SMALL CELL CLOUD PROOF OF CONCEPT IMPLEMENTATION AND MONITORING SCHEMES ANALYSIS

FINAL CAREER PROJECT

TELECOMMUNICATIONS ENGINEER

Author: Javier García Lloreda

Tutor: José María Sierra Cámara

Leganés, February 2015

Abstract

Cloud Computing has grown exponentially in popularity in the last few years, becoming a key technology for both personal and enterprise applications due to the numerous benefits it offers. On the other hand, Small Cell technology is considered by many to be the solution to the challenges that are expected to arise caused by the continuously increasing number of interconnected mobile devices.

This project presents a basic design and a proof of concept implementation of a Small Cell Cloud, a current research field on mobile communications that aims to leverage the capabilities offered by the parallel and distributed computation of Cloud Computing to enhance Small Cells functionality.

The purpose of the described Small Cell Cloud is to allow application offloading of mobile devices to Small Cells, allowing the execution of more resource demanding applications at the same time energy consumption is reduced in those devices.

Furthermore, a detailed analysis on different Small Cell monitoring schemes is carried out, comparing the achieved performance with each of them in terms of data reliability and generated network traffic.

Finally, based on the proof of concept implementation and a series of stress performance test, conclusions on the viability of the proposed Small Cell Cloud design and the most appropriate monitoring scheme are presented. Guidelines for future research work are also provided, considering the work developed in this project as a first step towards a new mobile technology.

Table of Contents

CHAPTER 1 INTRODUCTION 1

1.1. Introduction 1

1.2. Motivation 1

1.3. Relation to FP7 TROPIC Project 2

1.4. Objectives 3

1.5. Project structure 3

CHAPTER 2 STATE OF THE ART 5

2.1. Cloud Computing 5

2.1.1. Architecture 6

2.1.2. Business model 6

2.1.3. Types of clouds 7

2.2. Virtual Infrastructure Managers 8

2.3. Hypervisors 10

2.3.1. Xen Hypervisor 12

2.3.2. KVM 14

2.3.3. VMware vSphere 15

2.3.4. Hyper-V 16

2.4. Small Cell Technology 18

2.4.1. Small cells 18

2.4.2. Small cell challenges 19

2.4.3. Small cell ecosystem 20

2.4.4. Small cell market 21

2.5. Mobile Cloud 23

2.5.1. Fog Computing 24

2.6. Computation offloading methods 25

2.6.1. Client-server communication 25

2.6.2. Virtualization 25

2.6.3. Code mobility 26

2.7. C-RAN 26

2.7.1. C-RAN vs. Small Cells 28

CHAPTER 3 SMALL CELL CLOUD DESIGN 29

3.1. Description 29

3.2. Small Cell Manager (SCM) 30

3.2.1. Server module *31*
3.2.2. Process message module *31*
3.2.3. VM Management module *32*
3.2.4. Monitoring module *34*
3.2.5. Context Vector *35*
3.3. SMALL CELL (SC) 36
3.3.1. Server module *38*
3.3.2. Process message module *38*
3.3.3. VM action module *38*
3.3.4. Monitoring module *39*
3.3.5. Hypervisor *39*
3.3.6. Guest VMs *40*
3.4. COMMUNICATION BETWEEN COMPONENTS 41
3.4.1. Z Protocol *41*
3.5. MONITORING 47
3.5.1. Reactive monitoring *48*
3.5.2. Proactive monitoring *50*
CHAPTER 4 PROOF OF CONCEPT IMPLEMENTATION 55
4.1. SCM IMPLEMENTATION 55
4.1.1. Main thread *56*
4.1.2. Server module *57*
4.1.3. Process message module *57*
4.1.4. VM Management module *58*
4.1.5. Monitoring module *59*
4.1.6. Context Vector *60*
4.2. SC IMPLEMENTATION 63
4.2.1. Hypervisor configuration *64*
4.2.2. Guest VMs *65*
4.2.3. Libvirt API *65*
4.2.4. Main thread *66*
4.2.5. Server module *67*
4.2.6. Process message module *67*
4.2.7. VM Management module *68*
4.2.8. Monitoring module *69*
4.3. COMMUNICATION 70
4.4. UNIT TESTING 70
CHAPTER 5 PERFORMANCE EVALUATION 73

5.1. STRESS TESTS RESULTS 74
5.1.1. Context Vector *74*
5.1.2. SC connection *76*
5.1.3. Primary VM deployment *77*
5.1.4. Communication failure *77*
5.2. PROACTIVE MONITORING SIMULATION ANALYSIS 78
5.2.1. Accuracy comparison *80*
5.2.2. Generated traffic comparison *85*
CHAPTER 6 PLANNING AND COSTS 90
6.1. PROJECT PLANNING 90
6.2. PROJECT COSTS 92
6.2.1. Personnel cost *92*
6.2.2. Hardware cost *92*
6.2.3. Software cost *93*
6.2.4. Other costs *94*
6.2.5. Total costs *94*
CHAPTER 7 CONCLUSIONS AND FUTURE WORK 95
7.1. CONCLUSIONS 95
7.2. FUTURE WORK 96
LIST OF ABBREVIATIONS AND ACRONYMS 99
REFERENCES 101

List of Figures

Figure 1. Cloud computing layered structure [Image from: [14]]....7
Figure 2. Relative classification of open source VIM by niche [16]....10
Figure 3. Bare-metal hypervisor architecture....11
Figure 4. Hosted hypervisor architecture....11
Figure 5. Xen Hypervisor architecture [21]....13
Figure 6. Xen Paravirtualization overview [21]....13
Figure 7. Xen Full Virtualization overview[21]....14
Figure 8. KVM Architecture [23]....15
Figure 9. VMware vSphere architecture [24]....16
Figure 10. Hyper-V architecture [25]....17
Figure 11. Small Cells coverage comparison.....19
Figure 12. Small Cell ecosystem [27]....21
Figure 13. Small cell deployment expectation [28]....23
Figure 14. C-RAN architecture [38]....28
Figure 15. SCM basic schematic diagram....31
Figure 16. VM deployment algorithm....33
Figure 17. VM migration algorithm....33
Figure 18. VM destruction/saving algorithm....34
Figure 19. SC basic schematic diagram....38
Figure 20. VM lifecycle [43]....40
Figure 21. ZProtocol structure....42
Figure 22. Messages from SCM to SC....44
Figure 23. Messages from SC to SCM....46
Figure 24. Messages from UE to SC....47
Figure 25. Reactive monitoring sequence....49
Figure 26. Proactive monitoring sequence....51
Figure 27. Threshold-based proactive monitoring example....52
Figure 28. Threshold-based proactive monitoring, mean metric value approach....53
Figure 29. Threshold-based proactive monitoring, last sent metric approach....54
Figure 30. SCM class diagram....56
Figure 31. SCM Server and Process message modules class diagram....57
Figure 32. SCM VM Management module class diagram....58
Figure 33. VM deployment time diagram....59
Figure 34. SCM Monitoring module class diagram....60
Figure 35. SC class diagram....64
Figure 36. XML Domain definition template....66
Figure 37. SC Process Message and Server modules class diagram....67
Figure 38. SC VM Management module....68
Figure 39. SC Monitoring module....69
Figure 40. Ping test between SCM and SC....73
Figure 41. CV stress test row insertion result....74

Figure 42. CV stress test row retrieval result 75
Figure 43. SC connection delay stress test result 76
Figure 44. VM deployment delay stress test resutl 77
Figure 45. Low variability CPU pattern 79
Figure 46. Periodic with low variability CPU pattern 79
Figure 47. Periodic with high variability CPU pattern 80
Figure 48. Pseudo-random CPU pattern 80
Figure 49. Periodic monitoring accuracy simulation results 81
Figure 50. Threshold monitoring, mean metric value (2 last metrics) accuracy simulation results 82
Figure 51. Threshold monitoring, mean metric value (5 last metrics) accuracy simulation results 83
Figure 52. Threshold monitoring, mean metric value (10 last metrics) accuracy simulation results 83
Figure 53. Threshold monitoring, last sent metric accuracy simulation results 84
Figure 54. Threshold monitoring, mean metric value (2 last metrics) number of metrics sent simulation results 86
Figure 55. Threshold monitoring, mean metric value (5 last metrics) number of metrics sent simulation results 86
Figure 56. Threshold monitoring, mean metric value (10 last metrics) number of metrics sent simulation results 87
Figure 57. Threshold monitoring, last sent metric number of metrics sent simulation results 88
Figure 58. Project planning Gantt diagram 91

List of Tables

TABLE 1. MESSAGES FROM SCM TO SC 43
TABLE 2. MESSAGES FROM SC TO SCM 44
TABLE 3. MESSAGES FROM UE TO SC 47
TABLE 4. CV DATABASE TECHNOLOGY COMPARISON 62
TABLE 5. HARDWARE COST DETAIL 93
TABLE 6. SOFTWARE COST DETAIL 93
TABLE 7. TOTAL PROJECT COST DETAIL 94

CHAPTER 1
INTRODUCTION

1.1. Introduction

With the increasing number of mobile devices, the more and more demanding applications and high bandwidth requiring network services, wireless communications are being exposed to new technical challenges: increase coverage areas to allow ubiquitous access to the network, have enough capacity to deal with a huge number of devices and improve the energy efficiency of the elements of the network at the same time that the node availability and the security of the communication are guaranteed.

In order to cope with the coverage and data transmission requirements that are currently demanded, and are expected to be exponentially larger in the next few years, network operators see in the Small Cell trend the opportunity to extend and improve their mobile service. First appearing in 2007, Small Cells consist on reduced and densely distributed versions of mobile cells. Due to their characteristics, Small Cells are nowadays becoming more and more relevant in the wireless sector with the explosion of mobile devices and the arrival of broadband mobile communication technologies such as LTE (Long Term Evolution) and LTE-A (Long Term Evolution – Advanced).

On the other hand, Cloud Computing has nowadays become a common solution for enterprises and individuals to provide and consume services over the network. Cloud Computing has allowed the appearance of new distributed services, ranging from infrastructure resources up to single software applications, able to manage and process large amounts of data thanks to the parallel computing paradigm.

Currently, research efforts are taking place in order to move the Cloud Computing capabilities towards the network edge, closer to the end user, in order to reduce the data transmission delay and the traffic in the core network. The work developed in this project becomes part of that research analysis going a step beyond aiming to enhance the Small Cell technology leveraging the capabilities of Cloud Computing.

1.2. Motivation

Smartphones, tablets, wearables and the emerging devices under the technological wave of the Internet of Things, despite their continuous development, will always have some limitations in

terms of computing power, battery life or storage capacity due to size constraints. Some technological approaches to improve the capacity of those devices that involve task offloading techniques are being studied. By sharing the load among different devices or transferring it to more powerful equipment, mobile devices are able to become lighter, cheaper and more efficient in terms of energy consumption.

The Cloud Computing paradigm is currently able and used to offload high demanding computation tasks by leveraging parallel computation among several nodes that can be either centralized or distributed. However, the hardware servers that form the "cloud" are located at the core of the network, i.e. far from the client, usually in a different city or country or even on the other side of the Earth.

The trend is thus to increase the number of serving nodes and approach them to the end-users in order to improve both the network delay and the reliability of the communication against node failures by means of redundancy. The proximity of the cloud to the edge of the network also allows reducing the amount of traffic that is sent to the network core, lowering the traffic congestion probability.

The smallest versions of Small Cells, called femtocells, are aimed precisely to be located at the very end of the network, with a reduced coverage area but highly dense deployment, to provide low energy traffic offloading from wireless to wired networks. Thus femtocells characteristics make them a suitable research area to solve the trending Cloud Computing requirements.

1.3. Relation to FP7 TROPIC Project

The contents of the present work are developed under the framework provided by the Seventh Framework Programme (FP7) TROPIC project.

The Framework Programmes for Research and Technological Developments are funding plans created by the European Commission to support and improve research activities in Europe [1]. Specifically, FP7 was launched in 2007 with duration of seven years, until 2013, and a total budget of €50 billion.

TROPIC [2] is a Specific Targeted Research Project (STREP) of the 7th Framework Programme that aims at exploiting the convergence of pervasive femto-network infrastructure and cloud computing paradigms for virtualization/distribution of applications and services, providing a far deeper research analysis on the subject than the one carried in this project.

This project takes a small subset of the work developed for TROPIC project as a basis to introduce more detailed description and new features on the components, extend it with the work related to the definition and analysis of monitoring schemes for a Small Cell Cloud and perform the actual implementation of the system, with its corresponding tests and results.

The work developed in this project has been taken as an input for the on-going TROPIC developments and will be available as part of a deliverable for the EU Commission at the end of TROPIC project.

1.4. Objectives

The purpose of the present project is to provide a proof of concept implementation of a femtocell system that combines the capabilities of Cloud Computing, thus making a so called Small Cell Cloud. The work done is primarily focused in the distributed virtualization requirements and monitoring mechanisms. The integration with existing mobile communication base stations will be left for further study.

The objectives expected to be achieved in the project are summarized in the following points:

- **Provide a design of Small Cell Cloud components**: by defining the key components that would be required for moving the Cloud Computing virtualization and parallel computing capabilities to femtocells.
- **Develop a proof-of-concept implementation of the Small Cell Cloud based on the previous design principles**: by developing a real implementation of the defined components with the key functionalities necessary to act as a distributed cloud.
- **Define and study femtocell monitoring mechanisms**: by describing different monitoring solutions that can be used for the femtocells in the system and doing a detailed analysis of their performance in terms of network congestion and data reliability.
- **Determine the viability of the system based on the proof-of-concept implementation**: by executing a series of performance test to measure the characteristics achieved by the proof-of-concept implementation.

1.5. Project structure

The present project is structured in 7 chapters following a design-implementation-tests scheme.

Chapter 2 describes the current State of the Art of the involved technologies. It starts with a detailed description regarding Cloud Computing types and virtualization management, follows with a review on the Small Cell paradigm to finally end with an overview of Mobile Cloud and related technologies intended to enhance mobile devices capabilities.

In Chapter 3 a description of the main elements of the Small Cell Cloud are provided, as well as the introduction of various monitoring mechanisms that can be used. This will provide the key points to be implemented and afterwards tested.

Chapter 4 is devoted to the implementation details of the system components and its configuration, following the same steps taken in the design section.

In Chapter 5, the complete system performance is evaluated by detailing the tests and simulations carried over, with the corresponding obtained results.

Chapter 6 describes the planning of the whole project and the total estimated cost related to the work developed throughout this project.

Finally, Chapter 7 contains the conclusions extracted from the work done in this project and the work that has been left for future studies on the topic.

CHAPTER 2
STATE OF THE ART

2.1. Cloud Computing

The definition of Cloud Computing can vary depending on the source, as analyzed in [3], where more than 20 different definitions are compared in order to establish a unified description of Cloud Computing. According to the National Institute of Standards and Technology (NIST), *Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction* [4].

The term cloud has been addressed to describe networks since ATM interconnections in the 1990s. However, it was not until 2006 when Google's CEO at that time, Eric Schmidt, referred the business model of providing services through the Internet to the word cloud. Cloud computing is more than a new computation model than a new technology, a paradigm that brings together several technologies such as virtualization and parallel computing to meet today's Information Technologies requirements.

Cloud computing offers several features that are of special interest for business owners such as:

- **No big initial investment**: resource provisioning costs are minimized to a computer with Internet connection combined with the pay as you go pricing model for resources on the cloud.
- **High scalability**: resources can be scaled up on demand to match traffic peaks without difficulty and accommodate accordingly to business growth.
- **Easier management**: users do not need to know the underlying technology and worry about power or space consumption.
- **Location independent**: services offered through Internet become accessible from every location where an Internet connection is present.
- **Device independent**: most web-based services are not designed for an specific platform, allowing users to use

- **Reducing business risks and maintenance expenses**: service providers avoid risks of resource failures by outsourcing the infrastructure to the cloud.

2.1.1. Architecture

The cloud computing architecture is primarily composed of 4 layers: the hardware layer, the infrastructure layer, the platform layer and the application layer.

The hardware layer: This layer is composed by the physical resources of the cloud: routers, switches, physical servers, power systems… This layer is represented by data centers where servers are stacked in racks and interconnected by means of routers and switches.

The infrastructure layer: the purpose of this layer, also named as virtualization layer, is to create and manage the dynamic provisioning of a pool of virtual resources created on top of previous layer using virtualization technologies such as Xen, KVM and VMware.

The platform layer: this layer is where operating systems and application frameworks are hold. This eases the deployment of applications for developers as they do not need to deal with virtual machines directly.

The application layer: this is the top layer and it is where cloud applications are stored. The difference of this kind of applications compared to a more traditional application is that cloud apps can take advantage of the dynamic allocation of resources provided by virtual machines and therefore scale up accordingly to the demand.

The modular structure of cloud computing allows the creation of a wide range of applications and business models associated to them, depending on the layer or layers where providers are focused.

2.1.2. Business model

The business model of cloud computing is based on providing services through Internet either they are hardware virtualized or software resources. Although every layer of the architecture can be implemented as an independent service, cloud services are grouped into three categories, from bottom to top: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS).

Infrastructure as a Service: this model offers the dynamic provisioning of resources, typically already virtualized. The IaaS provider owns and maintains the equipment while the user rents out the specific services it needs, usually on a "pay as you go" basis. IaaS providers are Amazon Web Services (AWS) [5], Windows Azure [6] or IBM SmartCloud Enterprise [7].

Platform as a Service: this model provides the elements of the platform layer as a service, offering components that are pre-configured and maintained by the service provider, including software frameworks, databases or the whole operating system for users to develop their web

applications. Apart from the previous IaaS providers that also offer PaaS platforms, in this category specific providers such as Google App Engine [8], Heroku [9] or Red Hat OpenShift [10] can be found.

Software as a Service: this model offers on-demand web applications over the Internet. Those applications are hosted and managed by the service provider, therefore eliminating the administration and maintenance tasks from the user perspective and providing access from different locations everywhere through Internet. An example of SaaS providers are services from Akamai [11], Cloud9 [12] or SAP Business ByDesign [13] platforms.

Although each service provider can be an independent entity, even relying in bottom-layer providers, in practice both IaaS and SaaS are built together from the same organization, receiving the name of infrastructure or cloud providers.

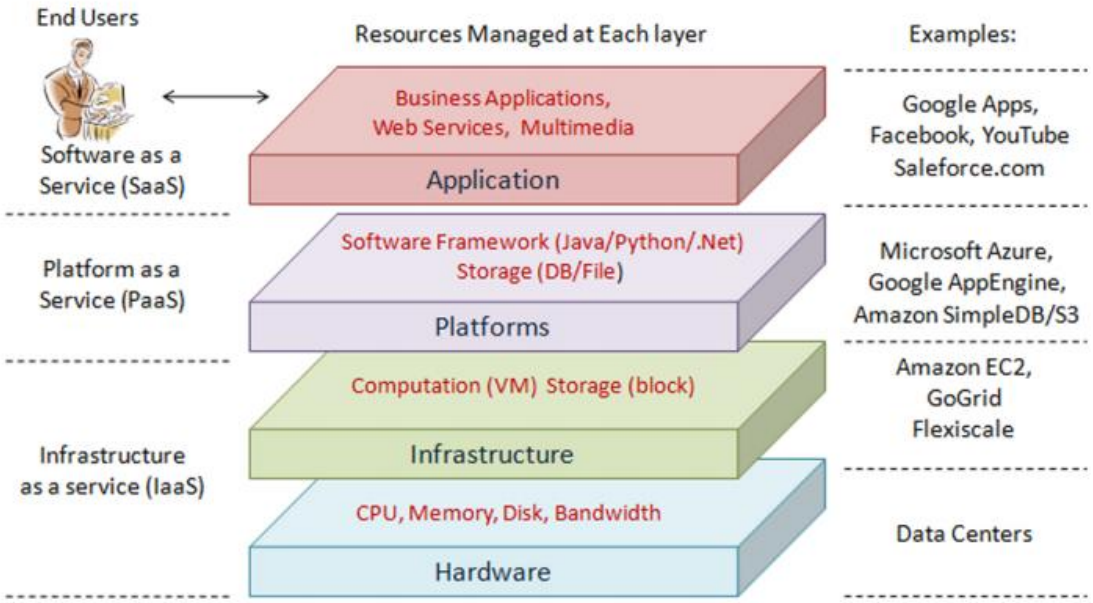

Figure 1. Cloud computing layered structure [Image from: [14]]

2.1.3. Types of clouds

Once the business models used on cloud computing are described, it is required to do a classification on the different types of cloud that are offered nowadays. Depending on the location and the requirements of cloud applications, clouds can be classified in basically three types:

Public cloud: these are clouds in which the computing infrastructure is offered to the general public according to provider premises and conditions. The customer does not know where the resources are located and the infrastructure is also shared between different organizations. The benefits of this type of cloud are that there is not any need of initial costs on infrastructures and resource managing responsibility is moved to the provider side. On the other hand, customers

lack the capacity of fine tuning those infrastructures in terms of data control, network or security settings.

Private cloud: in this scenario cloud infrastructure is dedicated to a specific customer. Because of that, they are more expensive but at the same time provide more control and security than the public cloud. Apart from being served by dedicated cloud providers, private clouds can be hosted in the same organization. However, that approach does not offer the initial investment benefits that public clouds provide and produces some controversy regarding whether to consider it as a real cloud environment or just an extension of the traditional datacenter.

Hybrid cloud: a hybrid cloud is composed of public and private clouds with the purpose of achieving the best of both cloud models. The service to be offered is divided in two components and each component is hosted in a different type of cloud. In this way, higher control and security is obtained from the private cloud while having the benefits of scalability and reduced infrastructure costs. The critical point when choosing a hybrid cloud model is to determine the best distribution of the service components between the clouds.

