient for energy saving especially considering low-cost smartphones. It also improves the performance of applications that require a small amount of data, such as Math application. mCSOS' mobility strategy implemented through a mobility management application in an OpenFlow controller saves signaling from both mobile device and mCSOS' offloading middleware, since neither smartphone has to use its air interface to execute handover signaling, nor the middleware has to repeat discovery and offloading operations upon each handover.

Besides, the mobility management ensures seamless offloading even in scenarios with frequent handovers and saves energy without affecting the performance of the mobile device. The remote caching speeds up result delivery for mobile devices that triggers the handover, however, it consumes more energy due to additional offloading operations and resources discovery signaling.

The hypothesis testing with ANOVA showed that there are significant differences between the three strategies in both application's performance and device's battery consumption. This result proves that mobility management and remote caching have improvements to be added that justify their use in real environments.

In summary, during the mobility experiments, it was noticeable that network switching was impacted from the benchmark application perspective, especially when we use strategies with remote caching and default mobility. This is because after the first remote execution made on the current wireless network and subsequent switch to the new network, the application terminated unexpectedly, due to the change of IP and access port, which consequently broke the TCP connection. So, the user must start the application again to obtain a new IP/Port and then it is able to perform offloading successfully in the new network. With the mCSOS' mobility management strategy, the benchmark application does not break during handover, because the mobility manager provides the same IP acquired in the first wireless network, allowing server-side middleware to deliver the result

without having to reprocess unnecessarily. Hence, the mCSOS reduces the signaling and computational overhead to achieve seamless offloading in MCC.

5.5 Summary

This chapter presented mCSOS, a system able to perform mobility-aware offloading that uses the network controller to ensure seamless offloading so as to soften the result delivery of processed data. We introduced the mCSOS architecture, described its components, and signaling between its entities. In addition, we presented details of the mCSOS' implementation through the BenchMath and mobility manager application class diagram.

In order to evaluate mCSOS, we performed experiments in different evaluation environments, using three different smartphones and two benchmark applications. The results have demonstrated that the mobility management in WLANs may improve the performance of mobile applications, as well as save energy of low-cost smartphones, which represent a great market share. Nevertheless, remote caching in cloudlets proved to be an attractive alternative to achieve the best application performance when a network infrastructure with mobility management is not available. Besides that, the results showed that mCSOS reduces the overhead of the connection and handover signaling, once it avoids many useless duplicate packets and delay due to handover.

6 CONCLUSIONS AND FUTURE WORK

This PhD research achieved a number of results in the areas that it has explored so far. The major contributions are the new insights of how to save energy and time of MCC applications with high accuracy and seamless mobility. We believe that context-sensitive offloading system and mobility management scheme have provided meaningful success, despite these mechanisms may still evolve and conquer further improvements.

First, this research introduced the results of a systematic mapping study about mobility mechanisms for cloud-based service by investigating scientific literature production. Given the current state of MCC research, we judge that there are few studies with controlled experiments using real solutions. We believe that this mapping study generated state-of-the-art information about the main issues because the studied subject can be understood through the provided answers. In future work, more systematic mappings should be conducted to acquire further experience to aid new experiments.This part of the thesis was published by one journal (JUNIOR; SILVA; DIAS, 2018).

Next, this research designed and developed a mobility-aware offloading system to ensure seamless offloading so as to soften the result delivery of processed data when the user is moving between PoAs. The proposed system provides SDN-based network controller, mobility management application, and remote caching. To the best of our knowledge, this is the first work to use SDN solutions in the field of MCC to provide mobility. It was evaluated using one cloudlet server, two benchmark applications, three smartphones, two wireless routers, and one OpenFlow switch. The results have demonstrated that the proposed mobility management may improve the applications' runtime, as well as save energy of low-cost smartphones. Besides that, the remote caching proved to be an attractive alternative to achieve the best application performance when a network infrastructure with mobility management is not available. This part of the thesis was published by one journal (JUNIOR et al., 2017).

Other original contribution in this work is a context-sensitive offloading technique that offload tasks of mobile applications to the cloudlet server only in favorable situations. The approach intends to minimize runtime of mobile applications and save energy of smartphones by using decision engine along with machine-learning classifiers against fluctuations in the wireless communication throughput, hardware configuration, and processing availability. We evaluated the dynamic approach by experimenting face detection and image processing applications on Android devices. To the best of our knowledge, this is the first work showing such an strategy, comprising multiple classifiers and real time profilers. The proposed system, called CSOS, integrates middleware, machine-learning

classification algorithms, and a robust profiling system, as well as transforms raw context elements to high level context information at runtime. We first evaluate the main classification algorithms under our database and the results show that JRIP and J48 classifiers achieves 95% accuracy. Secondly, we evaluate our system under controlled and real scenarios, where context information changes from one experiment to another. The experiments evidenced that CSOS makes correct decisions as well as ensuring performance gains and energy efficiency. This part of the thesis is currently under review in a journal.

Finally, this research has provided one more step in the maturation of MCC, but mobile and dynamic environment will continue being a hard research challenge.

6.1 Future Work

Following, we list some possible future work:

- **Multiple Classifier Systems (MCS):** This work focuses on the binary classification of where is better to process offloading candidates codes, locally or remotely (*Yes* and *No* classes). Our engine decision does not consider different cloud models when the remote execution is better than local execution. Thus, further work consists on increasing the spectrum of offloading opportunities considering hybrid systems with cloudlet, D2D, and remote cloud (FLORES et al., 2017). Additionally, we intend to investigate a MCS (CRUZ; SABOURIN; CAVALCANTI, 2018) approach to evolve our system and solve many real-world problems, such as ensuring the predictability of application performance, the management of multi-tasking environments, and the adaptation of workload to multiple cloud models (BHATTACHARYA; DE, 2017).

- **Distributed Mobility Management (DMM):** This research uses only one centralized controller to manage its network infrastructure and does not handles the mobility in heterogeneous networks (e.g., WiFi, LTE-A). The centralized approach represents a single point of failure, therefore poses scalability issues. According to (GIUST; COMINARDI; BERNARDOS, 2015), the DMM has recently emerged as a new paradigm to design a flat and flexible mobility architecture, exploiting the use of different networks devices (controllers, switches, routers, APs, and EnodeB) for traffic with different connectivity and mobility requirements. Hence, distributed controller and mobility management schemes are required to evolve our system so as to be aligned to fault tolerant, dense, and heterogeneous 5G networks. Also, we intend to evaluate the above-mentioned issues, as well as to investigate new resource management mechanisms to ensure QoS for delay-sensitive applications running on mobile clouds.

- **Follow Me Edge (FME):** Although in this work we mention MCC environment in general, in practice we only focused on cloudlets.Mobile Edge Computing (MEC)

pushes computing resources in the proximity of mobile users (i.e. at the mobile network edge). This highly distributed computing environment can be used to provide ultra-low latency, precise positional awareness and agile applications, which could significantly improve the user experience. Its overall objective is to demonstrate how high QoS can be maintained regardless of the mobility of users through the use of MEC, more particularly through the concept of FME. The FME ensures that the service constantly follows the user and that the user is always serviced from the closest edge (TALEB et al., 2017). However, this concept is hard to achieve, because there are no adequate techniques to efficiently migrate a service from one edge to another. Thus, further proposals could benefit from SDN and Network Functions Virtualization (NFV) in order to orchestrate the joint mobility of both device and virtual machines.

- **QoS mapping between networks/clouds:** Content-rich and resource-intensive applications such as video streaming, image processing, online games, and those using augmented reality, require QoS guarantees from the mobile network and remote cloud. These applications are delay-sensitive, and the change of location and speed of the mobile device is a non-trivial challenge to ensure a better user experience, since both access network and cloud resources may float while the user is handed over across PoAs. Thus, the proliferation of mobile devices with multi-interface support, as well as the increased demand for mobile applications, require techniques for providing computational offloading continuity and QoS to mobile users while they change their PoAs (e.g. from LTE to WiFi network and vice versa) and remote clouds (e.g. from public cloud to cloudlet server and vice versa). Besides that, effective resource management frameworks with mobility in mind must be devised in current and also in future dense 5G networks.

- **Context-aware handover decision:** The context-aware handover concept is based on the knowledge of the relevant context information of the mobile device, wireless networks, and remote clouds in order to take intelligent and accurate decisions. Thus, a context-aware handover decision strategy manages this information and evaluates context changes to come to a decision on whether the handover is necessary (or not) and on the best target access network/cloud. Contextual information such as user preferences, device location, network coverage, application input data, and cloud availability can be more relevant for one application category than for another. This requires an adaptive monitoring scheme that allows the monitoring of only the metrics that are relevant to the handover decision. For example, computation-intensive applications (e.g. face detection, mathematical operations) may require monitoring cloud-related metrics, such as the processing capacity of virtual machines, while communication-intensive applications (e.g. video streaming,

online games) may require the monitoring of network-related metrics, such as download/upload rate and RTT. Since the context information varies from application to application, and comprises network, cloud, device, and even human behavior information, a detailed study is required. Besides that, machine-learning techniques could be applied with regard to its appropriateness and accuracy to assist the handover decision.