Besides those traditional cloud models, a type of cloud based on the combination of VPN and private clouds can be considered, called Virtual Private Cloud.

Virtual Private Cloud: this type of cloud comes as an alternative to hybrid clouds. Instead of using a private cloud, a new virtual layer is created on top of a public cloud by using virtual private network (VPN) technology, which allows customers not only to use the infrastructure provisioning they need but also to design and configure the virtual network topology and security.

The use of public cloud over private cloud or the other way around is primarily based on the application requirements. Customers have to evaluate the infrastructure costs and the level of control they need and then choose whichever cloud fits better for the service they want to provide.

2.2. Virtual Infrastructure Managers

Within the increasing cloud computing ecosystem, Infrastructure-as-a-Service has become one of the most popular offered services. Although these services have been usually centered on public clouds, the interest of companies on private clouds and building their own IaaS clouds has grown in last years. The aim of deploying their own private clouds is to provide internal users a flexible and dynamic infrastructure to run different applications within their own network. Regardless of the location of the private cloud, they have to fulfill a series of requirements such as: provide a uniform and homogeneous view of virtualized resources independently of the underlying virtualization platform, manage virtual machines full life cycle, support configurable resource allocation policies that meet organization's goals and be flexible to organization resource demands [15].

In order to achieve those requirements, virtual infrastructure managers (VIM), sometimes referred to as Cloud Management Platforms (CMP), have become essential components in private clouds. Open source cloud management platforms such as OpenStack, OpenNebula, Eucalyptus and CloudStack have increased their popularity within the virtual infrastructure management market against proprietary solutions like the ones offered by Amazon or VMware.

OpenNebula was released in 2008, being one of the first VIM to appear in the market. Developed under Apache License, OpenNebula was aimed to provide a high level of centralization and customization of virtual infrastructures [16]. It is based on a pure private cloud concept, in which users use a front-end interface in the head node to manage the rest of the network. OpenNebula manages the storage, networking and virtualization technologies allowing the creation of services on a distributed infrastructure, combining resources of physical machines and cloud servers. It is composed by the core, which manages the lifecycle of virtual machines (deployment, monitoring and migration); the scheduler, module in charge of performing workload balance from the core in virtual machines; and the virtualization layer, where virtual access drivers are available. OpenNebula customization capabilities offer administrators and users to access low level network functions and configure to a high detail the creation, migration and destruction of virtual machines. However, this customization may lead to errors by unexperienced users since they have to set up almost every parameter.

OpenStack was released two years after OpenNebula as well under Apache License as an open source project and since then it has received a great support from the community and major cloud, software and hardware vendors. According to [17], in the last quarter of 2013 it can be seen that OpenStack is leading the market of cloud management frameworks, followed by CloudStack, OpenNebula and Eucalyptus. Opens tack consists of a series of different modules each one in charge of managing and control a component of the cloud infrastructure, like processing pools, storage and networking resources. For that purpose, it provides a centralized web-based dashboard, command-line tools and a RESTful API. OpenStack consists of 5 main modules although it counts with 11 more tools and services to complete their catalog [18]. Computing control is carried by projects Nova (OpenStack Compute), the main part of an IaaS system, and Glance (OpenStack Image service), that performs the discovery, registration and delivery services for disks images; Neutron (OpenStack Networking) is in charge of managing networks and IP addresses and Swift (OpenStack Object Storage) and Cinder (OpenStack Block Storage) form the storage system.

Apache CloudStack is the second most popular platform to manage and orchestrate resources on an IaaS cloud. It provides similar characteristics as OpenNebula and OpenStack based on the regions and zones concepts. It allows cloud resources to be grouped into multiple geographic regions, each one controlled by its own cluster of Management Servers, useful for providing fault tolerance and disaster recovery [19]. Datacenters inside a region are grouped by zones which benefit infrastructure organization to provide physical isolation and redundancy. Server racks and hosts maintained in them have their own nomenclature as well, called pods and clusters.

Despite the fact that all these open source platforms seem to be competing for a leading position in the VIM market, it is important to notice that they are not completely substitutive technologies but they have their own specialized market niche and can coexist in the cloud computing ecosystem [20]. Depending on the vision organizations have when building a cloud two cloud models can be distinguished: datacenter virtualization and infrastructure provisioning. Organizations that see cloud as a datacenter virtualization look for extending the existing datacenter with virtualization technologies and therefore they need tools that help them with the management of virtual resources. This approach is usually deployed in private clouds where a higher control of the infrastructure is required. On the other hand, cloud can be understood as a way to serve resources similar to what Amazon Web Services offers and the tools they need are focused on the on-demand provisioning and supply of virtualized resources. This cloud model is usually deployed in public clouds, simplifying the management of lower cloud layers.

A chart with an approximate classification of the 4 most popular open source cloud management platforms according to the cloud models previously mentioned is shown in Figure 2.

This figure offers an overview of the relative characteristic different VIMs provide. It shows how depending on enterprise needs, some cloud platform may fit better than others, although all of them have common features that will meet basic management of cloud resources.

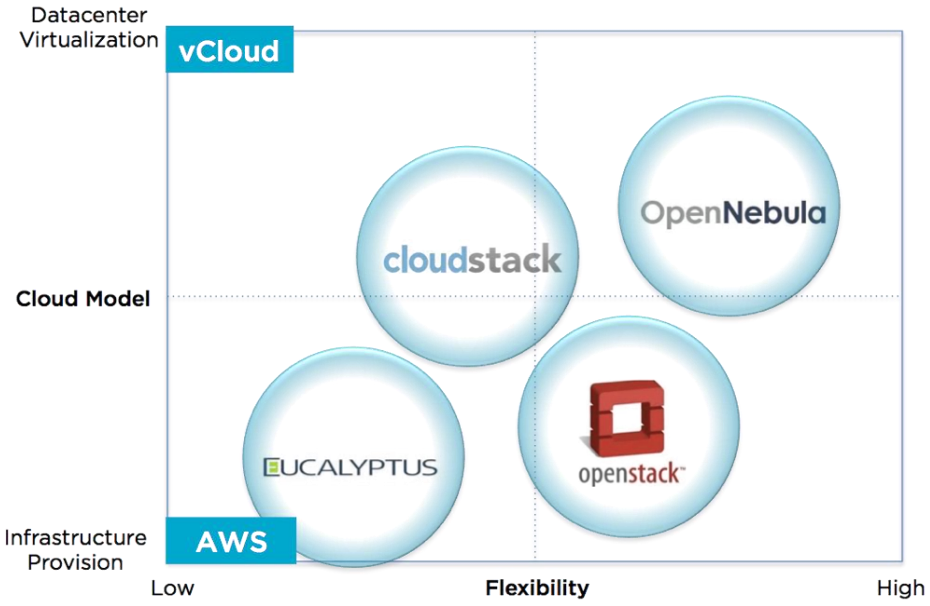

Figure 2. Relative classification of open source VIM by niche [16]

2.3. Hypervisors

An essential component in virtualization environments is what is called the Hypervisor or Virtual Machine Manager (VMM). This is a piece of software whose purpose is to manage

virtual machines and provide them with physical resources like CPU, memory, storage… Hypervisor emulates a hardware device for each virtualized Operating System (OS) and handles its communication with physical resources.

Two different configurations can be found depending on where the hypervisor is located in the virtualization architecture.

- **Bare-metal or native hypervisor**. Also known as Type 1. In this configuration Hypervisor is located right above of the physical hardware layer, acting as an intermediary between VMs and hardware. A main management VM is used to manage guest VMs and their communication with physical resources.

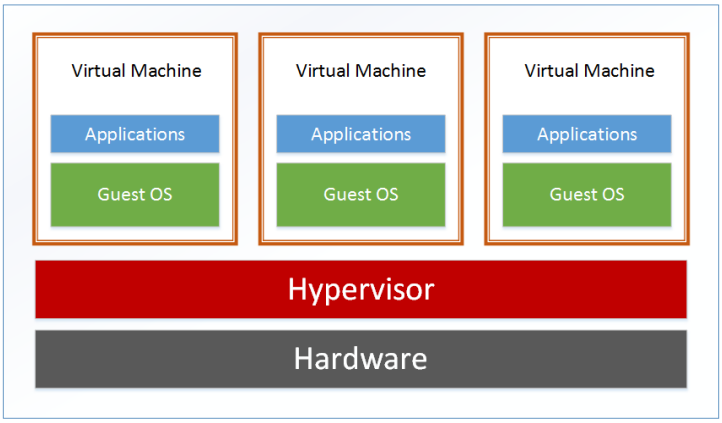

Figure 3. Bare-metal hypervisor architecture

- **Hosted hypervisor**. Also known as type 2. In this type of virtualization architecture, the Hypervisor is located on top of the host OS, running as an actual application. Contrary to the previous model, in hosted virtualization the host OS is the one in charge of managing access to physical hardware and guest VMs.

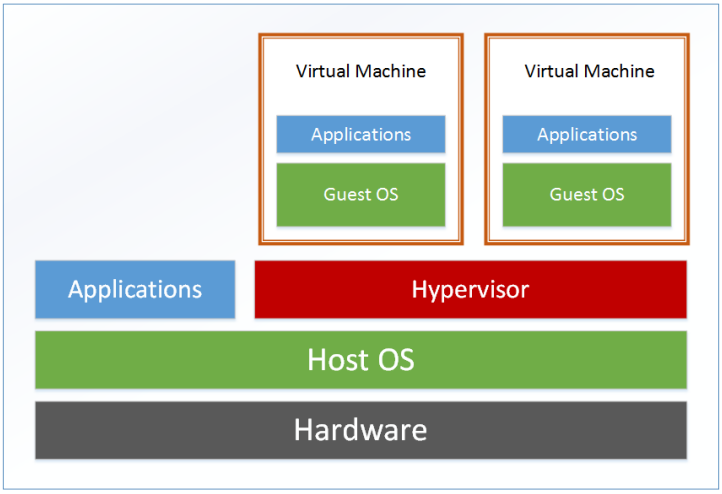

Figure 4. Hosted hypervisor architecture

The key aspect of Hypervisors is that they allow running different OS on the same physical hardware while being isolated from one another. The computer on which the hypervisor is running is called host machine and all the virtual machines created are called guests machines. Virtual Machines are just the logical units in which hardware is partitioned. Therefore, Hypervisors provide isolation between VMs and VMs and the host OS, management of all low level operations on the physical resources by intercepting host I/O calls and mapping them to the corresponding device, and access to the complete state of every VM running on the host machine, feature that allows to capture a snapshot of the current VM state and perform backup and rollback operations on it.

2.3.1. Xen Hypervisor

Xen Hypervisor is part of the XenProject, an Open Source project developed by a world-wide community of individuals, researchers and companies and that follow the Xen Project Governance process based on openness, transparency and meritocracy [21]. It is currently supported by companies such as Citrix, Amazon WS, Google, Intel, Cisco and Oracle among others and it is hosted by the Linux Foundation as a Collaborative Project.

It powers the largest clouds in production, being the core virtualization technology on Amazon Web Services.

2.3.1.1. Xen Architecture

The Xen Hypervisor is a Type 1 Bare metal hypervisor, making it the only type-1 hypervisor that is available as open source.

However it differs from the traditional Type 1 architecture: the hypervisor only contains the minimum required functionality to share the physical resources among VMs, not including device models and drivers for those resources. Therefore it requires an actual Operative System with appropriate modifications to be executed on top of Xen, to provide a way to manage the system. The manager OS resides in the Control domain or *Dom0* as it is referred in Xen terminology. Any GNU/Linux (except Red Hat Enterprise Linux 6) and BSD distributions are supported as *Dom0* OS. This control domain runs in top of the hypervisor and contains the device models and drivers that guest VMs need in order to communicate with the hardware. Thus The Xen hypervisor is not usable without *Dom0*, which is the first domain started by the system.

With this approach, the hypervisor layer becomes lighter and more easily maintainable at the same time that it doesn't require special drivers and allows to be managed as a GNU/Linux system, which significantly lowers its learning curve.

Guests VMs can run their own operating system and applications and they are totally isolated from the hardware. This means that Guests VMs have no privilege to access hardware or I/O functionality, granting them the name *DomU* for unprivileged domain.

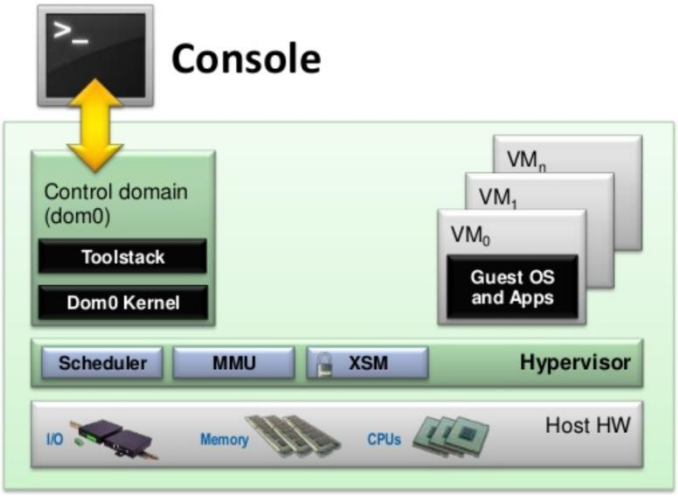

Figure 5. Xen Hypervisor architecture [21]

Xen supports two different virtualization modes: Paravirtualization (PV) and Hardware-assisted or Full Virtualization (HVM). Both guest types can be used at the same time on a single Xen system. It is also possible to use techniques used for Paravirtualization in an HVM guest. This approach is called PV on HVM.

2.3.1.1.1. Xen Paravirtualization (PV)

Paravirtualization is an efficient and lightweight virtualization technique introduced by Xen, later adopted by other virtualization platforms. PV does not require virtualization extensions from the host CPU. However, paravirtualized guests require a specially modified kernel and PV drivers, so the guests are aware of the hypervisor and can run efficiently without emulation or virtual emulated hardware. Figure 6 show the PV architecture as used in Xen.

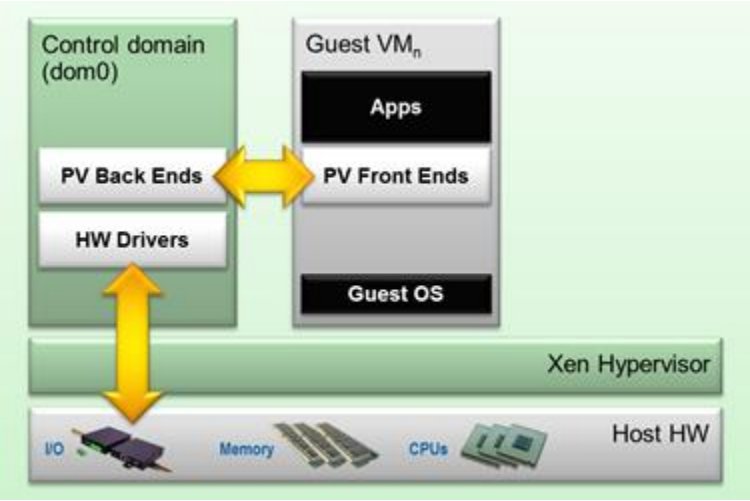

Figure 6. Xen Paravirtualization overview [21]

2.3.1.1.2. Xen Full Virtualization (HVM)

Full Virtualization or Hardware-assisted virtualization uses virtualization extensions from the host CPU to virtualize guests aiming to boost the performance of the emulation. Contrary to PV, fully virtualized guests do not require any kernel support, thus allowing any type of OS to be used as a HVM guest. Due to the required hardware emulation, fully virtualized guests are usually slower than paravirtualized guests.

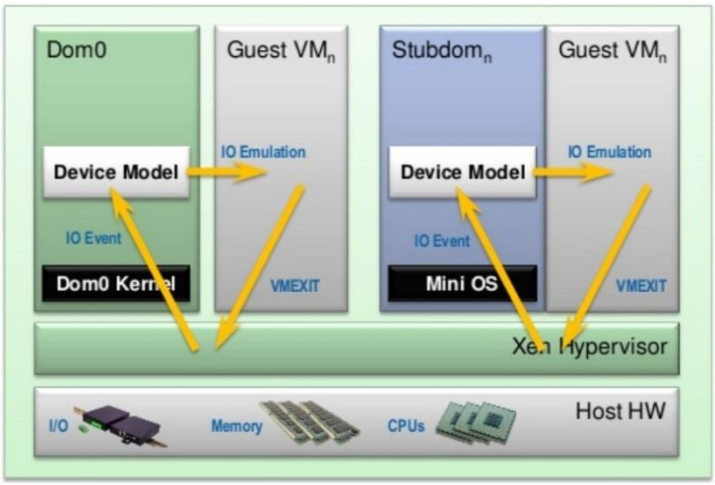

Figure 7. Xen Full Virtualization overview[21]

A stub domain is a specialized system domain running on a Xen host used in order to disaggregate the control domain (*Dom0*). The use of stub domains to run device models instead of as processes in *Dom0* provides advantages such as: Security, the stub only has privileges over the specific HVM domain to which it is attached; Scalability, by providing unique access to the resources allocated to the stub; Isolation, stub domains do not share resources among other stubs; Performance, stubs are directly scheduled by the hypervisor, eliminating the waiting time for the control domain to be scheduled first.

2.3.2. KVM

According to [22], KVM (Kernel-based Virtual Machine) is a full open source virtualization solution for Linux on x86 hardware containing virtualization extensions on modern Intel and AMD processors (Intel VT or AMD-V). KVM is not an independent hypervisor but a collection of kernel modules, that provide the core virtualization infrastructure, and processor specific modules for main processor manufacturers Intel and AMD. As part of the Linux kernel, KVM uses the regular Linux scheduler and memory management, resulting in a light and simple virtualization solution.

KVM also requires a modified QEMU (Quick EMUlator) binary. QEMU is a piece of software developed to emulate every physical resource of a machine i.e. CPU, memory, disk… KVM leverages both the capabilities of QEMU and the performance enhancements of the Linux kernel.

The type of virtualization KVM provides is HVM virtualization and although it does not support paravirtualization for CPU, it can support paravirtualization for device drivers to improve I/O performance.

Due to the nature of KVM it is not clear whether it should be categorized as a native or host hypervisor. The fact is that KVM runs within the Linux kernel, what makes it not a pure native hypervisor. But in contrast, KVM is not a hosted hypervisor either, since it does not run over the operating system and deploys virtual machines completely separated from the host operating system. Therefore it is widely agreed to consider KVM as a native hypervisor despite not running explicitly over the hardware.

2.3.2.1. KVM Architecture

As described before, KVM is integrated into the Linux kernel. All Linux guests have virtIO drivers integrated and windows and other operating systems have as well virtIO drivers that can be installed. Along with guest operating systems, native Linux applications run side by side with them without modification.

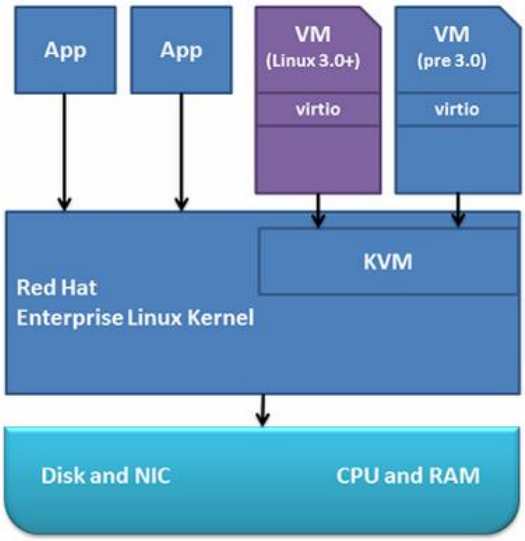

Figure 8. KVM Architecture [23]

VirtIO is an API for virtual Input/Output that implements network and block driver logic. Its motivation resides in defining a common I/O virtualization API that can be used among the many existing hypervisors of all types.

2.3.3. VMware vSphere

VMware vSphere is virtualization product that contains a native hypervisor called ESXi hypervisor (also developed by VMware). VMware vSphere is offered as a basic free version that includes the hypervisor and a built-in management software, and a non-free version with more advanced features.

VMware vSphere software is characterized for the amount of features the package includes, although a license has to be purchased to get all the features it offers. One of the paid features is a read-write API which can be used by an external tool to manage the system.

VMware minimum system requirements are higher than the previous hypervisor solutions. As described in [24] a dual core CPU, 4 GB of RAM and 80 GB of storage are needed for the minimum setup.

2.3.3.1. Architecture

The hypervisor architecture of vSphere is common to any traditional hypervisor; the virtualization layer abstracts the running virtual machines with their own operating systems from the hardware layer. Advanced products such as vCenter suite add new management layers to the architecture, allowing a complete decentralized management and higher levels of resource virtualization.

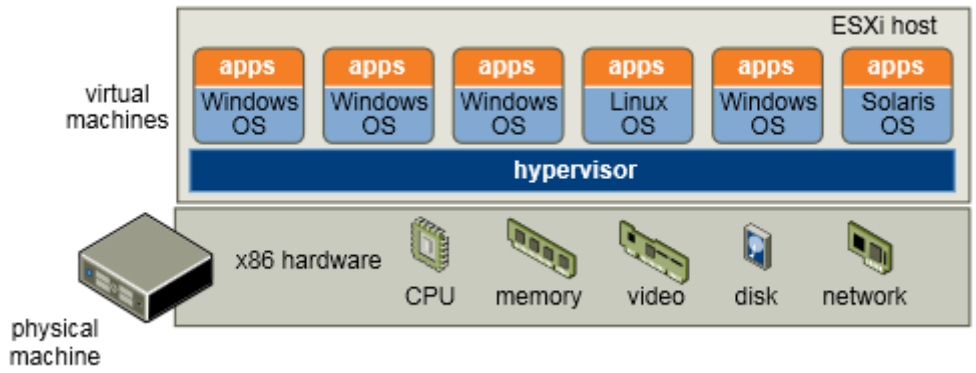

Figure 9. VMware vSphere architecture [24]

2.3.4. Hyper-V

Hyper-V is a native hypervisor-based server virtualization product from Microsoft for x86-64 architectures. As a Microsoft product, Hyper-V is not open source and it requires a license. It comes in two versions: as a stand-alone product called Hyper-V Server or as an extension to Windows Server editions [25].