REFERENCES

ABOWD, G.; DEY, A.; BROWN, P.; DAVIES, N.; SMITH, M.; STEGGLES, P. Towards a better understanding of context and context-awareness. In: SPRINGER. *Handheld and ubiquitous computing*. [S.l.], 1999. p. 304–307.

ABOWD, G. D.; MYNATT, E. D. Charting past, present, and future research in ubiquitous computing. *ACM Transactions on Computer-Human Interaction (TOCHI)*, ACM, v. 7, n. 1, p. 29–58, 2000.

ÅHLUND, C.; BRÄNNSTRÖM, R.; ZASLAVSKY, A. M-mip: extended mobile ip to maintain multiple connections to overlapping wireless access networks. *Networking-ICN 2005*, Springer, p. 204–213, 2005.

AHMED, E.; GANI, A.; Khurram Khan, M.; BUYYA, R.; KHAN, S. U. Seamless application execution in mobile cloud computing: Motivation, taxonomy, and open challenges. *Journal of Network and Computer Applications*, v. 52, p. 154–172, 2015. ISSN 10848045.

ALIZADEH, S.; GHAZANFARI, M.; TEIMORPOUR, B. Data mining and knowledge discovery. *Publication of Iran University of Science and Technology*, 2011.

ARCHANA, S.; ELANGOVAN, K. Survey of Classification Techniques in Data Mining. *International Journal of Computer Science and Mobile Applications*, v. 2, n. 2, p. 65–71, 2014. ISSN 2321-8363.

BACCARELLI, E.; CORDESCHI, N.; MEI, A.; PANELLA, M.; SHOJAFAR, M.; STEFA, J. Energy-efficient dynamic traffic offloading and reconfiguration of networked data centers for big data stream mobile computing: review, challenges, and a case study. *IEEE Network*, v. 30, n. 2, p. 54–61, mar 2016. ISSN 0890-8044.

BARBA, E.; MACINTYRE, B.; MYNATT, E. D. Here we are! where are we? locating mixed reality in the age of the smartphone. *Proceedings of the IEEE*, IEEE, v. 100, n. 4, p. 929–936, 2012.

BARBERA, M. V.; KOSTA, S.; MEI, A.; PERTA, V. C.; STEFA, J. Mobile offloading in the wild: Findings and lessons learned through a real-life experiment with a new cloud-aware system. *Proceedings - IEEE INFOCOM*, n. 1, p. 2355–2363, 2014. ISSN 0743166X.

BECVAR, Z.; PLACHY, J.; MACH, P. Path selection using handover in mobile networks with cloud-enabled small cells. In: *2014 IEEE 25th Annual International Symposium on Personal, Indoor, and Mobile Radio Communication (PIMRC)*. [S.l.]: IEEE, 2014. v. 2015-June, p. 1480–1485. ISBN 978-1-4799-4912-0. ISSN 2166-9570.

BETTINI, C.; BRDICZKA, O.; HENRICKSEN, K.; INDULSKA, J.; NICKLAS, D.; RANGANATHAN, A.; RIBONI, D. A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*, Elsevier B.V., v. 6, n. 2, p. 161–180, 2010. ISSN 15741192.

BHATIA, N.; AUTHOR, C. Survey of Nearest Neighbor Techniques. *IJCSIS) International Journal of Computer Science and Information Security*, v. 8, n. 2, p. 302–305, 2010. ISSN 1098-6596.

BHATTACHARYA, A.; DE, P. A survey of adaptation techniques in computation offloading. *Journal of Network and Computer Applications*, v. 78, n. October 2016, p. 97–115, jan 2017. ISSN 10848045.

BIFULCO, R.; BRUNNER, M.; CANONICO, R.; HASSELMEYER, P.; MIR, F. Scalability of a mobile cloud management system. In: ACM. *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. [S.l.], 2012. p. 17–22.

BIFULCO, R.; BRUNNER, M.; CANONICO, R.; HASSELMEYER, P.; MIR, F. Scalability of a mobile cloud management system. *Proceedings of the first edition of the MCC workshop on Mobile cloud computing - MCC '12*, p. 17, 2012.

CHEN, M.; HAO, Y.; LI, Y.; LAI, C. F.; WU, D. On the computation offloading at ad hoc cloudlet: Architecture and service modes. *IEEE Communications Magazine*, v. 53, n. 6, p. 18–24, 2015. ISSN 01636804.

COHEN, W. Fast effective rule induction. *Twelfth International Conference on Machine Learning*, p. 115–123, 1995.

CPUBURN. *The ultimate stability testing tool for overclockers*. 2017. Disponível em: <https://patrickmn.com/projects/cpuburn/>.

CPURUN. *Tool to consume CPU resource by constant usage rate.* 2017. Disponível em: <https://play.google.com/store/apps/details?id=jp.gr.java{_}conf.toytech.cpurun{&}hl>.

CRUZ, R. M.; SABOURIN, R.; CAVALCANTI, G. D. Dynamic classifier selection: Recent advances and perspectives. *Information Fusion*, Elsevier B.V., v. 41, p. 195–216, may 2018. ISSN 15662535.

CUERVO, E.; BALASUBRAMANIAN, A.; CHO, D.-k.; WOLMAN, A.; SAROIU, S.; CHANDRA, R.; BAHL, P. MAUI : Making Smartphones Last Longer with Code Offload. In: *8th international conference on Mobile systems, applications, and services.* [S.l.: s.n.], 2010. p. 49–62. ISBN 9781605589855.

DAS, K.; BEHERA, R. N. A Survey on Machine Learning: Concept, Algorithms and Applications. p. 1301–1309, 2017.

DEMŠAR, J. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, v. 7, n. Jan, p. 1–30, 2006.

DEY, A. K.; ABOWD, G. D.; SALBER, D. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-computer interaction*, L. Erlbaum Associates Inc., v. 16, n. 2, p. 97–166, 2001.

DOMINGOS, P. A few useful things to know about machine learning. *Communications of the ACM*, v. 55, n. 10, p. 78, 2012. ISSN 00010782.

DUMA, M.; TWALA, B.; MARWALA, T.; NELWAMONDO, F. V. Improving the Performance of the Ripper in Insurance Risk Classification : a Comparitive Study Using Feature Selection. *Electrical Engineering*, 2010.

ENGG, S.; SCIENCE, C. Survey on Classification Methods using WEKA. v. 86, n. 18, p. 16–19, 2014.

ENNS, R.; BJORKLUND, M.; SCHOENWAELDER, J. Network configuration protocol (netconf). *Network*, 2011.

ENZAI, N. I. M.; TANG, M. A Taxonomy of Computation Offloading in Mobile Cloud Computing. *2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, Ieee, p. 19–28, apr 2014.

EOM, H.; FIGUEIREDO, R.; CAI, H.; ZHANG, Y.; HUANG, G. MALMOS: Machine Learning-Based Mobile Offloading Scheduler with Online Training. In: *2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*. [S.l.]: IEEE, 2015. p. 51–60. ISBN 978-1-4799-8977-5.

FARHADY, H.; LEE, H.; NAKAO, A. Software-Defined Networking: A survey. *Computer Networks*, v. 81, p. 79–95, apr 2015. ISSN 13891286.

FELEMBAN, M.; BASALAMAH, S.; GHAFOOR, A. A distributed cloud architecture for mobile multimedia services. *IEEE Network*, v. 27, n. 5, p. 20–27, sep 2013. ISSN 0890-8044.

FERNANDO, N.; LOKE, S. W.; RAHAYU, W. Mobile cloud computing: A survey. *Future Generation Computer Systems*, Elsevier B.V., v. 29, n. 1, p. 84–106, jan 2013. ISSN 0167739X.

FERRARI, A.; GIORDANO, S.; PUCCINELLI, D. Reducing your local footprint with anyrun computing. *Computer Communications*, Elsevier B.V., v. 81, p. 1–11, may 2016. ISSN 01403664.

FLORES, H.; HUI, P.; TARKOMA, S.; LI, Y.; SRIRAMA, S.; BUYYA, R. Mobile code offloading: from concept to practice and beyond. *IEEE Communications Magazine*, v. 53, n. 3, p. 80–88, mar 2015. ISSN 0163-6804.

FLORES, H.; SHARMA, R.; FERREIRA, D.; KOSTAKOS, V.; MANNER, J.; TARKOMA, S.; HUI, P.; LI, Y. Social-aware hybrid mobile offloading. *Pervasive and Mobile Computing*, v. 36, p. 25–43, apr 2017. ISSN 15741192.

FLORES, H.; SRIRAMA, S. Adaptive code offloading for mobile cloud applications: Exploiting fuzzy sets and evidence-based learning. *MCS '13*, p. 9–16, 2013.

FRIEDMAN, M. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, v. 32, n. 200, p. 675–701, 1937.

FRIEDMAN, M. A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics*, Institute of Mathematical Statistics, v. 11, n. 1, p. 86–92, 1940. ISSN 00034851. Disponível em: <http://www.jstor.org/stable/2235971>.

FUNDATION, O. N. Software-defined networking: The new norm for networks. *ONF White Paper*, v. 2, p. 2–6, 2012.