Microsoft Hyper-V Server is designed as a reduced version of Windows Server containing the minimum requirements for virtualization, removing services and Graphical User Interface (GUI) to make the server as small as possible therefore requiring less configuration and maintenance.

One of the most common uses for Hyper-V Server is in corporate Virtual Desktop Infrastructure (VDI) environments. VDI allows a Windows client operating system to run on server-based virtual machines in the datacenter, which the user can access from a PC, thin client, or other client device.

Although the host OS has to be Windows, any kind of OS can run as a guest operating system.

2.3.4.1. Architecture

The Microsoft hypervisor needs a root or parent partition, running Windows OS, where the virtualization stack is available and has direct access to the hardware resources. Guests OS are hosted in child partitions which are created by the parent partition. The guest OS have a virtual CPU and its own dedicated virtual memory address region. The hypervisor is in charge of handling the processor interrupts and redirects them to the appropriate partition.

Hyper-V counts with an Input Output Memory Management Unit (IOMMU) that operates independently of the memory management hardware used by the CPU and can improve the speed of memory translation among guests virtual address spaces by remapping physical memory addresses to the addresses that are used by child partitions.

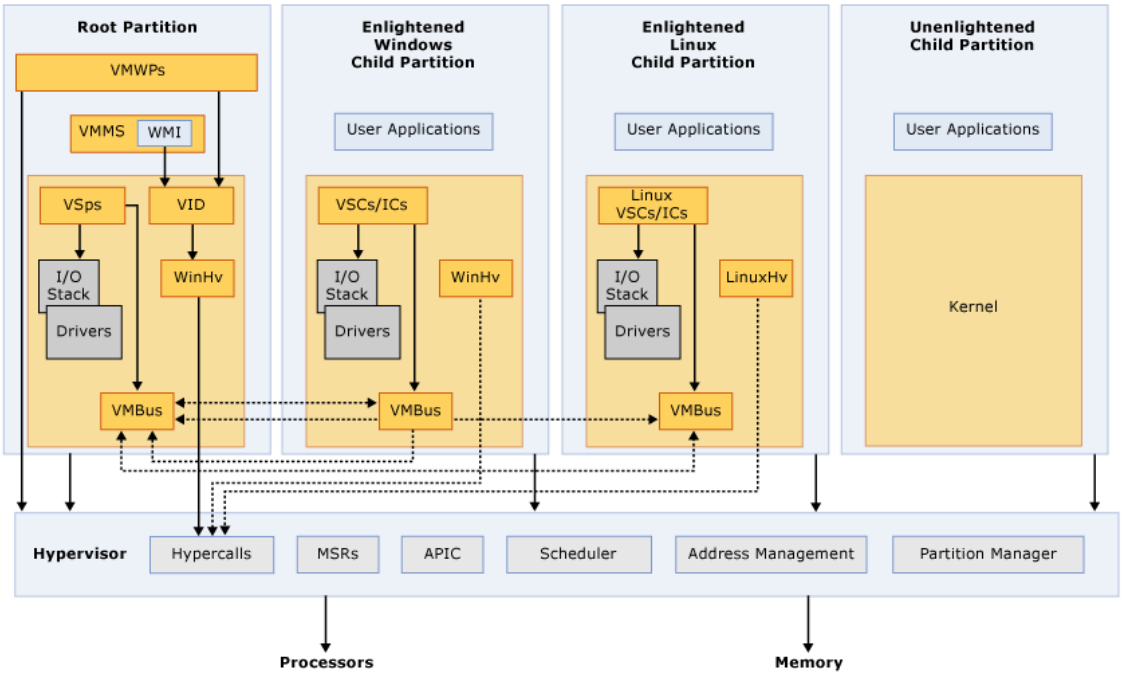

Figure 10. Hyper-V architecture [25]

Hardware resources are seen by child partitions as virtual devices (VDevs). Requests to those virtual devices are handled by the root partition through a VMBus or the hypervisor. The VMBus offers a communication channel between the Virtualization Service Providers (VSPs) hosted in the parent partition, which handle the device access requests, with the Virtualization Service Consumers (VSCs) running in each guest partition. This whole process is done transparently to the guest OS.

Hyper-V requires a processor that includes hardware assisted virtualization, such as is provided with Intel VT or AMD Virtualization (AMD-V) technology.

2.4. Small Cell Technology

The increasing penetration of smartphones and mobile devices, together with higher data-consuming applications, demands higher capacity mobile networks and more efficient radio access networks (RAN). In order to achieve this, operators make use of new technologies such as Long Term Evolution (LTE) and LTE-Advanced (LTE-A) to improve spectral efficiency and look for increasing the density of their networks by deploying small cells thus achieving a larger number of cells per unit area. These solutions help optimizing the network capacity by moving it closer to the users.

Radio networks that combine small cells to complement large macrocells are called Heterogeneous Networks (Het-Nets). Het-Nets make use of different wireless technologies like 3G, LTE, Wi-Fi and WiMAX in a transparent way to users to provide them broadband mobile access.

2.4.1. Small cells

Small cells are considered low-power low-cost infrastructure equipment that can operate in licensed bands of cellular mobile networks. Currently, the concept of small cell has evolved to include a wide range of cell sizes from femto, pico and micro to metro cells. The prefix added determines the costs, size and range of that cell:

- **Femtocells**: they are the smallest range of cells and they are mainly deployed in homes, offices and enterprise environments.
- **Picocells**: pico- refers to a slightly larger range of coverage and therefore these types of cells are expected to be deployed in indoor public areas such as airports, train stations and shopping areas.
- **Microcells**: following with the scale, microcells cover a larger range and they are usually deployed in small urban areas.
- **Metrocells**: they are the cells with highest range in this classification and they are deployed at urban area levels to provide load balancing for macrocells.

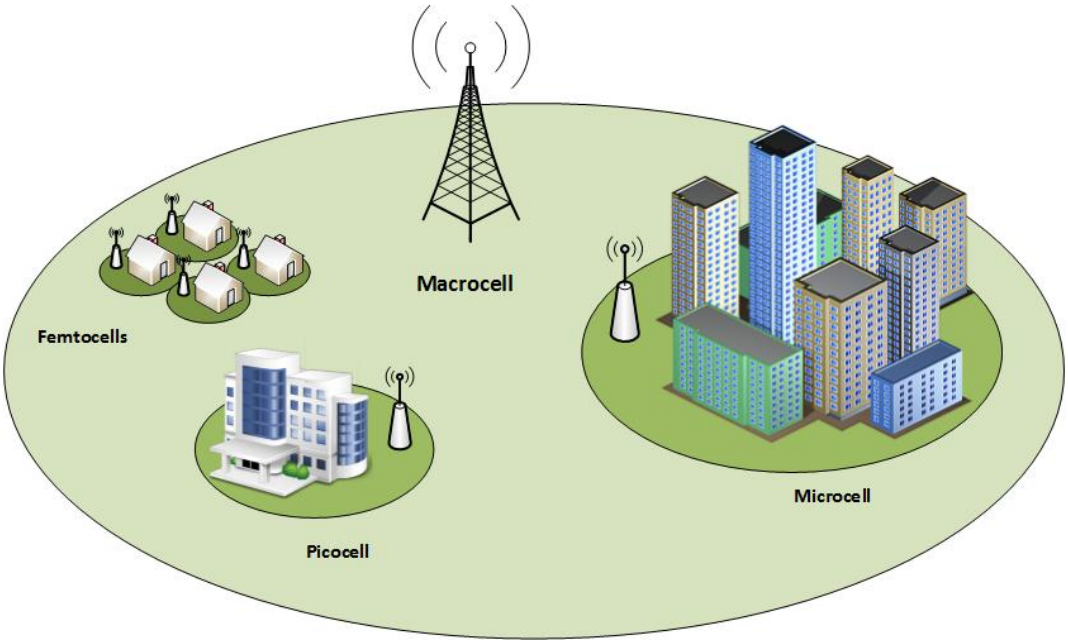

Figure 11. Small Cells coverage comparison.

Independently of its size, small cells are characterized by the following aspects:

- Provide a capacity relative to macrocells.
- Power range from 50mW to a maximum of 40 W.
- They are used to improve network capacity of highly concentrated traffic in indoor/outdoor areas.
- Integrated or external antennas with different directivities depending on the small cell application.
- Use of Multiple Input Multiple Output (MIMO) technologies.
- Automated capabilities such as auto-configuration, location awareness and self-optimizing networks.

Small cell technology allows operators to improve network capacity at a lower cost than introducing more elements to the core. They have great deployment flexibility and being closer to the users provide higher speeds and better performance, which reflects in the offering of a better quality of service.

2.4.2. Small cell challenges

As with every new technology, small cells have to overcome a series of challenges before being completely integrated in mobile networks.

- **Backhaul connectivity.** Due to the small cell deployment characteristics being different to those of macrocells, backhaul connectivity appears as one of the main technical challenges. For residential and enterprise areas, users can bring their own backhaul like DSL, cable or fiber. However, backhaul connectivity for outdoor deployments may result in a bigger challenge since nodes are located in areas where there are not backhaul facilities. In those cases wireless backhaul offers many advantages over wired solutions but radio interferences and efficient radio use have to be considered.

- **Small Cell deployment.** Operators have yet to introduce clearly the benefits of femtocells to consumers and to be ready to invest in this new paradigm, both in physical elements and connectivity technologies. Deployment challenges are also related to identify optimal location for access points in outdoor environments and planning and dimensioning power requirements, costs, environmental impact and previously mentioned backhaul issues.

- **Small Cell mobility.** Mobility in small cell is expected to be high because of its size and the displacement of users. This leads to an increase of the number of handover connections and with it a higher chance of losing connectivity in the process. According to 3GPP Release 11 TR 36.839 studies the outbound mobility from a small cell has a higher failure rate than other types of Het-Net mobility [26].

- **Small Cell interference.** Another key aspect of small cells is the amount of interferences that can be generated in dense cell areas. They are not only important between small cell users but also with respect to macrocell signals that are already present. This presents an environment where interference avoidance mechanisms play a major role in the communications.

- **Capacity management.** Het-Nets propose the traffic load distribution between different radio access technologies in an optimal way. This leads to the necessity of a proper planning of load balancing between small cells and macrocells and being able to dynamically adjust it to avoid congestion problems. Technologies like Wi-Fi can be used as a complement of cellular technologies to improve Het-Nets capacity at the expense of introducing more complexity to the radio management.

- **Security.** Security concerns have also to be address in small cell technology in order to guarantee authorized access to sensitive data that may be transferred to or stored in femtocells. Public areas present a higher challenge since users data may be exposed to malicious users connected to the same cell. Moreover backhaul and wireless connectivity need to be carefully designed to prevent sensitive data leakage.

2.4.3. Small cell ecosystem

The participants in the small cell ecosystem can be classified into equipment vendors, service providers and customers. Vendors supply the equipment that will be used by service providers

to offer different services to the customers. The Small Cell forum is a not-for-profit membership organization founded in 2007 whose purpose is to accelerate small cell adoption and to maximize the potential of mobile services [27]. As today, Small Cell forum includes 70 vendors and 67 mobile operators that represent a 47% of the global total of mobile subscribers across multiple wireless technologies.

Figure 12. Small Cell ecosystem [27]

2.4.4. Small cell market

The market motivation for small cell deployment can be divided in two categories: market drivers from vendor's and operator's perspective and market drivers from users' perspective.

2.4.4.1. Vendor's and operator's perspective

The motivation for vendors and operators is mainly driven by the increase of data rates currently appearing in LTE communications and the expectation of much higher rates for incoming technologies like LTE-Advanced. Small cell will represent an improvement in mobile network's capacity and a better handling of the generated traffic. Furthermore, small cells also allow operators to increase their coverage area to residential and enterprise environments.

Needless to say, the introduction of new elements in the mobile network architecture will also benefit equipment vendors by opening a new device market in mobile system.

2.4.4.2. Users' perspective

From the user's perspective, the small cell market can be considered in two different points of view. The first one refers to small cells that are deployed by end-users in either residential or enterprise environments in order to improve the network coverage area for their mobile devices. In the second one, small cells are deployed by mobile network operators with the purpose of increasing the mobile broadband connectivity, offering new services such as wireless traffic offloading and essentially to improve the overall user experience with their networks.

Cost of small cell deployment will then depend on the specific purpose of the user, although the second approach seems to be the most common way of implementing small cell technology for end-users.

2.4.4.3. Market development

The initial release of femtocells was performed in 2007 by telecommunications operator Sprint with the purpose of improving cellular coverage in home environments. Since then, several operators have participated in the development of femtocell market and the creation of new services. Although residential and enterprise environments are currently the main areas for femtocell deployment, public small cells are expected to enter the market as a complement to macrocell networks.

According to the Informa study included in the latest Small Cell Forum market status report on Q1 2013, small cell revenue is expected to exceed the $20 billion mark [28]. The growth of small cells brings more mobile operators and powers the commercial deployments of femtocells to 46 across 25 countries. Furthermore, the study remarks the increasing interest of operators in public small cell and metrocell deployments. Important network operators worldwide like AT&T, NTT DoCoMo and Virgin Media, have announced plans to deploy small cell and femtocell services to cover dense urban areas.

Finally, Informa survey indicates that the deployment of small cells, including all different sizes, will grow to 91 million deployments, assuring that 98% of operators believe small cells are essential for future mobile networks.

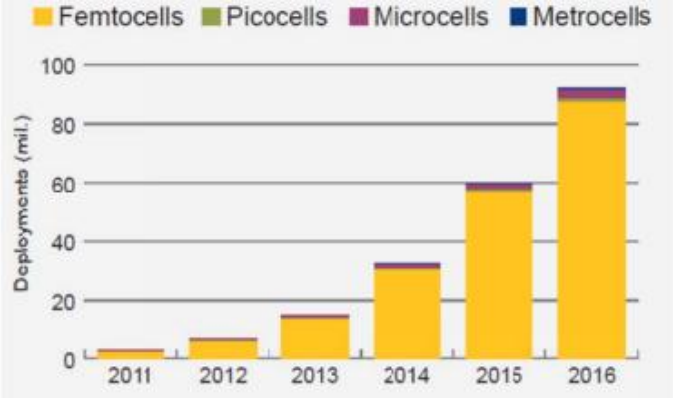

Figure 13. Small cell deployment expectation [28]

2.5. Mobile Cloud

It is obvious that mobile devices are growing at such a fast rate that in less than a decade are nowadays completely integrated in the society. They are used for many different purposes beyond communication like sensing, storing and sharing information, and in a wide variety of environments: social, medical, emergency… However, in spite of the great improvement these devices have suffered in terms of computing capabilities, they may lack the capacity to handle the upcoming requirements of data processing that society is expected to generate. Those requirements range from processing power, internal memory, local storage and even most importantly energy resources.

Cloud-based applications deal with storage and processing limitations by offloading data and computation from mobile devices to the cloud. This approach also eases mobility since data is stored and computed in the cloud and can be accessed from any device with an Internet connection. Leveraging the cloud infrastructure to process and storage data outside the mobile device can be referred as a "mobile cloud" [29]. With the capabilities of mobile cloud, computer intensive applications can be executed on low resource mobile devices.

Currently, there exist a number of mobile cloud applications that connect to a cloud server that is in charge of performing all the computations, leaving mobile devices as simple clients. However, those cloud applications introduce the requirement of having available a good connectivity and do not solve the finite energy limitation mobile devices, a problem that because of the effort of manufactures to achieve thinner, lighter and more compact devices, battery efficiency has become lowered to a second place. On the other side, communications over short distances consume less energy and achieve lower latencies, both essential aspects to be considered. Considering data fee access, latency and bandwidth issues and energy demands of 3G/4G connectivity, local clouds are considered as a better alternative to remote clouds [30].

The term mobile cloud can be associated to different concepts:

- As described before, mobile cloud computing can be referred to execute applications on a higher capacity server while mobile devices act as thin clients. Google's Gmail application is one in a vast number of examples.
- A different perspective considers mobile cloud computing as a computing environment built on a peer-to-peer network of mobile devices where they are used themselves as resource providers [31]. This approach supports user mobility and exposes the potential of these types of clouds to do as well collective sensor data gathering.
- Finally, there is the concept proposed by Satyanarayanan [30] called cloudlet. Cloudlets are computers located close to the network edge acting as intermediary between mobile devices and remote cloud servers. Mobile devices offload the computation tasks to those cloudlets that would have several virtualized resources available to the users connected to them. Cloudlets can be placed in public, residential or office areas in order to reduce network traffic across thin clients and remote servers.

Similarly to cloud computing applications, mobile clouds demand the same requirements such as flexibility, scalability and availability. One key aspect is to determine, according to the needed workload, the number of cloudlet resources to be assigned to the user or to limit the offloading of applications that do not require it because of certain power saving conditions.

This cloudlet interpretation of mobile cloud computing is the one that is considered in the context of this document.

2.5.1. Fog Computing

Related to mobile cloud computing, there is the recently addressed term of Fog Computing. It was originally introduced by Cisco [32] to define the changing paradigm of extending cloud computing to the edge of the network, also known as Edge Computing.

The increasing popularity of the Internet of Things (IoT) and sensor networks where lots of sensors are placed everywhere and send data to processing units, have brought the necessity of having low-latency servers located at the edge of the network in which all the gathered data is processed without the need of using remote cloud servers. Similarly to the concept of mobile cloud, fog computing paradigm provides the same characteristic elements of cloud computing such as distributed computing capability as well as storage and networking resources. Fog computing comprises not only mobile devices but sensor networks, connected vehicles and Smart Grid components [33].

Fog computing characteristics are: being in the edge of the network, having location awareness, providing low latency, mobility support and real-time operations, being geographically distributed and heterogeneous and having connectivity with cloud networks. Whereas cloud computing offers global centralization, fog computing provides low latency localization. Computing resources like processing or storage will gradually be separated from end devices and moved to this new layer of fog computing, making simpler clients that make use of newer

and faster connectivity technologies to communicate with more complex nodes that execute tasks for them.

2.6. Computation offloading methods

An important aspect of Mobile Cloud technology is the capacity to provide offloading methods which allow users to leverage the computing capabilities of the network. Small Cells on the other hand, can be employed by Service Providers to offload network traffic from macrocells avoiding congestion in high demanding situations and therefore guaranteeing a better Quality of Service.

This section is going to put the focus on the offloading methods to increase the computing capabilities of User Equipment (UE). Following the work of [29], computing offloading methods can be analyzed in three different aspects: client-server communication, virtualization and code mobility methods.

2.6.1. Client-server communication

Client-server communication refers to the communication process between a mobile device and the surrogate device or cloudlet in this context. This communication can be done using existing technologies such as Remote Procedure Calls (RPC), Remote Method Invocation (RMI) and Sockets. Although RPC and RMI are long established technologies with well-documented APIs, they require services demanded to be preinstalled in the participant devices, an approach that goes against the mobility and flexible nature of mobile clouds blocking devices that do not have preinstalled those services from using the cloud.

2.6.2. Virtualization

Virtualization is used in cloudlets as in cloud computing, to take advantage of existing resources while maintaining high resource occupation efficiency. Virtual machines OS can be copies of the mobile OS that are instantiated in the cloudlet at the moment of performing the work offload, thus eliminating the need of adapting mobile software applications to run in a different virtualized environment. Those VMs can be migrated between servers without stopping its execution which provides a transparent performance for the user. This way, the mobile device would be connected to the cloudlet through a low latency wireless connection that can be even used for real time operations. When the user is far from the cloud, the mobile device would go back to local computations or continue the offloading if another allowed cloudlet is available nearby.

Regarding the creation of VMs, it can be done by VM migration or VM synthesis. VM migration implies copying and moving the complete VM to a different cloudlet whereas with VM synthesis, virtual machines are created dynamically based on configuration parameters.

Projects like MAUI [34] employ a combination of VM migration and code partitioning with the purpose of saving energy. Developers write specific methods that can be offloaded and if there is any available server when executing the application MAUI decides whether or not offload those methods. CloneCloud [35] approach is based on VM migration as well, but instead of marking the code that can be offloaded as MAUI does, it uses exact copies of the software running on the device to avoid any code modifications. In CloneCloud, a cost function is used to analyze whether it is convenient for the mobile device to offload any task in comparison to the cost of executing the task by itself.

2.6.3. Code mobility

Another approach to offloading is based on code mobility, called cyber foraging. This term was introduced by Satyanarayanan in 2001 [36] referring to the use of available resources of surrogate computers by mobile devices with limited computing capabilities. Scavenger framework [37] employs that concept to create applications that are able to partition and distribute jobs to surrogate computers leveraging parallel computing efficiency and performance. In order to perform the parallel computation, servers only need to have a compatible execution environment to receive and run tasks from mobile devices. By using a cost function on the surrogate speed it can determine if it is convenient for the task to be parallelized, asking the user whether to queue it for later or running it locally in case there were not available surrogates.

Both virtualization and code mobility seem to be more popular methods considering the benefits both offer over RPC methods. Although client-server communications are well known techniques, they require applications to be pre-installed on servers, lacking flexibility. Moreover, the continuous communication between client and server may end up causing network congestion in a multiple device environment. The trend is to go for VM migration, as previous mentioned frameworks do, because it allows an easier offloading without the necessity of rewriting applications to be able to offload tasks. In some cases VM migration may be too expensive in time, for example in the case of the user is in the range of a surrogate just for a few minutes as it is pointed in Scavenger analysis in [37].