GANI, A.; NAYEEM, G. M.; SHIRAZ, M.; SOOKHAK, M.; WHAIDUZZAMAN, M.; KHAN, S. A review on interworking and mobility techniques for seamless connectivity in mobile cloud computing. *Journal of Network and Computer Applications*, Elsevier, v. 43, p. 84–102, aug 2014. ISSN 10848045.

GAO, J.; GRUHN, V.; ROUSSOS, G. Mobile Cloud Computing Research - Issues, Challenges and Needs. *2013 IEEE Seventh International Symposium on Service-Oriented System Engineering*, Ieee, p. 442–453, mar 2013.

GHASEMI-FALAVARJANI, S.; NEMATBAKHSH, M.; Shahgholi Ghahfarokhi, B. Context-aware multi-objective resource allocation in mobile cloud. *Computers & Electrical Engineering*, Elsevier, v. 44, p. 218–240, may 2015. ISSN 00457906.

GIUST, F.; COMINARDI, L.; BERNARDOS, C. Distributed mobility management for future 5G networks: overview and analysis of existing approaches. *IEEE Communications Magazine*, v. 53, n. 1, p. 142–149, jan 2015. ISSN 0163-6804.

GUNDAVELLI, S.; LEUNG, K.; DEVARAPALLI, V.; CHOWDHURY, K.; PATIL, B. *Proxy mobile ipv6*. [S.l.], 2008.

HAN, J.; PEI, J.; KAMBER, M. *Data mining: concepts and techniques*. [S.l.]: Elsevier, 2011.

HOQUE, M. A.; SIEKKINEN, M.; KHAN, K. N.; XIAO, Y.; TARKOMA, S. Modeling, Profiling, and Debugging the Energy Consumption of Mobile Devices. *ACM Computing Surveys*, v. 48, n. 3, p. 1–40, dec 2015. ISSN 03600300.

HU, P.; INDULSKA, J.; ROBINSON, R. An autonomic context management system for pervasive computing. In: IEEE. *Pervasive Computing and Communications, 2008. PerCom 2008. Sixth Annual IEEE International Conference on*. [S.l.], 2008. p. 213–223.

HU, X.; CHU, T. H.; LEUNG, V. C.; NGAI, E. C.-H.; KRUCHTEN, P.; CHAN, H. C. A survey on mobile social networks: Applications, platforms, system architectures, and future research directions. *IEEE Communications Surveys & Tutorials*, IEEE, v. 17, n. 3, p. 1557–1581, 2015.

HUANG, D.; WANG, P.; NIYATO, D. A Dynamic Offloading Algorithm for Mobile Computing. v. 11, n. 6, p. 1991–1995, 2012.

HUGHES, J. F.; FOLEY, J. D. *Computer graphics: principles and practice*. [S.l.]: Pearson Education, 2014.

HWANG, C.-L.; LAI, Y.-J.; LIU, T.-Y. A new approach for multiple objective decision making. *Computers & operations research*, Elsevier, v. 20, n. 8, p. 889–899, 1993.

IPERF. *The ultimate speed test tool for TCP, UDP and SCT*. 2017. Disponível em: <https://iperf.fr/>.

JUNIOR, W.; FRANçA, A.; DIAS, K.; SOUZA, J. N. de. Supporting mobility-aware computational offloading in mobile cloud environment. *Journal of Network and Computer Applications*, Elsevier Ltd, v. 94, n. July, p. 93–108, sep 2017. ISSN 10848045.

JUNIOR, W.; SILVA, B.; DIAS, K. A systematic mapping study on mobility mechanisms for cloud service provisioning in mobile cloud ecosystems. *Computers & Electrical Engineering*, Elsevier Ltd, v. 52, p. 1–18, feb 2018. ISSN 00457906.

KANTARDZIC, M. *Data mining: concepts, models, methods, and algorithms*. [S.l.]: John Wiley & Sons, 2011.

KEMP, R.; PALMER, N.; KIELMANN, T.; SEINSTRA, F.; DROST, N.; MAASSEN, J.; BAL, H. eyedentify: Multimedia cyber foraging from a smartphone. In: IEEE. *Multimedia, 2009. ISM'09. 11th IEEE International Symposium on*. [S.l.], 2009. p. 392–399.

KEOGH, E. Naïve Bayes Classifier. 2006. ISSN 13652753.

KHAN, A. u. R.; OTHMAN, M.; KHAN, A. N.; ABID, S. A.; MADANI, S. A. MobiByte: An Application Development Model for Mobile Cloud Computing. *Journal of Grid Computing*, v. 13, n. 4, p. 605–628, dec 2015. ISSN 1570-7873.