2.7. C-RAN

Mobile network operators are starting to deal with several issues derived from the increasing number of mobile users and the high demand of data traffic, which are expected to continue growing at a fast pace for the next years. In first place, as mobile base stations are limited to a number of users, to cope with the high demand of mobile networks operators are forced to continuously deploy new BS. This not only implies deployment investment but also operational costs on power supply and maintenance equipment. Although solutions to reduce the power consumption of base stations, such as turning off selected carriers on idle hours like midnight or

the use of solar, wind and other renewable energy for base station's power supply, are employed in current mobile networks, they do not address the fundamental power consumption problem of the increasing number of base stations. On the other hand, the interference between base stations produced in a dense area where there are lots of mobile users generating high network traffic demand is also affecting the way mobile operators are able to provide their services.

C-RAN, proposed by the China Mobile Research Institute, stands for Cloud, Centralized, Collaborative and Clean RAN (Radio Access Network) and aims to deal with the previous challenges [38]. C-RAN system decouples the radio part (denominated Remote Radio Head, RRD) from the baseband processing unit (BBU) of a mobile Base Station, centralizing the BBU resources into a single resource pool leveraging virtualization solutions, similar to the Cloud computing concept.

This approach offers several advantages. Operational costs of maintaining base stations are reduced since the baseband processing units are all virtualized in a central location. The lower complexity of RRHs makes the deployment of radio elements to decrease as well, allowing operators to increment the coverage area and increase the cell density.

Furthermore, the virtualization of BBUs allows them to work in a cooperative fashion with the possibility of sharing signaling, traffic data and the channel information of the active user equipment (UE) in the system [38]. Centralized BSs offer the additional advantage of easing the application of joint processing and scheduling mechanisms that reduce the problems of inter-cell interference (ICI), such as the cooperative multi-point processing technology (CoMP) in LTE-Advanced.

However C-RAN also has some drawbacks that have to be considered before its adoption. Figure 14 shows the basic architecture of a C-RAN network. RRHs transmit the information sent to and received from the UE to the virtualized pool of BBU by means of fiber connections. The load can be distributed among the different virtualized resources through a load balancer device, allowing a more efficient usage of the total capacity of the system.

The connection of RRH to the BS pool has to cope with the real-time characteristics of the messages communicated through the radio elements. Thus, high-speed connectivity is essential for the fronthaul of C-RAN, demanding fiber deployment investment to link RRH and virtual pool elements. That is why fronthaul connectivity is regarded as the biggest challenge for C-RAN technology.

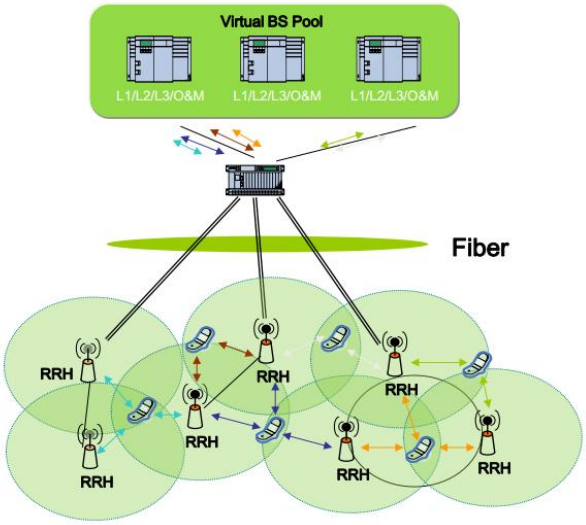

Figure 14. C-RAN architecture [38]

2.7.1. C-RAN vs. Small Cells

C-RAN technology can be seen as a competitor to Small Cells. Both technologies offer opposite solutions to mobile network operators that want to increase the capacity of their infrastructure. C-RAN proposes to move the baseband processing unit to a central location and lighten the radio interface whereas Small Cells opt for keeping the radio and processing elements of a BS together and move them closer to the end user.

When comparing these two technologies it is important to take into account the total bandwidth they require. Small cell data transmitted to the backhaul of the network consist on user data plus some overhead for control and management functions of the base station, while between the RRH to the central pool of C-RAN the data the complete streams of the different MIMO (Multiple Input Multiple Output) antennas. The higher the number of antennas in future base stations, the higher the resulting data rate and bandwidth [39]. Thus small cells demand much lower bandwidth for the backhaul, in the order of Mbps, than the fronthaul for C-RAN, in the order of Gbps, a difference that may be sufficient to tilt the balance towards Small Cell technology.

Each technology has its own benefits and drawbacks and is more suited for different deployment scenarios. Smalls cells will be more cost effective in homes and small offices, while C-RAN may be more efficient in terms of capacity and performance for medium- to large-size buildings and public venues like airports and shopping malls [40]. So, although sometimes seen as competing architectures, the combination of them may result in the best option for high-density mobile networks.

CHAPTER 3
SMALL CELL CLOUD DESIGN

3.1. Description

Heterogeneous Networks (HetNets) are used by network operators to increase the capacity of mobile networks to cope with the increasing demand of traffic. HetNets are formed by the combination of the existing Macrocells with the Small Cell technology, whose main purpose is to allow the offloading of traffic in congested areas.

However, due to their large-scale deployment in random locations, limited transmit power, and the lack of complete coordination, the coexistence and efficient operation of small cells is very challenging [41].

Femtocells are the ideal type of Small Cells for the deployment of the Small Cell Cloud presented in this project because of their proximity to end users. However, to avoid limitations on the scope of the implementation, the term Small Cell will be used instead in the following work.

In this Small Cell Cloud, a user can offload a computationally expensive mobile application to a Small Cell (SC) in order to save energy and processing time, leveraging the capabilities offered by the cloud of SCs. The control and management of the SC Cloud is carried out by a central element in the architecture called the SCM, which also takes decisions regarding VM actions and the application offloading.

The moment a user connects to a SC, a VM is created and associated to him or her, acting as a primary VM to perform the computation of the offloaded application. In case the application requires more processing, the user equipment can request more VM, called secondary, whose purpose is to parallelize the work among them. Because of the cloud aspect of the system, these VMs can be located in any SC currently connected. The decision on where to deploy the VMs, either primary or secondary, is taken by the SCM considering parameters such as the status of the SCs in terms of CPU and memory usage and the latency of the request introduced by the distance of the SCs to the user equipment. Further development of the algorithms of VM deployment is described in later sections of this chapter.

In this section the required components of the Small Cell Cloud are presented, detailing the functions each one need to perform from a high level perspective. Afterwards, the communication between those elements is tackled, introducing a new application protocol specifically designed for this system. Finally, in order for the SCM to know the status of each

connected SC monitoring of parameters are required to be transmitted so several monitoring schemes are presented, analyzing the benefits and drawbacks of each one of them.

3.2. Small Cell Manager (SCM)

The SCM is the element whose job is to act as a central management unit for numerous SC distributed over a wide coverage area. This SC environment, to the contrary of a typical Cloud Computing environment, allows SCs, hosting VMs, to be shut down and turned on at any time so running jobs and VMs themselves have to be dynamically managed. Therefore the SCM has to provide proper mechanisms to be able to manage VMs guaranteeing the best performance in terms of the quality of the service perceived by the end user.

The main functionalities that the SCM has to provide are: VM management (including deployment, migration and destruction), user management, backup and recovery operations, complete system performance monitoring and application offloading and job parallelization decisions. Those functionalities can be divided into control plane and user plane. Control plane groups those related to control and management policies such as the ones required by VMs. User plane is focused on the functionalities related to the user service delivering such as application offloading and Quality of Service optimization.

This section describes the design decisions taken for the implementation of the SCM functionalities of both control and user planes from a behavior perspective. The design has been done considering the following criteria:

- **Dynamicity**. The dynamic changes that can occur in the node elements have to be taken into account. Proper actions should be performed on SC connection and disconnection.
- **Monitoring**. VM management decisions have to be based on the current status of the SCs for which constant and updated monitoring information is required. Initially two monitoring schemes are considered namely Reactive Monitoring and Proactive Monitoring. These schemes will be further analyzed and tested in following sections, as well as possible improvements.
- **Parallelization**. Offloading requests sent by the user equipment to the SCM are processed and decisions regarding VM creation, migration or destruction are made and notified to the corresponding SCs.
- **Control of distributed components**. The SC system described in this project has a distributed architecture. The SCM has to ensure that the user can access his or her VM independently of the actual VM location in a transparent manner for the user.
- **Optimization**. The SCM is expected to manage a large amount of SC nodes to which VM management commands have to be sent or monitoring information to process. Therefore there have to exist optimization mechanisms to avoid performance losses and decrements on processing capabilities on the SCM and the network.

The SCM is modeled in this project as a computing unit running dedicated software, also developed as part of the work. The SCM is built as an aggregation of different modules each focused on a specific task. An overview of the SCM structure is shown in Figure 15 below.

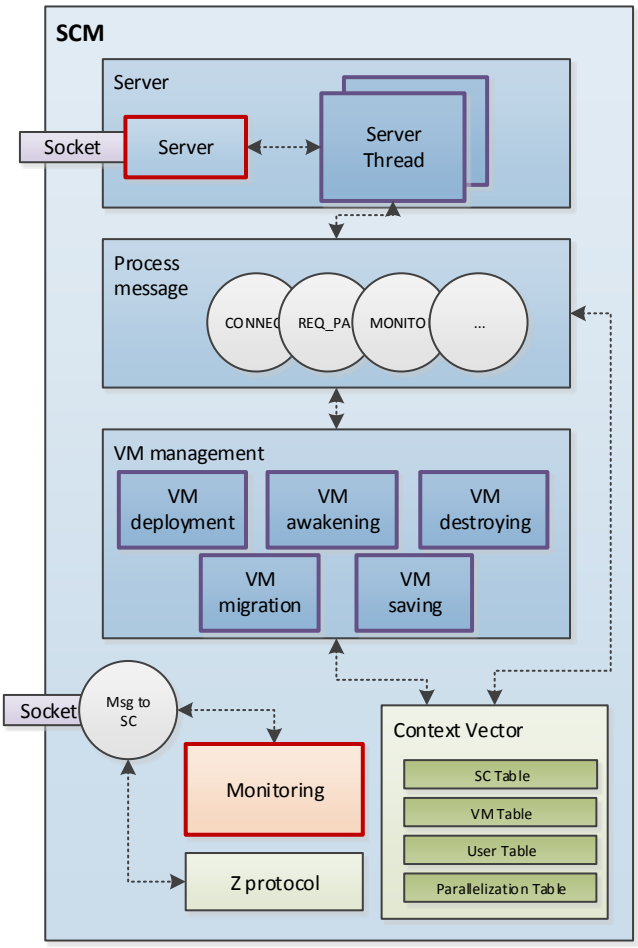

Figure 15. SCM basic schematic diagram

3.2.1. Server module

The server module listens continuously to the incoming network traffic, creating a new parallel task to attend every new stablished connection. This behavior allows the SCM to handle several requests simultaneously. In this project, the SCs are assumed to know the IP address of the SCM previous to the deployment of the system.

3.2.2. Process message module

It receives a request from the Server Module and detects the type of message. Depending on the command it creates a new parallelized task that takes care of the process from that point.

3.2.3. VM Management module

This module groups all the operations related to VM management such as creation, migration and destruction. It consists of five specialized submodules corresponding to each action that ca be executed in a parallel way.

- **VM deployment:** it is in charge of sending the VM deployment message to a specific SC. Primary VMs are deployed when the user connects to the system. Secondary VM are deployed when they are needed for parallelization.
- **VM awakening:** it sends an awakening message to a SC in order to activate a VM that is in a paused state.
- **VM destruction:** this module sends a VM destruction message to a SC. VM destruction can happen upon user disconnection from a SC or the VM is no longer in use.
- **VM migration:** it sends a migration message to a SC indicating that certain VMs have to be migrated to another SC. This can occur when the SC needs more space to deploy a VM for its owner but there are other users connected, or before the SC is switched off, to allow other possible users using a VM in that SC to continue their operation.
- **VM saving:** it sends the VM save message to a SC to save the current status of a VM, allowing cold migration or VM restoration in case of failure.

An important remark is that this module starts the VM deployment procedure but as soon as the message is sent to the SC the thread ends. Later, when the SC has finished the VM action, a message is sent to the SCM notifying that the process has been completed either correctly or not.

3.2.3.1. Service Model

VM deployment and migration decisions are defined in the Service Model. Depending on the configuration, different policies can be stablished to manage the active VMs on the system. The default behavior considered for the VM deployment, migration and destruction decisions is described below.

VM deployment

For deploying VMs, the SCM will first try to do it in the same SC to which the user is connected. If there are not enough free resources to do so, then the SCM will try to free space by checking if any VM in that SC is not being used so it can be either destroyed or migrated to another SC. In case this is not possible, another SC with enough available resources will be selected. And finally, if none of the previous steps can be done, the VM will be deployed in the requesting SC, despite increasing the SC overload probability.

VM deployment algorithm

```
IF the SC to which the user is connected has enough available resources THEN
        Deploy VM on that SC
ELSE IF there are unused VMs in that SC THEN
        IF the user is disconnected THEN
                Destroy unused VM
        ELSE
                Migrate unused VM
        ENDIF
ELSE IF there is another SC with enough available resources THEN
        Deploy VM on first SC with available resources
ELSE
        Deploy VM on SC to which the user is connected (with possible overload)
ENDIF
```

Figure 16. VM deployment algorithm

VM migration

For the migration of VM, the SCM will go through the active SC in the system looking for the ones that have enough available space to host the VM to be migrated. Once a set of candidate SC destinations are selected, the Round Trip Time (RTT) to each of them is calculated so to minimize the migration delay. In case there are no SCs capable of hosting the VM, then migration is not feasible, meaning the system is at its capacity limits at that time and no new VMs can be hosted.

VM migration algorithm

```
FOR each connected SC DO
        IF has enough available space for the VM to be migrated THEN
                Add SC identifier to the candidates list
        ENDIF
ENDFOR
IF SC candidates list is empty THEN
        Migration is not possible
ELSE
        FOR each SC candidate in list DO
                Obtain RTT with a ping request
                IF it has the lowest RTT THEN
                        Select SC as migration destination
                ENDIF
        ENDFOR
ENDIF
```

Figure 17. VM migration algorithm

VM destruction and saving

VM destruction and saving occurs when a user disconnects from the SC or a SC is switched off. A VM is only saved in case the user is the owner of the SC from where it is disconnecting. Otherwise, VMs are destroyed to save space for new users in the SC. When the SC is switched off, in order to avoid work losses active VMs are tried to be migrated. If there are not active VMs or migration is not possible, then VMs are destroyed.

VM destruction/saving algorithm

```
IF user disconnects from SC THEN
        IF user is the owner of the SC THEN
                Save the VM
        ELSE
                Destroy the VM
        ENDIF
ENDIF

IF SC switched off THEN
        IF it hosts running VMs THEN
                Migrate VMs
        ELSE
                Destroy VMs
        ENDIF
ENDIF
```

Figure 18. VM destruction/saving algorithm

3.2.4. Monitoring module

This module has several purposes. First, to process the monitoring messages received from the SCs. These messages are sent periodically from every SC and the SCM needs to keep track of the current status of each of them by updating the corresponding parameters in the Context Vector.

Second, because of the amount of SCs sending monitoring information there can happen that the network or the SCM itself becomes saturated. So, in order to avoid that, the monitoring module can also send monitoring commands to the SCs to adjust the period at which they send the information.

Lastly, because SCs can be switched off at any moment, meaning that there may not have enough time to communicate its shut down intention to the SCM, it is a task for the SCM to periodically probe the registered SCs to know their status. Inside the monitoring module there is then a task that sends ping messages to SCs waiting for them to answer with a ping acknowledgment response. The timing of these message probes is discussed later in this

document since optimization mechanisms have to be taken into account to reduce performance losses on the system.

3.2.5. Context Vector

In order to be able to manage the SC system, the SCM requires to have detailed information of the current status of the connected SCs, the VMs on each SC and the active users. The collection of this information, which is expected to be large, is named Context Vector. The Context Vector contains therefore a snapshot of the complete SC system at a given time and it is constantly being updated with new information. With that information the SCM is able to take decisions and apply different policies regarding VM actions and monitoring optimization.

The Context Vector is organized in four tables covering different scopes, each of those containing the minimum required parameters to effectively manage the complete system.

3.2.5.1. SC table

Stores all the context information related to physical SCs. In particular, the following parameters are stored:

- **SC id.** SC unique identifier.
- **IP address.** Either IPv4 or IPv6.
- **Port.** TCP/UDP port.
- **Status.** ON or OFF.
- **Owner id.** SC owner identifier.
- **CPU number.** Number of physical CPUs in the SC.
- **Memory.** Amount of physical memory in the SC in KB.
- **Disk.** Amount of physical disk space in MB.

3.2.5.2. VM table

Stores the context information regarding VMs deployed in a SC. The parameters this table contains are:

- **VM id.** VM unique identifier.
- **SC id.** SC id where the VM is located.
- **Type.** VMs can be either primary (1ARY) or secondary (2ARY).
- **State.** The current state of the VM. It can be. DEPLOYING, RUNNING, DEFINED, AWAKENING, SAVING, SAVED, DESTROYING and MIGRATING. States ending with "-ing" except RUNNING indicate that a message with the action has been received from the SCM but the VM has not finalized

executing it yet. A VM that has been deployed but has not been launched is considered DEFINED. SAVED status indicates that the VM is not running but has its environment saved from previous execution.

- **IP address.** Either IPv4 or IPv6.
- **VCPU number.** Number of virtual CPUs available to the VM.
- **Memory.** Amount of physical memory in the SC in KB.
- **Disk.** Amount of physical disk space in MB.
- **Priority.** A number between 1 and 10 indicating the priority of the VM inside the SC. A lower number indicates a higher priority.
- **Secondary assigned.** Specific parameter for secondary VMs that indicates if it is assigned to a user or not.
- **Secondary assigned to id.** Indicates the user id in case the VM is of secondary type and is assigned.

3.2.5.3. User table

Stores information related to the users of the system. The parameters are:

- **User id.** User unique identifier.
- **Name.** User name.
- **Status.** Parameter indicating if a user is CONNECTED to or DISCONNECTED from a SC.
- **VM id.** Identifier of the VM of primary type assigned to the user.

3.2.5.4. Parallelization table

In this table the information related to the parallelization requests is stored. This table differs from previous ones in that it stores tuples of two parameters:

- **Parallelization request time.** Time at which a parallelization request is made.
- **Unattended VMs number.** Number of VMs that has not been able to be assigned to that request.

3.3. Small Cell (SC)

The Small Cell is the element located at the edge of the network. The functionality of the SC described in this project is to act as a computing unit which the user can employ to offload

mobile applications. By leveraging the processing capabilities of the SC, the user aims to reduce energy consumption on his or her mobile device while performing computing intensive tasks.

The SC scenario is used to provide connectivity to mobile 3G or 4G networks, acting as NodeB or eNodeB elements of the architecture defined by 3GPP, placed at the user home. However, the role of SCs in mobile networks is out of the scope of the present project, which only considers the distributed cloud perspective it offers and the technical challenges it presents in terms of computing and network management. The integration of the SC element with 4G LTE network is further developed in the FP7 TROPIC project.

From a distributed SC Cloud perspective, the SC offers the following functionalities: act as a VM host executing VM management commands received by the SCM, monitor VMs and process user connection and disconnection sending the appropriate messages to the SCM.

As in the case of the SCM, the design requirements for the behavior of the SC cloud system aim to cover several criteria:

- **VM management**: the SC has to allow local VM migrations to another SC without interfering with the active tasks running.
- **Monitoring**: the SC has to monitor system parameters such as CPU utilization and memory consumption and notify them to the SCM using the proactive and the reactive schemes.
- **Optimization**: sending monitoring information to the SCM by numerous SCs can result in network and computing saturation. This process will require some optimization mechanisms so that the SCM will be able to process each request at the same time the performance of the SC and the VMs running locally in it will not be affected.

In this project the SC is modelled as a processing unit running dedicated software, divided in different modules each with a specific task, similar to those described for SCM design.

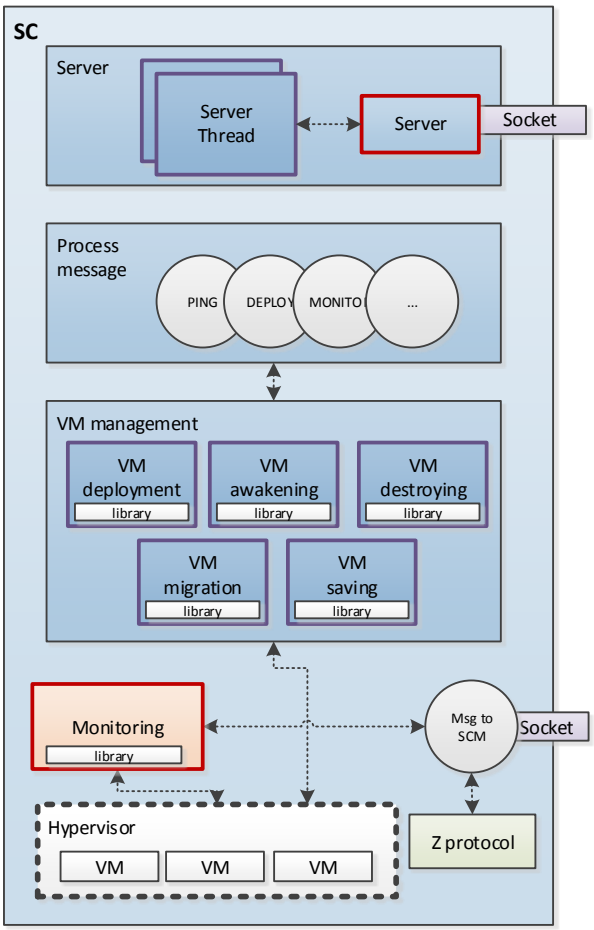

Figure 19. SC basic schematic diagram

3.3.1. Server module

The server part in the SC is equivalent to the server part of the SCM explained in section 3.2.1. There is a server process constantly waiting for incoming requests that starts new threads to attend each request in a parallel way.