KHAN, M. A. A survey of computation offloading strategies for performance improvement of applications running on mobile devices. *Journal of Network and Computer Applications*, Elsevier, v. 56, p. 28–40, 2015. ISSN 10958592.

KHAN, R.; OTHMAN, M.; MADANI, S. A.; MEMBER, I. A Survey of Mobile Cloud Computing Application Models. v. 16, n. 1, p. 393–413, 2014.

KIM, H.; FEAMSTER, N. Improving network management with software defined networking. *IEEE Communications Magazine*, v. 51, n. 2, p. 114–119, 2013. ISSN 01636804.

KOSTA, S.; AUCINAS, A.; Pan Hui; MORTIER, R.; Xinwen Zhang. ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In: *2012 Proceedings IEEE INFOCOM*. [S.l.]: IEEE, 2012. p. 945–953. ISBN 978-1-4673-0775-8. ISSN 0743166X.

KOTSIANTIS, S. B. Supervised Machine Learning : A Review of Classification Techniques. v. 31, p. 249–268, 2007.

KOTSIANTIS, S. B. Decision trees: a recent overview. *Artificial Intelligence Review*, Springer, p. 1–23, 2013.

KREUTZ, D.; RAMOS, F. M. V.; Esteves Verissimo, P.; Esteve Rothenberg, C.; AZODOLMOLKY, S.; UHLIG, S. Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*, v. 103, n. 1, p. 14–76, jan 2015. ISSN 0018-9219.

KWON, Y.; YI, H.; KWON, D.; YANG, S.; CHO, Y.; PAEK, Y. Precise execution offloading for applications with dynamic behavior in mobile cloud computing. *Pervasive and Mobile Computing*, v. 27, p. 58–74, 2016. ISSN 15741192.

LI, W.; ZHAO, Y.; LU, S.; CHEN, D. Mechanisms and challenges on mobility-augmented service provisioning for mobile cloud computing. *IEEE Communications Magazine*, v. 53, n. 3, p. 89–97, mar 2015. ISSN 0163-6804.

LOBILLO, F.; BECVAR, Z.; PUENTE, M. A.; MACH, P.; PRESTI, F. L.; GAMBETTI, F.; GOLDHAMER, M.; VIDAL, J.; WIDIAWAN, A. K.; CALVANESSE, E. An architecture for mobile computation offloading on cloud-enabled lte small cells. In: IEEE. *Wireless Communications and Networking Conference Workshops (WCNCW), 2014 IEEE*. [S.l.], 2014. p. 1–6.

LORENA, A. C.; JACINTHO, L. F.; SIQUEIRA, M. F.; GIOVANNI, R. D.; LOHMANN, L. G.; CARVALHO, A. C. D.; YAMAMOTO, M. Comparing machine learning classifiers in potential distribution modelling. *Expert Systems with Applications*, Elsevier, v. 38, n. 5, p. 5268–5275, 2011.

MAGURAWALAGE, C. M. S.; YANG, K.; HU, L.; ZHANG, J. Energy-efficient and network-aware offloading algorithm for mobile cloud computing. *Computer Networks*, Elsevier, v. 74, p. 22–33, 2014.

MAJEED, A. A.; KHAN, A. U. R.; ULAMIN, R.; MUHAMMAD, J.; AYUB, S. Code offloading using support vector machine. In: *2016 Sixth International Conference on Innovative Computing Technology (INTECH)*. [S.l.]: IEEE, 2016. p. 98–103. ISBN 978-1-5090-2000-3.

MASCOLO, C. The power of mobile computing in a social era. *IEEE Internet Computing*, IEEE, v. 14, n. 6, p. 76–79, 2010.

Matthieu Lemay and Luis Gomez. *OpenDayLight*. 2015. Disponível em: <https://wiki.opendaylight.org/view/Main{_}P>.

MAVEN. *Apache Maven Project*. 2016. Disponível em: <https://maven.apache.org/>.

MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; PETERSON, L.; REXFORD, J.; SHENKER, S.; TURNER, J. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, ACM, v. 38, n. 2, p. 69–74, 2008.

MEDEIROS, B.; USP, D. B.; ANTONIO, M.; JR, S.; CRISTINA, T.; BRITO, M. D.; USP, C.; ANTONIO, M.; ROJAS, T. Applying Software-defined Networks to Cloud Computing. In: *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. [S.l.: s.n.], 2015.

MERSHAD, K. SCORE : Data Scheduling at Roadside Units in Vehicle Ad Hoc Networks. n. Ict, p. 0–5, 2012.