3.3.2. Process message module

The process message module is also equivalent to the one in the SCM but in this case the internal functions handle the messages sent from the SCM to the SC. The message is processed and a dedicated thread is created for each type of request.

3.3.3. VM action module

This module performs the actions sent by the SCM regarding VM management. It communicates with the Hypervisor and executes the commands for:

- VM deployment

- VM destruction
- VM migration
- VM saving
- VM awakening

The result of those actions is notified back to the SCM.

3.3.4. Monitoring module

The functionality of this module is focused on obtaining metrics from the SC itself and sending them to the SCM. CPU or memory usage for VMs and the SC is retrieved by doing queries to the hypervisor. Detailed monitoring mechanisms and their benefits and drawbacks are studied in section 3.5.

3.3.5. Hypervisor

The Small Cell Cloud presented in this project is based on the capacity of Small Cells to handle multiple VMs and the SCM to manage their creation, destruction and migration in the fastest way in order to provide the best Quality of Service to the user.

The type of hypervisor that provides the most efficient VM creation and migration at the same time that it provides low computational overhead is a bare-metal or native hypervisor. A native hypervisor fulfils the following requirements:

- Lighter overall system weight.
- Faster deployment of the system at startup.
- Easier virtual machine management from external tools, such as the SCM.
- Easier virtual machine migration.

The bare-metal Xen Hypervisor is the one chosen for the SC implementation. Despite there exists several hypervisor solutions that can cope with the established requirements, Xen Hypervisor offers several advantages:

- It is an Open Source project that counts with a wide and active community that offers support and configuration guides.
- It is fully compatible with the kernel of the most popular Linux distributions, which allows a straightforward installation and setup.
- It counts with several VM management tools and it is compatible with the most popular VIM and virtualization APIs.

The communication between the developed software and the hypervisor will be done by means of a virtualization library which provides an API to execute commands from the code. Further development on this library is done in the corresponding section of the implementation chapter.

3.3.6. Guest VMs

Guest VMs are deployed under the paravirtualization scheme. By letting the VM know that is being virtualized, a faster performance can be achieved since there is no need to hide the system calls to access the hardware.

The UE has been selected to be running Android OS. The reasons for this selection are because by running Android in the VMs the possible mobile application offloading becomes easier since no software is required to translate the applications from the UE to another OS. An exact copy of the OS running in the UE can be placed in the VM so that once a user is connected to the SC and an Android VM is assigned to him or her, any installed application in the UE can be easily transferred to be executed in the VM.

Android OS is being developed as an Open Source project, which allows any kind of required modification for this project as well as having available enough documentation on the web. However, Android OS was initially developed to work over an ARM architecture, which enters in conflict with the actual CPU architecture used in the SC and the paravirtualization capabilities of the Xen Hypervisor. Fortunately, there exists an Open Source project named Android-x86 [42] whose purpose is to provide a complete Android OS implementation for x86 platforms.

3.3.6.1. VM lifecycle

The lifecycle of a VM in the SC is represented in the following diagram.

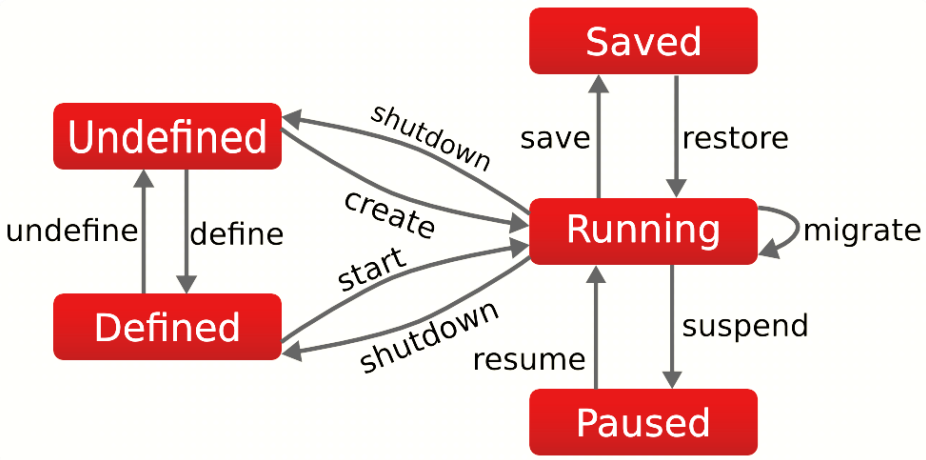

Figure 20. VM lifecycle [43]

A VM is in a *Defined* state when the characteristics are specified to the hypervisor but is has not been started yet. When this occurs, the VM changes its state to *Running*. From here, the VM can

be either *Paused* or *Saved*. The VM pauses its execution in either stated however, pausing the VM keeps its execution temporarily suspended until it is resumed, whereas saving it stores its execution status to a persistent memory. In both cases the VM does not know that its execution has been interrupted. The awakening call is the one in charge of resuming the work of the VM, setting its status to *running* again. Finally, the VM can be set to the *undefined* state, destroying it and its configuration, or again to the *Defined* state, where the VM definition is still stored.

3.3.6.2. VM classification

Guest VMs in the SC are divided in two categories: Primary and Secondary. Each type of VM serves a specific purpose:

- **Primary VM:** these are the main VMs to which the user will connect. They are supposed to be active and running as long as a user is connected to the SC. A Primary VM is uniquely assigned to each user and is in charge of attending the offloading requests. Upon user disconnection from the SC, the action to be done with the associated Primary VM depends on the user being the owner of the current SC or not. In case the user is the owner of the SC to which is connected, the Primary VM can be selected to go in a pause state when the user leaves the SC coverage area so that the reactivation time is faster than in the case the VM would have to be deployed again. On the other hand, if the user is not the owner of the SC, then the Primary VM is destroyed to leave space to other VMs.

- **Secondary VM:** besides Primary VM, specific applications may require additional computation capabilities to work in parallel so a number of Secondary VM can be deployed to help with the offloading work. Secondary VMs are not assigned to any user. They can be deployed in the same SC where the Primary VM is deployed, in different SCs or in any combination of those approaches, depending on the available space and SC utilization. By default two additional Secondary VMs are deployed to cope with the offloading of high resource consuming applications. The decision on where to deploy this type of VMs is taken by the SCM considering the status of the overall system.

3.4. Communication between components

3.4.1. Z Protocol

The Z Protocol is an application protocol defined specifically for the communication between the SCM and the SCs and the latter with the UE. The decision of defining a new protocol over the use of an existing one such as HTTP is carried by the purpose of achieving the following criteria:

- **Simplicity**: the communication between the SCM and SCs only requires short messages with a few set of parameters.
- **Reduced overhead**: messages need to be processed as fast as possible and don't require any kind of extra information for configuration.
- **Use of specific methods**: commands sent by the different elements of the Small Cell Cloud system do not exist as such in available application layer protocols. Besides, they have to be described with simple terminology, following the previous requirements, and therefore need to be descriptive enough to avoid the necessity of sending extra information.

This protocol is used for the communication between the UE and the SC on one the one hand, what is defined as the user plane, and between the SC and the SCM on the other hand, the control plane. The user plane covers the messages sent from the UE requesting a connection to the SC and the ones required to initiate an application offloading. The control plane is related to the VM management messages from the SCM, the connection of SCs to the SCM and the monitoring of SC parameters among others. Some messages, such as the offloading requests, are initiated by the user but then are communicated to the SCM through a SC so that the appropriate decisions can be made, covering both user and control plane communication.

The structure of the Z Protocol messages is made of a command and a set of arguments joined by a separator character as shown in Figure 21.

Figure 21. ZProtocol structure

The components of the message are:

- **COMMAND**: the command word is always at the beginning of the message. It identifies the action of the message and allows the receiver to handle the message accordingly.
- **Separator**: the separator between components in the message. It is defined as a whitespace.
- **Arguments**: the rest of the message includes a set of arguments which represent different parameters or values depending on the different messages. For example, they can be consecutive values of the same type or tuples parameter-value, etc.

An important aspect of the Z Protocol is that it is based on plain text messages. Although privacy is a very important aspect regarding network communications, the encryption mechanisms required to provide such privacy would increase the complexity of both the Z Protocol and the communication process between the SC Cloud components. The implementation of security mechanisms is therefore delegated to lower network layers. Protocols like IPSEC or secure channels using TLS/SSL can be used under the Z Protocol as

protection mechanisms. However, due to the objectives stablished for this project, further analysis on Z Protocol implementation over secure channels are left out for future work.

3.4.1.1. Messages from SCM to SC

The set of commands defined for the SCM to send to SC is shown in Table 1.

Table 1. Messages from SCM to SC

Command	Used when
PING	The SCM wants to determine the state of a SC. If the SC responds then it is considered active if not, it is marked as disconnected
MONITOR	The SCM requests monitoring information of a certain parameter of the SC
MONITOR_ADJUST	The SCM wants to adjust the monitoring thresholds for the threshold-based proactive monitoring
DEPLOY	The SCM commands a VM to be deployed in a SC
AWAKE	The SCM commands a VM to be awakened in a SC
DESTROY	The SCM commands a VM to be destroyed in a SC
SAVE	The SCM commands a VM to be saved in a SC
MIGRATE	The SCM commands a VM to be migrated in a SC

A representation of the commands sent by the SC and the possible responses from the SCM is illustrated in Figure 22. Variable arguments are represented between '< >'. Exclusive options are represented between '{ }'. The text represents the exact words to be transmitted through each message.

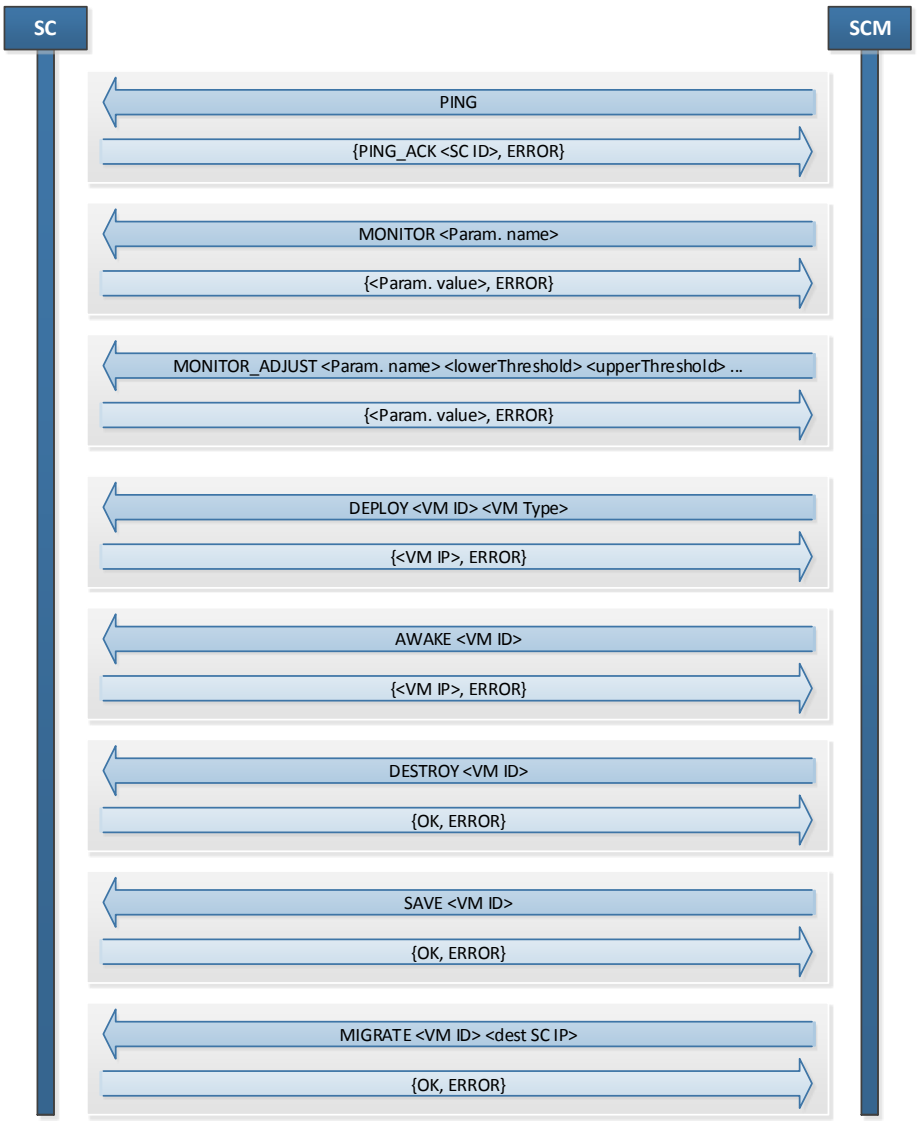

Figure 22. Messages from SCM to SC

As it can be seen in previous diagram, responses do not need a special command word to be sent. This is so because, when received, they indicate that the communication has been done correctly. They can contain either an OK followed by the requested parameter if needed when the processing has been done correctly or an ERROR in case the processing has not been successful.

3.4.1.2. Messages from SC to SCM

Table 2 below shows the defined commands for the SC to transmit to the SCM.

Table 2. Messages from SC to SCM

Command	Used when
CONNECT	A SC is switched on and notifies the SCM

RECONNECT	A SC has lost the connection and tries to reconnect
CONNECT_UE	A user is connected to the SC
DISCONNECT_UE	A user is disconnected from the SC
MONITOR	The SC sends monitoring information of a parameter to the SCM
REQ_VM_IP	The SC asks to the SCM for the IP of the VM assigned to a user
REQ_PARALLELIZATION	The SC requests additional VMs to parallelize the computation
END_PARALLELIZATION	The VMs assigned for parallelization end the task
VM_DEPLOYED	A VM is correctly deployed in SC
VM_AWAKENED	A VM is awakened and running in the SC
VM_DESTROYED	A VM is destroyed in the SC
VM_SAVED	A VM is successfully saved in the SC
VM_MIGRATED	A VM is successfully migrated from the SC to another SC

The diagram, similar to the one shown for SCM to SC messages, of requests and responses for the previous commands is shown in Figure 23.

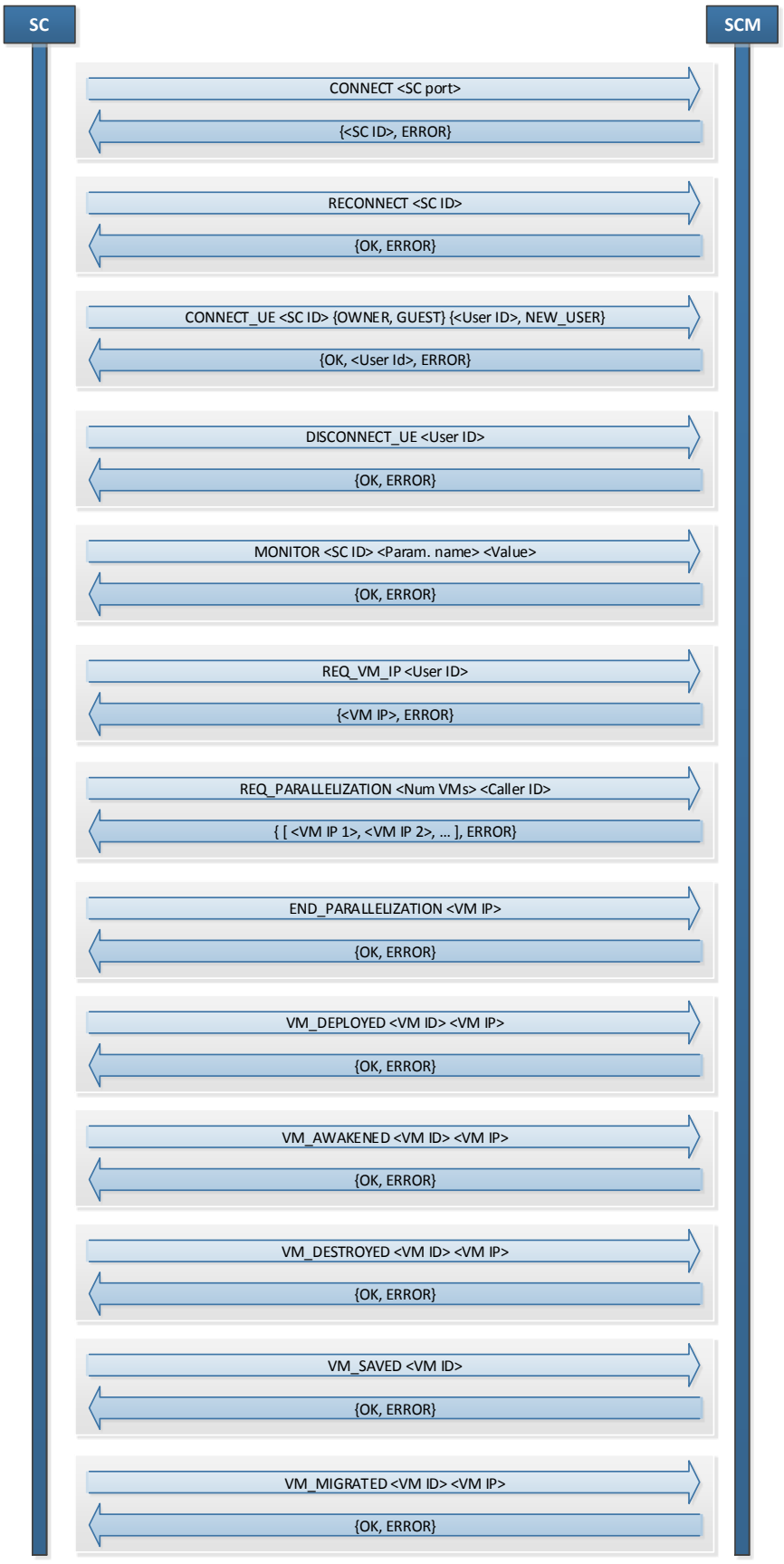

Figure 23. Messages from SC to SCM

3.4.1.3. Messages from UE to SC

Finally, the set of messages that can be sent from the UE to the SC are shown in Table 3.

Table 3. Messages from UE to SC

Command	Used when
CONNECT_UE	The User Equipment connects to the SC
DISCONNECT_UE	The User Equipment disconnects from the SC
OFFLOAD_REQUEST	The User Equipment request an offloading operation

The messages sent from the UE to the SC are redirected to the SCM, which takes the corresponding decisions and returns them back to the SC.

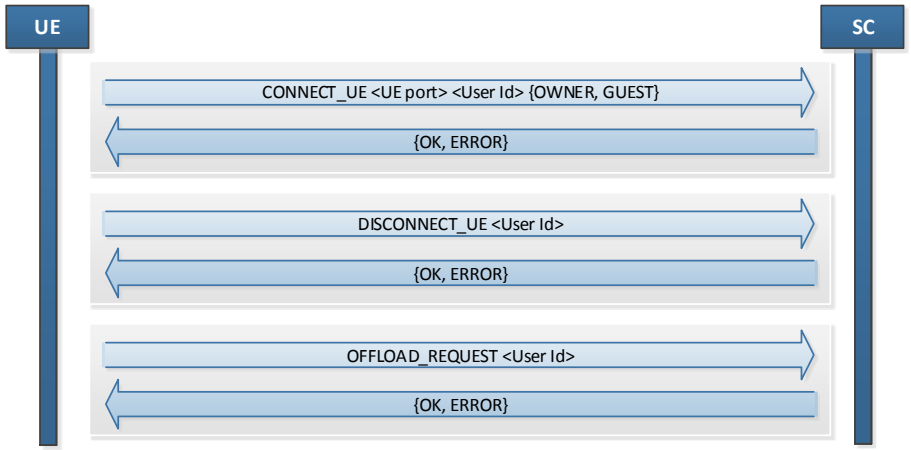

Figure 24. Messages from UE to SC

3.5. Monitoring

Monitoring is a key aspect of the Small Cell Cloud. Because of the whole architecture is being managed by a central component, the SCM, it is essential for this component to have all the relevant information related to the connected SCs at any time in order to take appropriate decisions.

Each SC is required to be able to measure system parameters and send the SCM that information. Although it is possible to obtain metrics from the running VMs, the most relevant parameters that the SCM needs to know in order to be able to take management and VM related decisions, are the CPU and memory usage of a SC.

The state of the SC and its hosted VMs is obtained by monitoring them periodically. Different approaches for the SCM to gather the status information of SC are studied in this section. First,

two basic monitoring mechanisms are presented, namely Reactive Monitoring and Proactive Monitoring, and later, an optimized scheme is proposed whose purpose is to reduce the amount of network traffic and resource usage in the SCM.

SCM decisions depend on the accuracy of the monitoring data on resource consumption on active SCs, meaning that the data stored in the Context Vector have to be as recent as possible. The older the available data, the less optimum configuration of the Small Cell Cloud system can be achieved. The SCM can select a SC to perform a certain action based on the CPU usage value stored in the Context Vector but because of the volatility of that parameter, it may result that an increase of CPU demand appeared since then due to some running processes. Therefore, the workload of that SC will be increased, even saturating it, which will lead to a slow operation and to higher probabilities of VM migration to lighten the work, with the associated overhead.

It is thus essential for monitoring mechanisms to provide the means of first, collecting accurate data and second, avoid resource and network congestion as much as possible that results in lower processing delays.