MITRA, K.; SAGUNA, S.; AHLUND, C.; LULEA, D. G. M2C2: A mobility management system for mobile cloud computing. In: *2015 IEEE Wireless Communications and Networking Conference (WCNC)*. [S.l.]: IEEE, 2015. p. 1608–1613. ISBN 978-1-4799-8406-0.

MUHAMMAD, I.; YAN, Z. Supervised machine learning approaches: A survey. *ICTACT Journal on Soft Computing*, v. 5, n. 3, 2015.

NEMENYI, P. *Distribution-free Multiple Comparisons*. [s.n.], 1963. Disponível em: <https://books.google.com.br/books?id=nhDMtgAACAAJ>.

OPENDAYLIGHT. *OpenDayLight Lithium*. 2016. Disponível em: <https://www.opendaylight.org/lithium>.

OPENFLOW. *OpenFlow (2009). Specification, openflow switch – v1.0.0*. 2009. Disponível em: <http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf>.

OPENFLOW. *OpenFlow (2012). Specification, openflow switch – v1.3.0*. 2012. Disponível em: <https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf>.

OpenStack Kilo. *OpenStack*. 2016. Disponível em: <https://www.openstack.org/software/kilo/>.

OSGI. *The Dynamic Module System for Java*. 2016. Disponível em: <https://www.osgi.org/>.

PAGALLO, G.; HAUSSLER, D. Boolean Feature Discovery in Empirical Learning. *Machine Learning*, v. 5, p. 71–99, 1990. ISSN 15730565.

PASCOE, J. Adding generic contextual capabilities to wearable computers. In: IEEE. *Wearable Computers, 1998. Digest of Papers. Second International Symposium on*. [S.l.], 1998. p. 92–99.

PERERA, C.; ZASLAVSKY, A.; CHRISTEN, P.; GEORGAKOPOULOS, D. Context Aware Computing for The Internet of Things: A Survey. *IEEE Communications Surveys & Tutorials*, v. 16, n. 1, p. 414–454, jan 2014. ISSN 1553-877X.

PETERSEN, K.; FELDT, R.; MUJTABA, S.; MATTSSON, M. Systematic mapping studies in software engineering. In: *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*. Swindon, UK: BCS Learning & Development Ltd., 2008. (EASE'08), p. 68–77.

PFAFF, B.; DAVIE, B. The open vswitch database management protocol. 2013.

PHYU, T. N. Survey of classification techniques in data mining. In: *Proceedings of the International MultiConference of Engineers and Computer Scientists*. [S.l.: s.n.], 2009. v. 1, p. 18–20.

QI, Q.; LIAO, J.; WANG, J.; WANG, J.; LI, Q.; CAO, Y. Integrated Multi-service Handoff Mechanism with QoS-Support Strategy in Mobile Cloud Computing. *Wireless Personal Communications*, v. 87, n. 2, p. 593–614, mar 2016. ISSN 0929-6212.

QUINLAN, J. R. Improved use of continuous attributes in c4.5. *Journal of artificial intelligence research*, v. 4, p. 77–90, 1996.

QUINLAN, J. R. *C4.5: programs for machine learning*. [S.l.]: Elsevier, 2014.

RAVI, A.; PEDDOJU, S. K. Handoff strategy for improving energy efficiency and cloud service availability for mobile devices. *Wireless Personal Communications*, Springer, v. 81, n. 1, p. 101–132, 2015.

REGO, P. A. L.; CHEONG, E.; COUTINHO, E. F.; TRINTA, F. A. M.; HASAN, M. Z.; SOUZA, J. N. de. Decision Tree-Based Approaches for Handling Offloading Decisions and Performing Adaptive Monitoring in MCC Systems. In: *2017 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*. [S.l.]: IEEE, 2017. p. 74–81. ISBN 978-1-5090-6325-3.

REGO, P. A. L.; COSTA, P. B.; COUTINHO, E. F.; ROCHA, L. S.; TRINTA, F. A. M.; SOUZA, J. N. de. Performing computation offloading on multiple platforms. *Computer Communications*, Elsevier B.V., v. 0, p. 1–13, 2015. ISSN 01403664.

ROKACH, L.; MAIMON, O. Top-Down Induction of Decision Trees Classifiers—A Survey. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, v. 35, n. 4, p. 476–487, 2005. ISSN 1094-6977.

ROMAN, R.; LOPEZ, J.; MAMBO, M. Mobile Edge Computing, Fog et al.: A Survey and Analysis of Security Threats and Challenges. *Future Generation Computer Systems*, p. 28, feb 2016. ISSN 0167739X.

RUGGIERI, S. Efficient c4.5 [classification algorithm]. *IEEE transactions on knowledge and data engineering*, IEEE, v. 14, n. 2, p. 438–444, 2002.