In order to evaluate the performance of each monitoring scheme, the definition of a computable parameter common to all schemes is required to be defined. For that matter, the term *accuracy* proposed, which indicates how similar the metrics in the SCM are compared to the ones measured by the SC at each instant.

The accuracy on the monitoring data is defined as one minus the normalized root-mean-square error (RMSE) of the difference between the metric measured in the SC and the metric values stored in the SCM over a time period. The normalization of the RMSE is done using the range value of the metric, represented by m_{range} (i.e. 100 as in the case of CPU utilization, or the maximum available memory when monitoring memory usage). Taking m_{SC} as the metric monitored in the SC and m_{SCM} as the metric value stored in the SCM, the accuracy can be computed using the following expressions:

$$RMSE = \sqrt{\frac{1}{T}\sum_{T}(m_{SC} - m_{SCM})^2} \quad (1)$$

$$Accuracy = 1 - RMSE/m_{range} \quad (2)$$

Therefore, the accuracy ranges from 0 to 1. An accuracy value of 0 means that the metrics in the SCM are completely different from the ones measured in the SC whereas a value of 1 indicates that the metrics stored in the SCM coincide with the ones obtained in the SC.

3.5.1. Reactive monitoring

Reactive monitoring refers to a kind of monitoring mechanisms where the metrics are sent by the SC by request of the SCM. In this way, the SCM sends a monitoring request to a number of

SCs, which can vary from a few to all the active ones depending on the situation, and then the SCs answer back with the requested metrics.

The reactive monitoring mechanism leaves the decision of obtaining monitoring metrics to the SCM itself. The main advantage of this approach is the ability of the SCM to have a tight control of the monitoring times, being able to adjust it to gather the information of connected SCs according to its current capacity. For example, the SCM can decide to send monitoring requests periodically to a different subset of active SCs each time in order to avoid saturation when receiving the responses. Or, in the case of a new VM deployment decision has to be made, the SCM can choose to send individual monitoring requests to the SCs starting from the nearest active one and going on progressively sending with farther SCs.

The reactive monitoring mechanism therefore increases the complexity of the SCM since it has to keep track of different parameters such as its CPU utilization and the network delay (e.g. by measuring the elapsed time between a monitoring request and the corresponding response) as well as to implement several decision algorithms to select when and to which SC send those requests. However, this higher complexity in the SCM leads to lowering the complexity of SCs since they only need to respond when they receive the monitoring requests.

The accuracy of the monitoring data stored in the Context Vector depends therefore on the time the SCM decides to request it. The counter part is that requesting the SCs for the latest metrics just before taking a decision will inevitably increase the time to process the action initiated by the user, resulting in a worse perceived Quality of Service.

Figure 25 below shows the basic sequence diagram of the reactive monitoring scheme when the SCM asks for monitoring metrics to active SCs prior to take a decision upon user request.

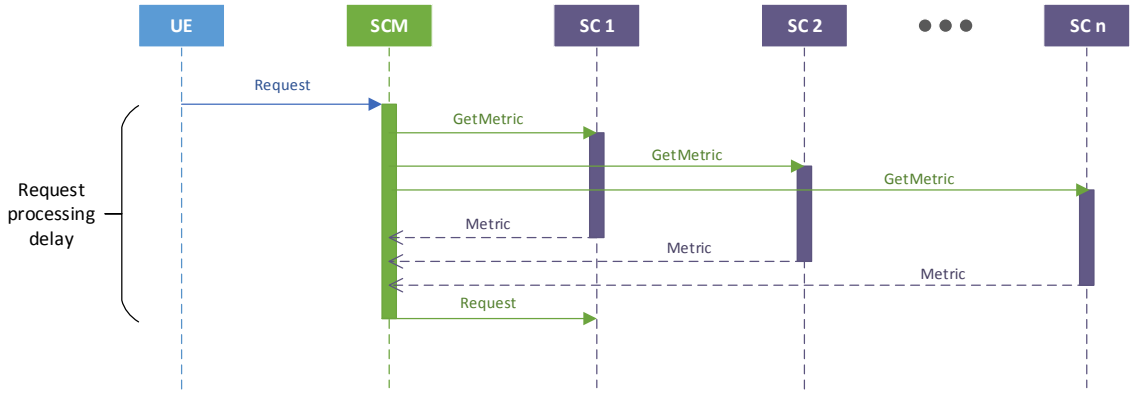

Figure 25. Reactive monitoring sequence

As it can be seen, this mechanism allows the SCM to have updated information to consider when it receives a user request, however, the time needed to query the SCs and to process the request increases.

Another consideration is that this scheme could produce an excessive amount of network overhead dedicated to monitoring since the SCM has to send a request to each SC of which it wants to know the state.

The monitoring overhead that the reactive monitoring may generate can be reduced stablishing a periodic request pattern to different subsets of connected SCs, distributing the monitoring signaling instead of doing bulk requests at specific instants. However requesting monitoring data periodically is not well optimized and it is better implemented using the proactive monitoring.

3.5.2. Proactive monitoring

Proactive monitoring is a monitoring scheme where the monitoring information is sent by the SCs without any previous request from the SCM, contrary to reactive monitoring. Proactive monitoring has the advantage of reducing the amount of network traffic dedicated to monitoring since no requests are needed to send the metric information to the SCM. Due to the communication being initiated by the SC, proactive monitoring also improves the network and computing resources usage in comparison with a reactive monitoring in which the SCM sends monitoring requests periodically.

In this scheme, there can be distinguished two different approaches: periodic proactive monitoring and threshold-based proactive monitoring.

Despite of the monitoring communication being initiated by the SC, both proactive monitoring schemes require configuration parameters that need to be defined by the SCM. This allows the SCM to adjust the amount of monitoring traffic coming from SCs depending on the network load and the amount of connected SCs.

3.5.2.1. Periodic proactive monitoring

In this approach the SCs send monitoring information to the SCM periodically. The sending period can be determined depending on the characteristics of each SC and by grouping SCs by area, so that the messages are evenly distributed in time.

The basic implementation of this monitoring mechanism is that the SCM fixes the monitoring period of each SC when they are first connected. In order to do that, the SCM evaluates the characteristics that the SC sends in the connection request and the current state of its own capacity. For example, for a high capacity SC the monitoring period may be higher than for a SC with lower characteristics since it may potentially be saturated sooner.

However, the network and the SCM state can be highly dynamic. Because of that, fixing a monitoring period on each SC based on an instant decision turns out in a poorly optimized solution. A better solution is to allow the SCM to dynamically adjust the monitoring period of each SC. The period can be reduced in case there are few active SCs or a SC has very low

computation workload, so that more metric information is sent to the SCM, which benefits their accuracy and improves the decision algorithms.

Allowing a dynamic period change means to increase the complexity of the SCM since it has to be aware of not only its current capacity, but the monitoring period and the capacity of each connected SCs, all that information being stored in the Context Vector. An adjusting algorithm has to evaluate those parameters and then notify the change to the SCs. SCs on the other hand just need to take care of the monitoring period to send the corresponding metrics so overall, the required complexity of the proactive monitoring scheme is not very high, resulting more affordable in terms of computation requirements than the reactive monitoring scheme presented before.

One disadvantage this approach presents is the lower accuracy on monitoring metrics that can be obtained in average. In order to attend the periodic receiving of metrics, the SCM has to evenly distribute the monitoring times which can lead to higher delays between metrics of the same SC.

Figure 26 shows the basic functionality of the proactive monitoring scheme.

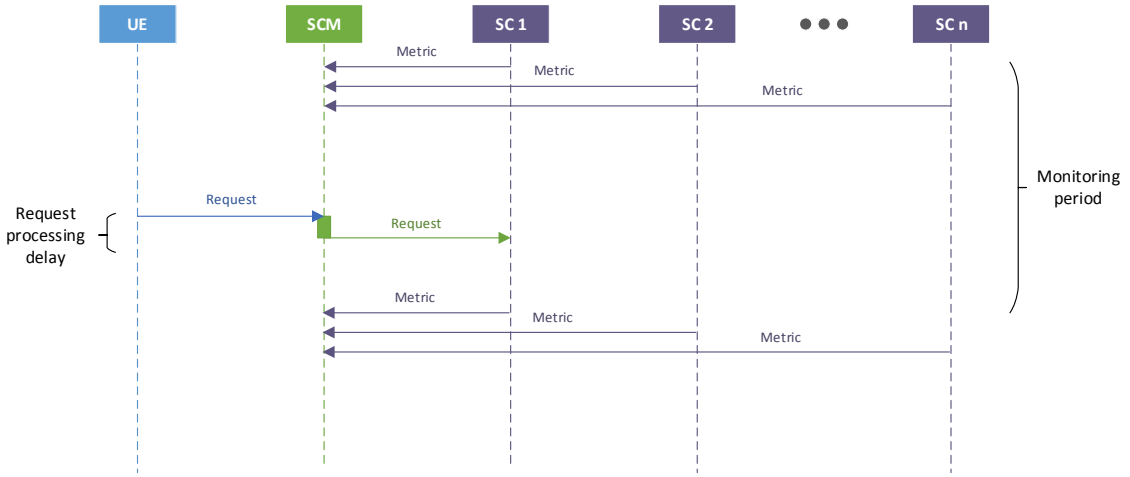

Figure 26. Proactive monitoring sequence

As it can be seen, the processing time of a user request is lower than the one achieved with the reactive monitoring but the metrics are not as accurate, which can lead to more VM migration or period readjustment because of not good enough decisions in the SCM.

3.5.2.2. Threshold-based proactive monitoring

This approach comes as an optimized version of the periodic proactive monitoring. As described before, sending metrics periodically exposes few disadvantages: using a fixed period reduces network and processing overhead but also reduces the accuracy of the metrics because of the dynamic nature of the system; using a dynamically adjusted period would solve the low accuracy problem but at an expense of increasing the network and processing overhead.

With threshold-based proactive monitoring, the SC only sends monitoring information when the metrics are out of a given threshold. This threshold is set by the SCM depending on both its current load and the network status. If the number of connected SCs is low, the SCM can set a low monitoring threshold of each SC. This will cause metrics to be sent more frequently, which will improve the accuracy, at expenses of a higher workload. On the other hand, the accuracy can be lowered in those cases when the SCM is under heavy workload or the number of active SCs rises.

The threshold is stablished over and under a given value. Metrics (m) are therefore sent whenever the following expression becomes true:

$$|x_c - m| > x_{th} \quad (3)$$

Where x_c is the central value around which the threshold (x_{th}) is stablished. Figure 27 represents graphically this mechanism. The blue line corresponds to a possible metric change over time. As long as the metric value is inside the region delimited by the lower and upper thresholds, the metric is not sent to the SCM. The red mark indicates the value that will be notified to the SCM.

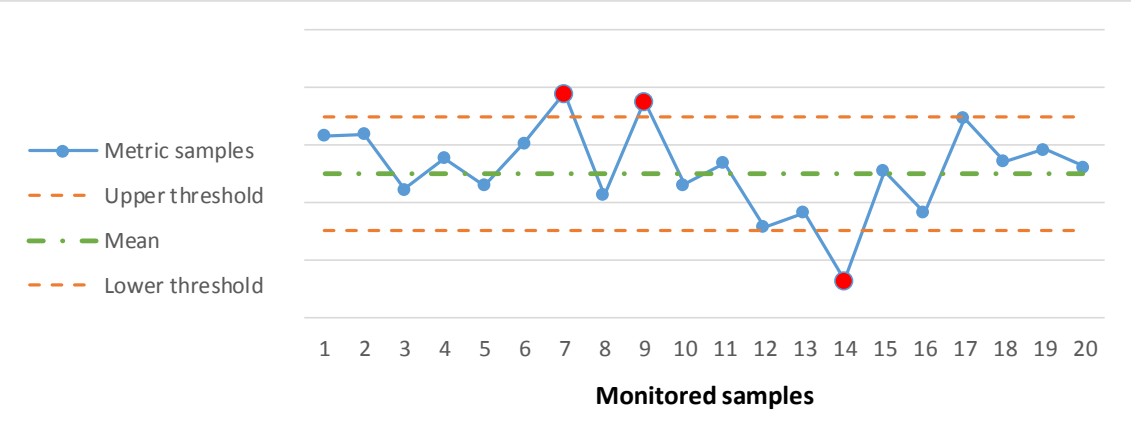

Figure 27. Threshold-based proactive monitoring example

Because of the variability of the different metrics, it is important to stablish a method to update the central value around which the thresholds are stablished. The central value can be determined using different parameters, which will also impact on when the frequency metrics will be sent.

- **Mean metric value**: the central value is the mean of the *n* previous metrics. This approach makes the central value to slowly change despite of highly variable parameters. The speed of change can be adjusted with the number of metrics to average. A higher number makes the mean to change slowly whereas a lower number makes it to follow high changes more quickly. Figure 28 shows an example of this method using the CPU usage percentage as the monitored metric. In this case, the mean is computing taking into account the last 10 metrics. The red marks are the metrics that would be sent to the SCM following this mechanism of threshold update.

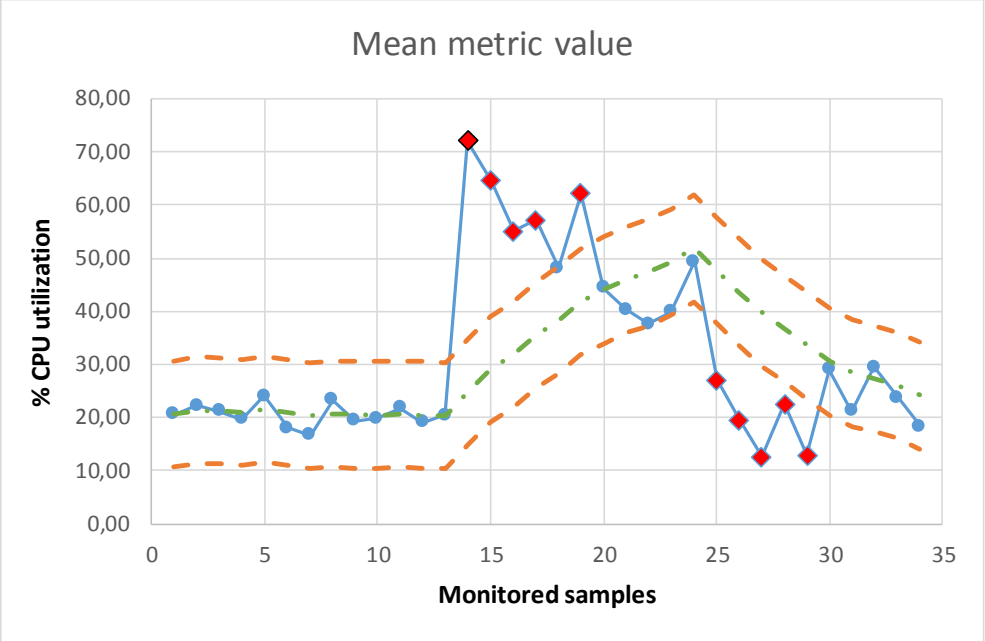

Figure 28. Threshold-based proactive monitoring, mean metric value approach

As it can be seen, for a highly variable metric pattern, taking into account a high number of previous metrics for determining the mean value leads to a higher number of metrics notified to the SCM.

There also exists the possibility that by using the mean value a situation could occur where, for example, the number of metrics to average is high and the metrics are increasing but so slowly that they do not fall out of the threshold. This would make the mean value to increase accordingly and because the metrics would be within the threshold, they would not be notified to the SCM, what would cause the current metrics in the SC to be quite different from the ones that the SCM has and therefore leading to a high loss of accuracy. To avoid this kind of situation, narrower thresholds can be selected.

- **Last sent metric**: in this case the central value corresponds to the last sent metric. This method avoids the accuracy problem that can arise using the mean as a central value because in this case the central value is notified to the SCM whenever it changes. Therefore, it responds faster to highly volatile metrics. The drawback is that if the thresholds are too small, it may result in high network and processing overhead. An example of this kind of threshold establishment is shown in Figure 29 where a CPU usage percentage is represented, with red marks showing the metrics that would be sent to the SCM.

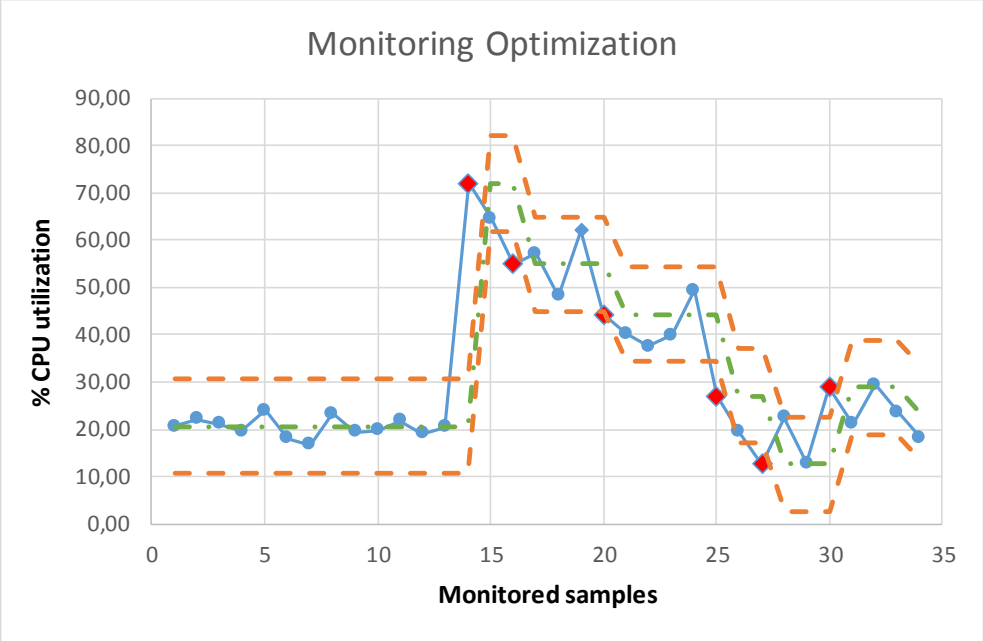

Figure 29. Threshold-based proactive monitoring, last sent metric approach

Compared to the previous method of updating the central value, the last sent metric approach provides higher adaptability to abrupt changes without requiring the adjustment of any parameter. In the case of the mean metric value approach, the adaptability to changes directly depends on the number of previous metrics to compute the average, the higher that number, the lower the thresholds change.

CHAPTER 4
PROOF OF CONCEPT IMPLEMENTATION

In this chapter the implementation details of the system are going to be described. It will be divided in the SC and the SCM components, explaining at a lower level the decisions that have been taken to achieve the functionality detailed in previous chapter.

The programming language used for the development of the code to model the Small Cell Cloud system has been Java SE8. The reasons for having chosen Java are the following:

- System implemented based on parallel processing. The different modules inside both the SC and the SCM components have been decided to be implemented as threads and Java provides an easy to use API to work on threads.
- Multi-platform language. The Java Virtual Machine (JVM) allows running Java code in both Windows and Linux.
- Widely used. Java is a very popular programming language and it also counts with a very rich community.
- High level of knowledge of the language. Java has been one of the primary programming languages throughout the career and during the time working in Atos so this provides a great advantage over other languages regarding the efficiency and organization of the code.

4.1. SCM Implementation

The code for the SCM has been structured in two main parts: the core functionality, which includes server functions, message processing and decision making algorithms, and the monitoring module, in charge of dealing with the monitoring-related computations. Common to both parts it is the Context Vector, which can be accessed by any SCM module.

As it has been described before, the SCM has to perform several tasks in parallel such as attending network requests, computing management decision algorithms, notifying SCs,

analyzing monitoring information… Therefore, that behavior has been implemented using dedicated threads for each task.

In order to achieve the best performance when dealing with multiple threads, it has been decided to use thread pools whenever possible. Thread pools create a predefine set of threads waiting for new tasks to be executed and can also include a queue for those cases when there are more pending tasks than available threads. Due to the expected amount of threads running in the SCM, this approach reduces the time consumption required to create and destroy threads every time, which impacts positively in the overall performance of the code.

The class diagram of the overall SCM implementation is shown in Figure 30.

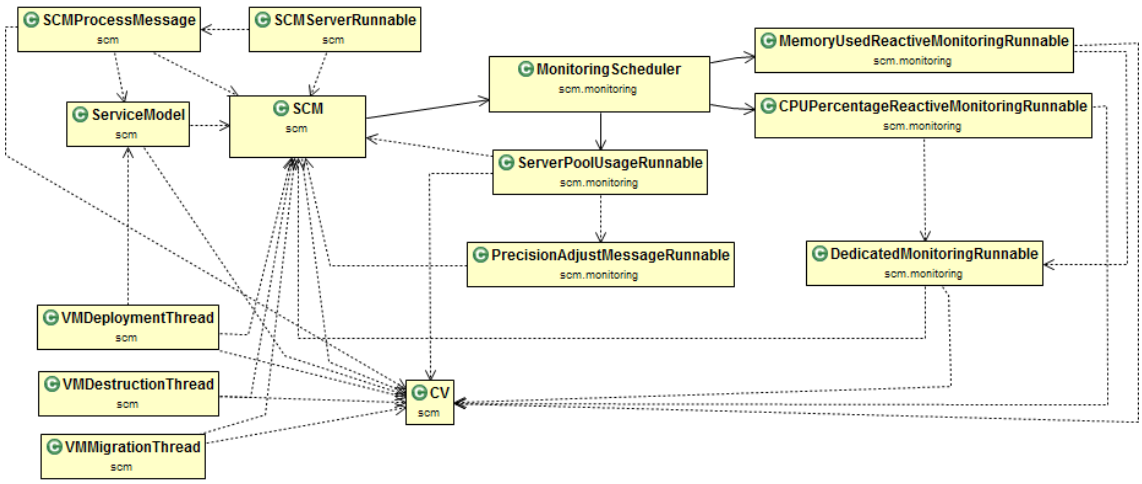

Figure 30. SCM class diagram

4.1.1. Main thread

The SCM is launched using the main thread. This thread first initializes the monitoring elements and then creates a pool of threads that will be used by the server module. The thread waits until a new request is received, for which the server module is called and the whole process is initiated. Parallel to the initialization of the server module, the monitoring module is started though the MonitoringScheduler class and a console control panel are launched. The control panel can be used to display the status of the SCM, the number of SCs connected and active VMs as well as the number of requests received and responded.