RYU, S.; LEE, K.; MUN, Y. Optimized fast handover scheme in Mobile IPv6 networks to support mobile users for cloud computing. *The Journal of Supercomputing*, v. 59, n. 2, p. 658–675, feb 2012. ISSN 0920-8542.

SANCHEZ, L.; LANZA, J.; OLSEN, R.; BAUER, M.; GIROD-GENET, M. A generic context management framework for personal networking environments. In: IEEE. *Mobile and Ubiquitous Systems-Workshops, 2006. 3rd Annual International Conference on*. [S.l.], 2006. p. 1–8.

SATYANARAYANAN, M.; BAHL, P.; CACERES, R.; DAVIES, N. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing*, v. 8, n. 4, p. 14–23, oct 2009. ISSN 1536-1268.

SATYANARAYANAN, M.; SCHUSTER, R.; EBLING, M.; FETTWEIS, G.; FLINCK, H.; JOSHI, K. An Open Ecosystem for Mobile-Cloud Convergence. n. March, p. 63–70, 2015.

SCHILIT, B.; ADAMS, N.; WANT, R. Context-aware computing applications. In: IEEE. *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*. [S.l.], 1994. p. 85–90.

SHAUKAT, U.; AHMED, E.; ANWAR, Z.; XIA, F. Cloudlet deployment in local wireless networks: Motivation, architectures, applications, and open challenges. *Journal of Network and Computer Applications*, Elsevier, v. 62, p. 18–40, feb 2016. ISSN 10848045.

SILVA, F. A.; ZAICANER, G.; QUESADO, E.; DORNELAS, M.; SILVA, B.; MACIEL, P. Benchmark applications used in mobile cloud computing research: a systematic mapping study. *The Journal of Supercomputing*, Springer, v. 72, n. 4, p. 1431–1452, 2016.

TALEB, T.; DUTTA, S.; KSENTINI, A.; IQBAL, M.; FLINCK, H. Mobile Edge Computing Potential in Making Cities Smarter. n. March, p. 38–43, 2017.

Ting-Yi Lin; Ting-An Lin; Cheng-Hsin Hsu; Chung-Ta King. Context-aware decision engine for mobile cloud offloading. In: *2013 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*. [S.l.]: IEEE, 2013. p. 111–116. ISBN 978-1-4799-0110-4.

VIOLA, P.; JONES, M. Rapid object detection using a boosted cascade of simple features. In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001.* [S.l.]: IEEE Comput. Soc, 2001. v. 1, p. I–511–I–518. ISBN 0-7695-1272-0. ISSN 0920-5691.

WANG, Q.; WOLTER, K.; WU, H.; WANG, Q.; WOLTER, K. Mobile Healthcare Systems with Multi-cloud Mobile Healthcare Systems with Multi-cloud Offloading. n. June, 2013.

WANG, Z.; ZHONG, Z.; NI, M.; HU, M.; CHANG, C.-y. Bus-Based Content Offloading for Vehicular Networks. v. 19, n. 3, p. 250–258, 2017.

WENG, C.-H.; HUANG, T. C.-K.; HAN, R.-P. Disease prediction with different types of neural network classifiers. *Telematics and Informatics*, Elsevier, v. 33, n. 2, p. 277–292, 2016.

WITTEN, I. H.; FRANK, E.; HALL, M. A.; PAL, C. J. *Data Mining: Practical machine learning tools and techniques*. [S.l.]: Morgan Kaufmann, 2016.

WU, X.; KUMAR, V.; QUINLAN, J. R.; GHOSH, J.; YANG, Q.; MOTODA, H.; MCLACHLAN, G. J.; NG, A.; LIU, B.; PHILIP, S. Y. et al. Top 10 algorithms in data mining. *Knowledge and information systems*, Springer, v. 14, n. 1, p. 1–37, 2008.

YANG, Z.; LIU, X.; HU, Z.; YUAN, C. Seamless Service Handoff Based on Delaunay Triangulation for Mobile Cloud Computing. *Wireless Personal Communications*, dec 2014. ISSN 0929-6212.

ZEKRI, M.; JOUABER, B.; ZEGHLACHE, D. A review on mobility management and vertical handover solutions over heterogeneous wireless networks. v. 35, p. 2055–2068, 2012.

ZHOU, B.; Vahid Dastjerdi, A.; CALHEIROS, R.; SRIRAMA, S.; BUYYA, R. mCloud: A Context-aware Offloading Framework for Heterogeneous Mobile Cloud. *IEEE Transactions on Services Computing*, PP, n. 99, p. 1–1, 2015. ISSN 1939-1374.