The functionality of this block is implemented in the SCM class.

The class also contains several static functions that can be used by any SCM module, being the most relevant the one in charge of sending a message to a SC. Every communication that starts in the SCM is sent through that function.

Several configuration parameters such as the listening port and the server thread pool size and string constants used as keywords across the modules are set in this class as well.

The default pool size to attend server requests has been set initially to 1,000 threads, using an unbound queue for the extra tasks. According to the Java API documentation, this approach is appropriate when each task is independent of others, such in a web page server, which is what occurs with the SCM. An infinite queue implies that no extra threads are created and it may end up growing uncontrolled if the requests are more frequent than processed. However, the time to process those requests by the SCM is expected to be low so an unbounded queue should not cause any issue.

4.1.2. Server module

The server module has been developed using a thread that takes a socket as a constructor argument and afterwards processes its content. The functionality of this module is implemented in the SCMServerRunnable class.

The actual server thread is queued to the thread pool created in the main thread, which remains idle waiting for network requests though a server socket. When a new connection is stablished, the server thread is executed and reads the contents received through the socket as a string. Then splits the received string using the separator character defined in by the Z Protocol, i.e. a whitespace, and calls the process message module with the array of received words. The result given by the process message module is sent back though the socket to the client.

As soon as a request is received, a new server thread is taken from the pool, leaving the server module ready for a new request.

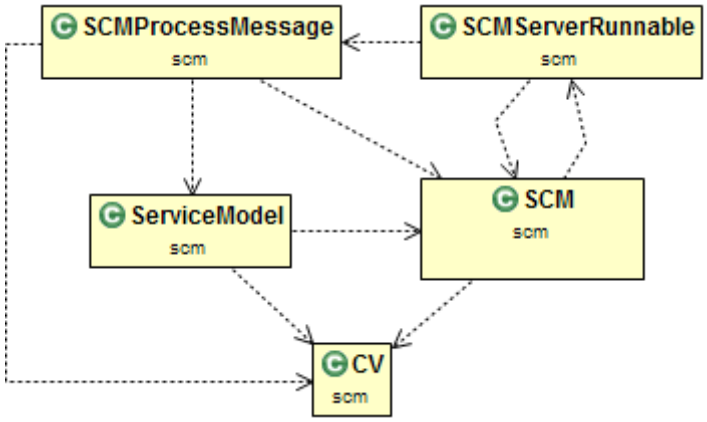

Figure 31. SCM Server and Process message modules class diagram

4.1.3. Process message module

This module has been implemented in the SCMProcessMessage class which contains a static function in charge of parsing the received message and performs the corresponding operations. It checks the first element of the array sent by the server module looking for a Z Protocol operation keyword. The Context Vector is updated accordingly to the received message, returning *OK* if everything has been done correctly or *ERROR* in case there has been an error.

When further actions beyond a CV update need to be taken, such as VM deployment, migration or destruction whenever a user is connected o disconnected for example, the VM management module is called.

4.1.4. VM Management module

The VM management module consists on several independent threads implemented as VMDeploymentThread, VMDestructionThread and VMMigrationThread, each one dedicated to a specific VM task, i.e. deployment, destruction and migration respectively. These threads are started from a function inside the main SCM thread. All of them have the same structure: A set of parameters is received depending on the action to be done, such as the type of VM, the SC identifier and the user identifier. The corresponding queries and updates to the VM table inside the CV are done if needed and then a message is sent to the SC with the VM command.

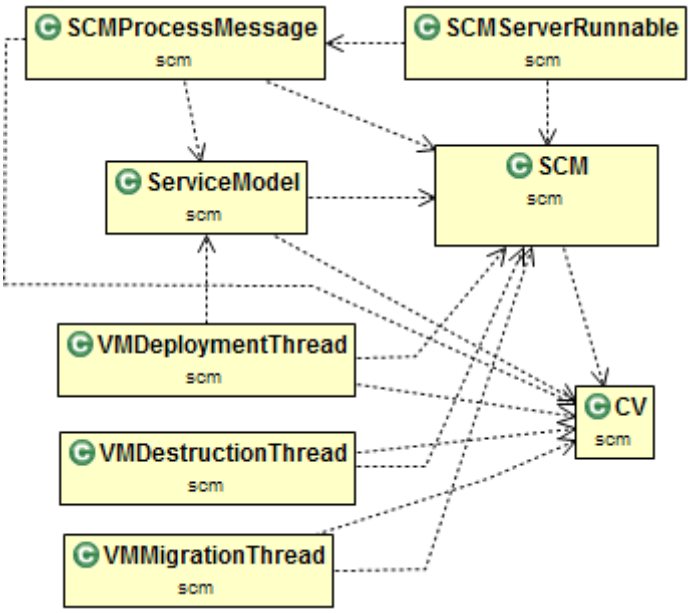

Figure 32. SCM VM Management module class diagram

The selection algorithms of SCs for VM deployment and VM migration are implemented in a specific class called ServiceModel, to which VM management threads access to get the best possible SC target.

By default, whenever a new SC connects to the SCM, two secondary VMs are commanded to be deployed. The following time diagram represents the SC connection process along with the VM deployment request.

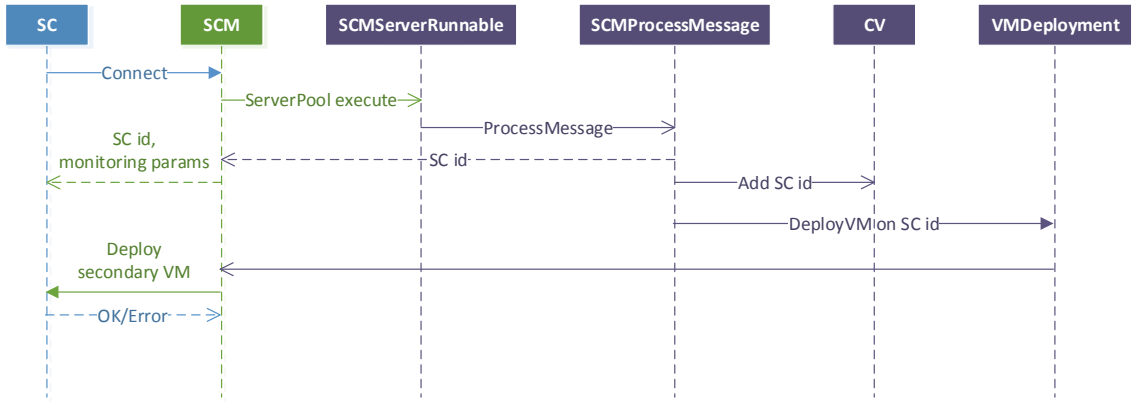

Figure 33. VM deployment time diagram

When the SCM receives the connect message from a SC, it sends a SCMServerRunnable task to the server pool which afterwards calls the process message module. This module parses the connect message, generates a new SC identification number and updates the CV to finally call the send message function of the SCM class to notify the SC. Parallel to that process, a VMDeployment thread is created which will be in charge of requesting the SC to deploy the default number of secondary VMs.

4.1.5. Monitoring module

Two different approaches have been taken into account for the monitoring module implementation. On the one hand, the components required for the reactive monitoring, and on the other hand the ones required for the proactive monitoring.

For both monitoring schemes, the module is composed of different classes, being the central component of this module the MonitoringScheduler class. This class creates another dedicated thread pool to process the messages that need to be sent to SCs and starts different periodic monitoring threads depending on the selected scheme. It also contains the configuration parameters and initialization variables.

The class diagram of Figure 34 shows the elements that compose the monitoring module.

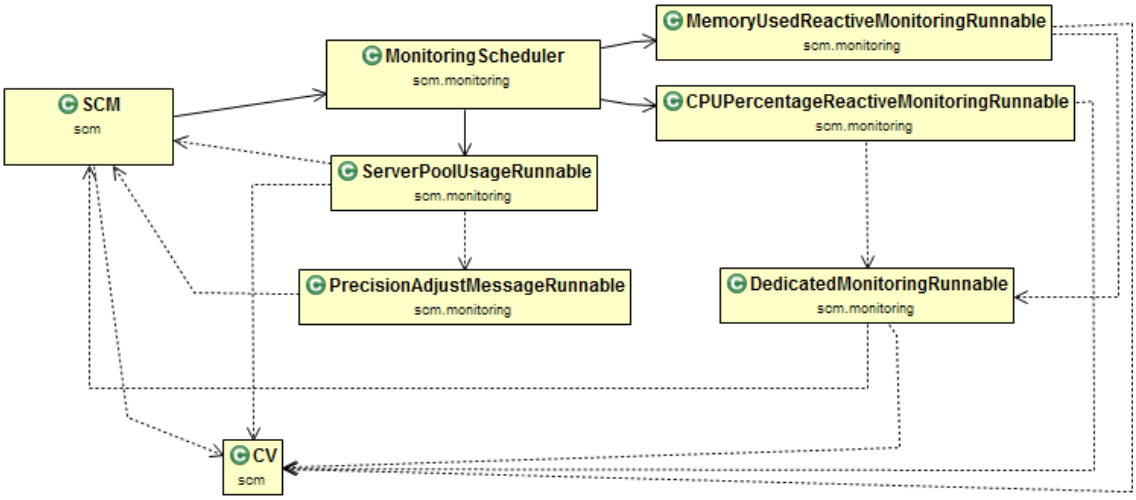

Figure 34. SCM Monitoring module class diagram

For the reactive monitoring scheme the Monitoring Scheduler starts two different threads, one for each received metric from the SCs: CPU and memory usage. These threads run periodically throughout the CV checking the difference between the time of a stored metric and the current time and. If that difference is higher than the configured value, a metric request task (i.e. a DedicatedMonitoringRunnable) is sent to the send message thread pool that will later ask the corresponding SC for metrics of such parameter.

When proactive monitoring is selected, the monitoring scheduler launches a different periodic thread, ServerPoolUsageRunnable, whose purpose is to check the utilization of the server thread pool and keeping it at a desired level. If the usage is under a determined level, a set of new tasks responsible of narrowing the monitoring thresholds or reducing the sending metric period of the SCs are sent to the send message pool, depending on which method to adjust the thresholds is active. This way, SCs will send monitoring information more frequently thus increasing the server pool utilization. If the utilization level exceeds another level, the same procedure as before is done but this time indicating the SCs to expand their monitoring thresholds or increase the sending metric period to reduce the utilization.

The server utilization depends not only on the received monitoring requests but on any other incoming requests from SCs. Therefore, the low and high usage levels can be adjusted to maintain a more or less constant utilization depending on the overall performance of the SCM.

4.1.6. Context Vector

The Context Vector acts as a database where a snapshot of the current state of the system in stored. It has to show the status of every registered SC, either active or inactive, the connected users and available resources on each of them. Furthermore, the number of running VM with their status and characteristics, as well as the connected users have to be available in the Context Vector for the SCM to perform better management decisions.

All this information have to be retrieved and stored very fast to improve the time the SCM employs on communicating with the CV, which will impact on its overall performance.

There exist two main approaches when considering where the CV is located: either local to the SCM or on a remote server. Both configurations offer benefits and drawbacks. Placing the CV in the same machine offers speed improvements to access the database when compared to the remote server configuration, reducing also the amount of network traffic. On the other hand, the remote CV reduces the storage requirements on SCM and offers the possibility of having a central server where all, or several, SCM across a wide area can access and store common information. For the present work, only a SCM is going to be deployed so the benefits a client-server configuration provides are not relevant. It has been considered more important to improve the read and write speed of the CV, therefore selecting the local database approach.

Three different database implementations for the CV have been considered. The first one is the use of a hash table [44]. Hash table is a common data structure whose main characteristic is the searching speed: it keeps a constant lookup complexity, independently of the number of entries. The Java API provides various implementations of hash table structures, namely HashMap [45] and ConcurrentHashMap [46] classes. Both allow common operations such as insertion, selection and key or element existence checking, but HashMap implementation is not thread-safe whereas ConcurrentHashMap is. Avoiding thread synchronization makes HashMap perform better, however, because of the different SCM modules being implemented as threads, it is a requirement for the CV to be accessible concurrently without damaging its integrity. A ConcurrentHashMap object avoids read locks which offers read improvements over older concurrent hash table implementations such as HashTable.

The second option is the use of an in-memory database, such as HSQLDB (HyperSQL DataBase). HSQLDB is a relational database written in Java that offers in-memory and disk-based tables in both embedded and client-server scenarios [47]. The connection between the database and the Java code is done using the JDBC (Java Database Connectivity) API. Thanks to the in-memory tables and its multithreaded execution, HSQLDB provides very fast operations with the management capabilities defined in the SQL Standard.

The last option is to use a RDBMS (Relational DataBase Management System) like MySQL. MySQL is developed by Oracle as an Open Source project and it has become the most popular Open Source database according to the website [48]. It provides a fast and scalable database able to work on client-server as well as on embedded systems. Furthermore, MySQL offers lots of capabilities in terms of data support, security and scalability making it one of the most complete database systems. There also exist multiple programming language connectors to work with MySQL.

Comparing the available options to implement the CV in the SCM some conclusions can be extracted, which are summarized in Table 4. In terms of overall performance, i.e. the speed of read and write operations, using the ConcurrentHashMap object is the best solution. It is stored in memory and avoids the overhead of relational databases. This has some drawbacks, as the integrity checks have to be done by the SCM itself, instead of the database management system.

However, because there is no relation between SCs, this should not be a problem. According to [49], HSQLDB speed of operations is really high compared to the one achieved by MySQL, mainly because of the in-memory tables and multithreaded operations.

In terms of functionality MySQL offers the most complete solution, closely followed by HSQLDB. Both offer the power of SQL queries and extra functions such as security and database consistency checks, something that cannot be leveraged with a ConcurrentHashMap object.

The memory overhead that each solution provides is also relevant. In both ConcurrentHashMap and HSQLDB there is a tradeoff between speed and memory usage, since the complete database is stored in memory. MySQL on the contrary, uses disk images, which reduces the speed but also reduces the overall memory consumption. HSQLDB also offers the possibility of using disk files to store the database.

Finally, because of the in-memory versus disk storage of the tables, it is important to note the persistency of the data in case of a failure. In this case, MySQL and HSQLDB using disk storage provide the means to recover the data in case of power failure whereas some extra mechanisms would need to be implemented to store the state of the ConcurrentHashMap object.

Table 4. CV database technology comparison

	ConcurrentHashMap	HSQLDB	MySQL
Performance	Very high	High	Medium
Functionality	Low	High	Very high
Memory usage	High	Very high	Low
Persistency	Low	Low	High

Considering the functionality of the SCM, the most important parameter to select the database implementation has been decided to be the performance. The faster the speed of operations, the more SCs and user requests can be attended. On the other hand, the simplicity of the CV as it has been previously described makes the extra functionality provided by relational database systems not so important, prioritizing the simplicity of the implementation over the available features.

Because all of this, the final decision has been to use the ConcurrentHashMap for the CV implementation, sacrificing memory usage for speed improvement.

The complete CV implementation is carried over using nested hash maps so that a hash map containing all the information related to a SC for example, can be stored in a higher level hash map using the SC identifier number as the key.

When dealing with hash tables, as ConcurrentHashMap is, rehashing is the most time consuming operation. If the number of entries in the table, that is, the load factor, exceeds a certain value, the hash table becomes more inefficient due to a high number of collisions and this is when it is necessary to perform a rehash. Rehashing creates a larger table and remaps the

elements of the old table into the new one using a new hash function. Thus, it is important to guess the number of elements a hash table is going to store in order to minimize the amount of required rehashing operations.

ConcurrentHashMap class, as every class that implements the Java Map interface, allows passing initialization parameters to the class constructor regarding the expected capacity and minimum load factor before rehashing is done. As described in the Java API documentation, the default load factor of 0.75 offers a good tradeoff between time and space costs, so this is the selected value. To avoid rehash operations, it is sufficient to select an initial capacity higher than the expected maximum number of entries divided by the load factor. The number of SCs connected to the SCM in the implementation is expected to be quite large (around 10,000), however, this is just an initial guess and further adjustment of this parameter could be needed once the complete system is deployed and tested. In any case, it is suggested in the Java API to create the table with sufficiently large capacity so that the mappings can be stored more efficiently instead of performing automatic rehashing to grow the table.

Because of the concurrency nature of the selected hash table implementation, one more parameter, namely concurrency level, has to be set. The concurrency level indicates the number of threads that are expected to be accessing the hash table at the same time, so that the Java internal implementation can divide the table to allow that number of threads to operate without blocking. The Java API recommendation is to choose a value to accommodate as many threads as will ever concurrently modify the table, avoiding significantly higher or lower values to reduce wasting space and time. The default concurrency level is set to 16, which seems a good number for the SCM operation, considering the amount of modules and checking operations. As in the case of the initial capacity of the hash table, this number may require further adjustments once the system is tested.

The initialization parameters of the ConcurrentHashMap class are therefore:

- **Initial capacity**: 10,000
- **Load factor**: 0.75
- **Concurrency level**: 16

4.2. SC Implementation

The implementation of the different SC modules is structured similarly to the one of the SCM: the core functionality includes the communication server as well as the message processing whereas the monitoring schemes and decisions are grouped in the monitoring module. The VM management that allows the communication with the underlying hypervisor completes the structure.

Due to the same reasons of multiple parallel tasks described in the SCM implementation, the SC also uses dedicated threads for each task, making use of thread pools whenever possible as a way to increase the performance of multiple threads.

Figure 35 shows the class diagram of the complete SC implementation.

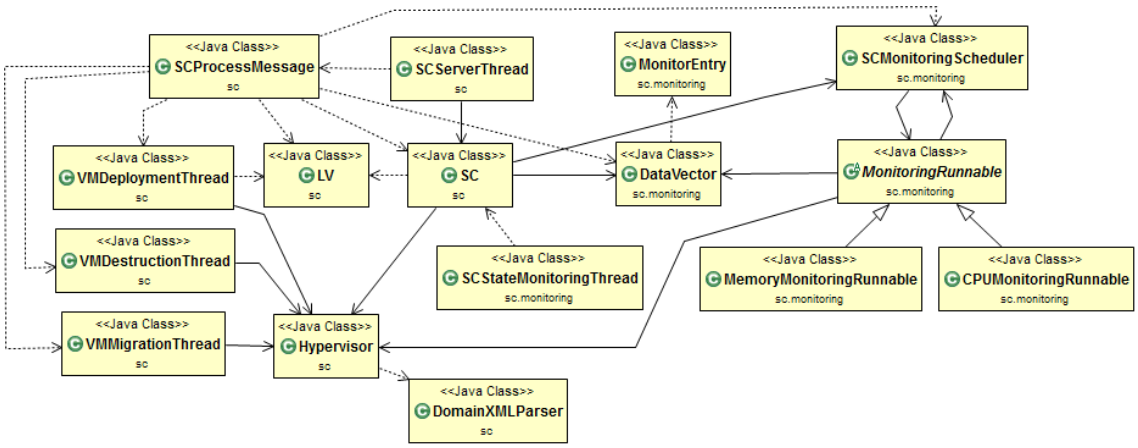

Figure 35. SC class diagram

4.2.1. Hypervisor configuration

An essential component of the SC structure is the configuration of the hypervisor and how the different modules are connected to it.

The host OS has been chosen to be Ubuntu version 12.04 LTS. Although Xen Hypervisor offers compatibility with a variety of OS, the selection of Ubuntu has been taken due to the compatibility and easy installation process of the Xen Hypervisor. Since Ubuntu version 11.10 the default kernel included in Ubuntu can be used directly with the Xen Hypervisor as the management (or control) domain (Dom0 or Domain0 in Xen terminology). Thus, the hypervisor installation is as simple as executing an "apt-get install" command and updating the GRUB system booting options. The installed version is Xen 4.1.6.1.

Following the Xen recommendations for the configuration of the host OS [50], a fixed amount of 2 GiB or RAM are reserved for the dom0, since the Linux kernel takes the amount of memory available at boot time to calculate various network related parameters and the needed memory to store memory metadata. The second reason is Linux needs memory to store the memory metadata (per page info structures), and this allocation is also based on the boot time amount of memory.

4.2.1.1. Networking

The network configuration for the SC has to allow that the VMs can be accessible directly from the outside in order for the UE to offload an application. Therefore, a virtual bridge has to be setup in the physical network interface of the host OS so that VMs can obtain its own IP. This virtual bridge also allows the direct communication between VMs.

4.2.2. Guest VMs

The allocated RAM for each VM is 256 MiB which is the minimum amount recommended according to the Android-x86 project website, which is also enough for the purpose of the SC Proof of Concept behavior. The VM configuration file is described in the next section following the required pattern for the virtualization API.

4.2.3. Libvirt API

Due to TROPIC project requirements, the selected API for the communication between the Java code and the Xen Hypervisor is Libvirt.

Libvirt is a virtualization toolkit that provides a common interface to many different hypervisor solutions among which Xen Hypervisor is included [43]. This virtualization API allows an easy access to the hypervisor in a lightweight manner directly from the host domain itself, avoiding the installation of a more complex solution such as XenServer, which includes the XAPI toolstack (the one that is used by Virtual Infrastructure Managers like OpenStack and OpenNebula), and uses RPC (Remote Procedure Call) communication between the client and the hypervisor.

Libvirt offers both a Command Line Interface (CLI) called Virsh as well as bindings for several programming languages such as C/C++, Python and Java, being this last interface the one chosen for the implementation of the SC. The Java binding implements all the basic functionality for configuring and managing the VMs although it is possible to achieve even more functionalities through the Java Native Interface to make use of the native C libraries. For the tasks that the SC has to perform on the hypervisor, i.e. VM creation, destruction and migration, the Java API is enough and there is no need to use native functions.

Libvirt uses the XML format for the virtual domain definition which is then translated to the specific Xen configuration file. Since the VMs that will be deployed by the SC will contain the same configuration, a generic XML domain definition file is used as a template where only two fields are modified for each VM. The XML template can be seen next.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<domain type="xen">
  <name>android0</name>
  <memory unit='MiB'>256</memory>
  <vcpu>1</vcpu>
  <bootloader/>
  <os>
    <type>linux</type>
  </os>
  <clock offset="utc"/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <disk device="disk" type="file">
      <source file="/home/tropic/android/android.img"/>
      <target dev="sda1"/>
    </disk>
    <console type="pty">
      <target port="0" type="xen"/>
    </console>
    <interface type="bridge">
      <source bridge="xenbr0"/>
      <mac address="00:aa:bb:cc:dd:00"/>
      <script path="/etc/xen/scripts/vif-bridge"/>
    </interface>
  </devices>
</domain>
```

Figure 36. XML Domain definition template

Guests VM are called domains in the Libvirt documentation so the configuration file defines a Xen-type domain where the sub-element tags are self-explanatory. As it can be seen, each guest domain has 256 MiB of reserved memory (MiB referring to mebibytes, that is, 2^{10} bytes, in order to avoid confusion with the megabyte unit used to express 10^6 bytes), and a single virtual CPU. Regarding networking, as it has been commented before, all VMs are set to work under a bridged configuration, thus allowing each one of them to be accessible directly from outside.

The only two parameters that differ from VM to VM are the name of the domain and the MAC address. The name ends with a number which is used as the VM identifier in order for the SC to better keep track of the available VMs. Similarly, the last two digits of the MAC address are set as well to that numeric identifier. This results in having a VM with identifier number of, for example, 4 being named as *android4* with MAC address *00:aa:bb:cc:dd:04*.

4.2.4. Main thread

The code implementation of the SC follows the same patterns than the SCM. Each SC is launched as an independent thread. As soon as the main thread is launched the monitoring module is started and the server module is activated and left listening on a port for incoming connections.

The SC constructor takes the listening port of the server module and the IP address of the SCM. It also allows different ways to create the SC object by means of two Boolean configuration parameters. The first one is used to print log messages on console about the sent and received messages whereas the second parameter indicates the SC whether to use the Xen Hypervisor or a dummy hypervisor used to test different parts of the code from a different machine with no hypervisor installed.

In this thread all the required configuration parameters common to the different SC modules are also set. Thread pools are also leveraged as much as possible to reduce the time used to create and destroy threads as it is done in the SCM. The SCs are expected to be receiving a good amount of messages from the SCM, mainly related to VM management and monitoring. Therefore, as a starting parameter the thread pool size is stablished to 1000 threads.

Furthermore there are implemented two functions to act as a common point to initiate the communication with the SCM and the UE.

A block diagram of the implemented modules is shown in Figure 37 below.

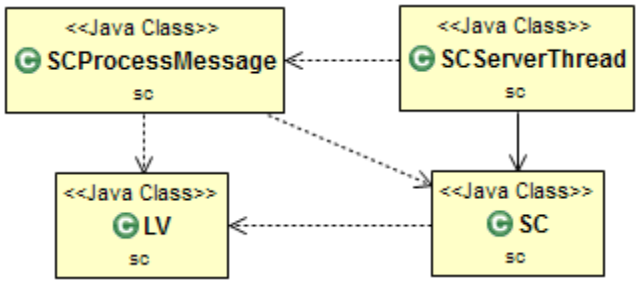

Figure 37. SC Process Message and Server modules class diagram

4.2.5. Server module

The server module implemented in the SC is exactly the same as in the SCM. It takes a socket as a constructor parameter and passes to the Process message module the input stream. The result of the message processing is then returned to the other end through the same socket.

The server module threads are executed using the thread pool configured in the SC main thread so that multiple connections can be attended in a parallel way.

4.2.6. Process message module

This module checks the command passed from the server module along with its corresponding parameters and performs the appropriate action: either passing the task to the VM management module, the monitoring module or sending a notification message to the SCM in case of a connection message is received from a UE.

4.2.7. VM Management module

The VM management module in the SC is composed by three different threads, each one in charge of performing a specific action regarding VMs: deployment, migration and destruction. The communication with Xen Hypervisor is done by each thread through a hypervisor class where there are implemented methods that use the Libvirt API. This hypervisor class connects to the Xen Hypervisor using the virtualization API at the start of the SC main thread if the actual hypervisor has been configured to be used during the SC initialization.

For the deployment of a new VM, the XML configuration template is loaded into memory and the name and MAC fields edited to include an assigned VM id. A single call to the domain creation function Libvirt provides it is then done, checking if there is any error during that process. In that moment, the SCM is notified indicating that the VM is being deployed. Once the VM is active and running a new message is sent to the SCM with the information update, so that it can be taken into account for any offloading decision.

The destruction of a VM follows a simpler process. Whenever the SC receives a message form the SCM indicating the destroy command followed by the VM identifier, the VM destruction thread is launched and performs the domain destruction call in the hypervisor class.

Due to the limitations in terms of hardware to have several hypervisors running, the VM migration is simulated by changing the SC to which belongs on the CV and adding a 10 seconds delay as an estimated value.

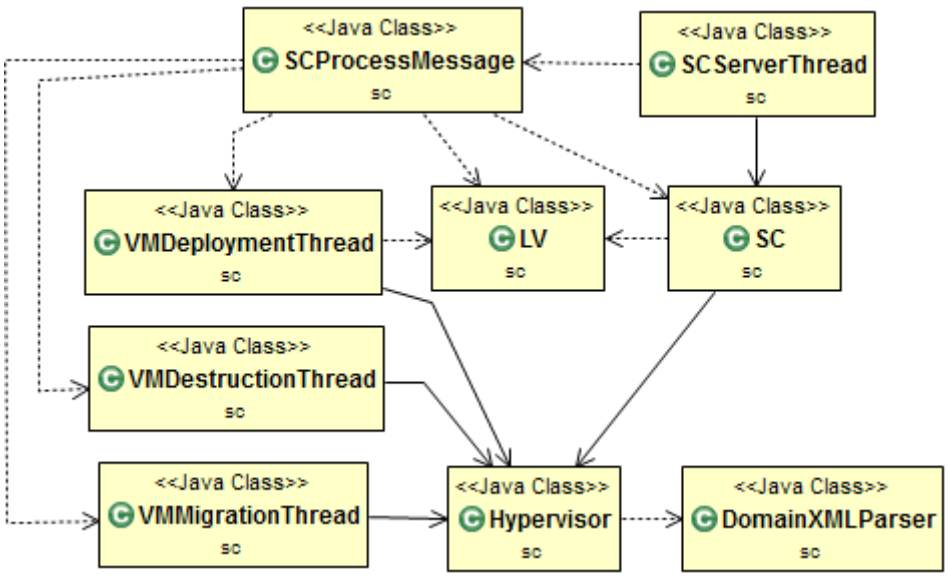

Figure 38. SC VM Management module

4.2.8. Monitoring module

The monitoring module is composed of several classes. The central element of the module is the Monitoring Scheduler, which is responsible for starting and setting the time period at which metrics are going to be retrieved from the hypervisor. The relevant metrics for monitoring are the CPU usage percentage and the amount of memory in use in the SC. There is a dedicated thread per each metric, set to be periodically retrieving the metric values from the hypervisor. The metrics, represented by the MonitoringEntry class, are stored in a circular buffer of configurable length implemented in the DataVector class. By default is set to store the last 5 metric values for each parameter.

Depending on the monitoring scheme selected to be active at the moment of the SC creation, the monitoring threads execute different functions. If reactive monitoring is active, every time the SC receives a monitoring request of a certain parameter from the SCM, it retrieves the last stored monitoring value of the buffer and sends it back.

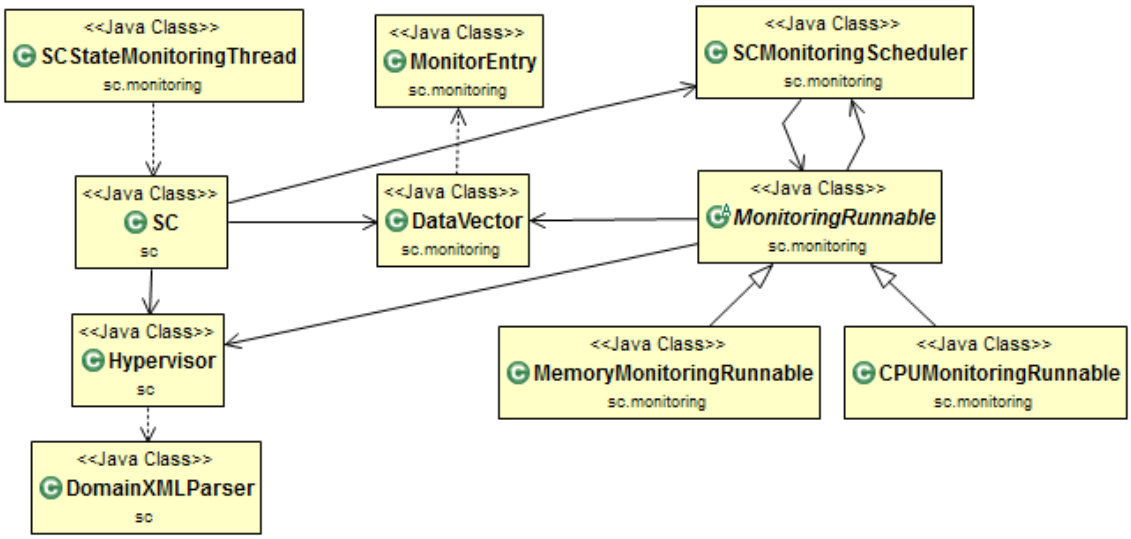

Figure 39. SC Monitoring module

For the proactive scheme, because of the less optimization level of the periodic monitoring, the threshold-based option is the one that is implemented. Whenever a metric is retrieved, the monitoring thread checks if it is inside the threshold and sends a message to the SCM in case it lies outside. The central value around which the thresholds are stablished is updated taking into account either the mean value of the metrics stored in a circular buffer of fixed size or the last sent metric.

In order to allow more flexibility for the threshold-based monitoring, the possibility of adjusting the upper and lower thresholds independently is also implemented. Thresholds are updated by command of the SCM. The SCM analyzes the load in its server module and adjusts the SC monitoring thresholds accordingly with the corresponding message. The thresholds are increased or decreased by a fixed value configured in the SCM.

A default threshold distance is defined in the SC which can be overwritten by the SCM at any time.

4.3. Communication

The communication between the components of the SC Cloud has been developed using sockets. Java sockets provide a two-way link to connect two different programs across the network following the simple client-server model, which is the one that is used in this project. The server socket in the server module of each component listens to incoming connections on a specific port. On the other side, the client socket initiates the communication with the server using the IP/Port pair. Once the connection is stablished, both client and server can read from and write to the socket any data.

Sockets open a TCP connection for each message. Although the UDP protocol generates lower network usage because of the less required packets to send the information, TCP has been chosen in order to ensure the reliability of the connection. Since TCP is a connection-oriented protocol, it is easier to detect network failures and disconnections from SCs. As soon as the SCM tries to connect to a SC socket and that SC is unreachable for any reason, an exception is raised and proper actions can be taken, such as updating the SC to a disconnected status.

4.4. Unit testing

The development of each module and functionality for the SCM and the SC has been tested independently to assure that each piece works as expected. The methodology that has been followed has consisted in designing unit tests that focus from very specific parts of the code to a concrete functionality such as the deployment of a primary VM when a new user connects to a SC.

The tests have been done using the JUnit test framework [51]. Most of the tests classes contain a setup method which is started before running any test, usually dedicated to launch the SCM main thread for testing both the SCM and SC communication on the same machine. After that, a series of test functions are implemented to check that specific parts of the code work properly, e.g. information is correctly inserted in the CV, correct messages are sent and processed between the SCM and a SC or monitoring modules can communicate with the Xen Hypervisor through Libvirt. In order to check that the code operates correctly, JUnit assertions are placed after the code under test. JUnit assertions compare the result of the code execution with what it is expected to return. If every unit test succeeds then the code is set as verified.

The functionality checks covered by the tests include:

- **SC connection to the SCM**. Starting a SC to check if it initializes correctly and can reach the SCM either located in the same machine or another.

- **Communication between components and correct message processing on each component**. Once the SC and the SCM can reach each other through the network, a series of messages are sent between them to see if the responses are the expected ones. Different message parameters are sent as well to test the parsing at the other end.

- **Thread pool creation**. Starting both the SCM and a SC to test if thread pools associated to the server module and the monitoring module are started and work correctly, having the specified number of threads available without crashing.

- **Communication with Xen Hypervisor in the SC**. A series of calls to the hypervisor are tested using the virtualization API provided by Libvirt library.

- **VM management**. Actual VM deployment and destruction is tested through the Libvirt API. For these specific tests, the native tools that Xen provides were used to actually check that the VMs have been correctly deployed and destroyed. Migration tests were not possible due to the lack of more PCs in which install more Xen Hypervisors.

- **Monitoring modules**. The different monitoring approaches are tested in the SCM and a SC. For the reactive monitoring, the SCM has to send the monitoring request to the SC and the latter has to answer with the metric value of the required parameter. For the threshold-based monitoring first the mean metric value method is tested using a well-known set of metric values in the SC so that metrics that fall out of thresholds are known beforehand.

- **User connection to a SC**. This test checks that when the UE sends a connection requests to a SC, the message is notified to the SCM which updates de CV and sends a primary VM deployment command to the SC.

- **Use of Service Model algorithms in the SCM**. The VM deployment decisions taken by the SCM are tested to be done accordingly to the defined algorithm in the Service Model.

- **Data insertion to and retrieval from the CV**. By inserting and retrieving data from the CV to ensure it works correctly.

- **Parallelization request from the UE**. With this test a parallelization request with the number of desired VMs is sent from the UE to the SC and then redirected to the SCM, which will command one or several SCs to deploy that number of secondary VMs to attend the parallelized job.

Once the basic functionality of the different implemented modules is tested with the unit tests, a series of stress tests are executed to check the performance of the system under heavy load. Those tests include the capacity of the SCM to handle simultaneous connections, the performance of the CV for a high volume of data manipulations and the VM deployment delay

on a SC for multiple user connections. The results of these stress tests are analyzed in the following chapter.

CHAPTER 5
PERFORMANCE EVALUATION

This chapter presents the obtained results from the stress tests done to the complete system and the outcome of several simulations to compare the different described monitoring schemes.

The stress tests are used to put the system under heavy conditions to evaluate how it performs and what its limits are. For those tests, two identical laptop computers were used, one running the SCM code and the other having installed the Xen Hypervisor where simulated SCs connected to the SCM were launched. The characteristics of the laptops are the following:

- CPU: Intel i5
- RAM: 4 GB
- HDD: 200 GB of available space in the one running the Xen Hypervisor

The computers were connected through a corporate Gigabit Ethernet LAN network. The measured Round-Trip delay Time (RTT) is 1.122 ms on average, based on 20 PING messages sent between the machines using the *ping* command-line tool.

```
tropic@tropic:~$ ping 172.24.76.118
PING 172.24.76.118 (172.24.76.118) 56(84) bytes of data.
64 bytes from 172.24.76.118: icmp_req=1 ttl=128 time=1.23 ms
64 bytes from 172.24.76.118: icmp_req=2 ttl=128 time=1.18 ms
64 bytes from 172.24.76.118: icmp_req=3 ttl=128 time=1.24 ms
64 bytes from 172.24.76.118: icmp_req=4 ttl=128 time=1.10 ms
64 bytes from 172.24.76.118: icmp_req=5 ttl=128 time=1.12 ms
64 bytes from 172.24.76.118: icmp_req=6 ttl=128 time=1.16 ms
64 bytes from 172.24.76.118: icmp_req=7 ttl=128 time=1.01 ms
64 bytes from 172.24.76.118: icmp_req=8 ttl=128 time=1.15 ms
64 bytes from 172.24.76.118: icmp_req=9 ttl=128 time=1.01 ms
64 bytes from 172.24.76.118: icmp_req=10 ttl=128 time=1.16 ms
64 bytes from 172.24.76.118: icmp_req=11 ttl=128 time=1.04 ms
64 bytes from 172.24.76.118: icmp_req=12 ttl=128 time=1.17 ms
64 bytes from 172.24.76.118: icmp_req=13 ttl=128 time=1.14 ms
64 bytes from 172.24.76.118: icmp_req=14 ttl=128 time=1.23 ms
64 bytes from 172.24.76.118: icmp_req=15 ttl=128 time=1.09 ms
64 bytes from 172.24.76.118: icmp_req=16 ttl=128 time=1.17 ms
64 bytes from 172.24.76.118: icmp_req=17 ttl=128 time=1.18 ms
64 bytes from 172.24.76.118: icmp_req=18 ttl=128 time=1.02 ms
64 bytes from 172.24.76.118: icmp_req=19 ttl=128 time=1.20 ms
64 bytes from 172.24.76.118: icmp_req=20 ttl=128 time=0.761 ms
^C
--- 172.24.76.118 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19028ms
rtt min/avg/max/mdev = 0.761/1.122/1.241/0.109 ms
```

Figure 40. Ping test between SCM and SC

5.1. Stress tests results

5.1.1. Context Vector

As described in Chapter 4 of this document, the Context Vector was implemented using nested ConcurrentHashMaps that contain information about the SCs, the VMs and the users in the system.

The stress test for the Context Vector aims to analyze its performance regarding the insertion and retrieval operations. It was executed in the laptop which had the Xen Hypervisor installed because the memory consumption by default was lower than the one in the other laptop, therefore allowing more RAM space for the test.

The tests were carried out in a single table, inserting a row of 10 elements corresponding to the amount of information required for the SC table. Figure 41 shows the time delay and the memory consumption achieved for different number of rows inserted.

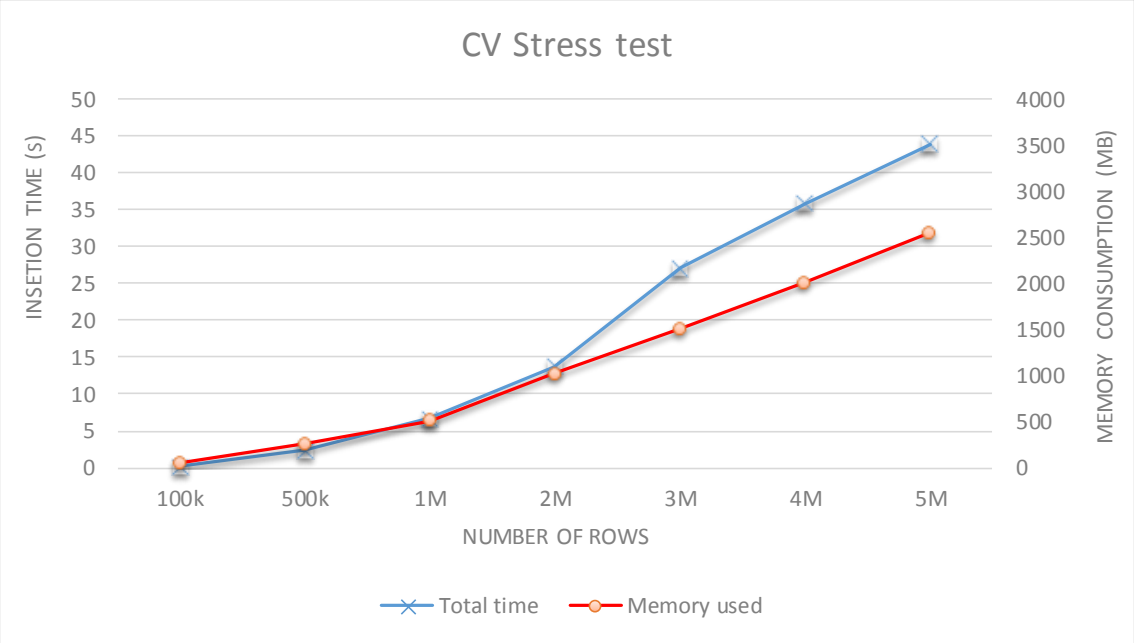

Figure 41. CV stress test row insertion result

The limit for this test was encountered at 5 million rows equivalent to 50 million registers. As it can be seen, the insertion time increases almost linearly with the amount of rows. The test was executing allowing the JVM (Java Virtual Machine) a maximum heap size of 4GB, 1GB more than the actual RAM available. Until 4 million rows, the memory of the SC was enough to carry out the test, however, when testing 5 million insertions, the swap memory was triggered. After that point, the time for the insertion raised to unpractical levels, surpassing the 10 minute barrier.

For 5 million rows, the time delay is slightly less than 45 seconds, which can be an acceptable value considering the amount of rows. However, for the amount of Small Cells that could be deployed in a real case scenario, e.g. around 250,000 SCs according to Cisco Small Cell Solutions [52], this behavior is far from being suitable and other CV storing solutions should be considered.

Besides analyzing the delay produced by data insertion in the CV, the data retrieving delay was also evaluated. A hash table by definition can achieve a search time of $O(1)$ in average thus making this time practically independent on the number of rows in the table. On the other hand, searching by value instead of by key requires traversing the table comparing each field with the one being searched until the value is found.

The search by value test was done using a hash map containing up to 6 million rows with 6 fields per row, as in the previous test. Figure 42 shows the obtained results.

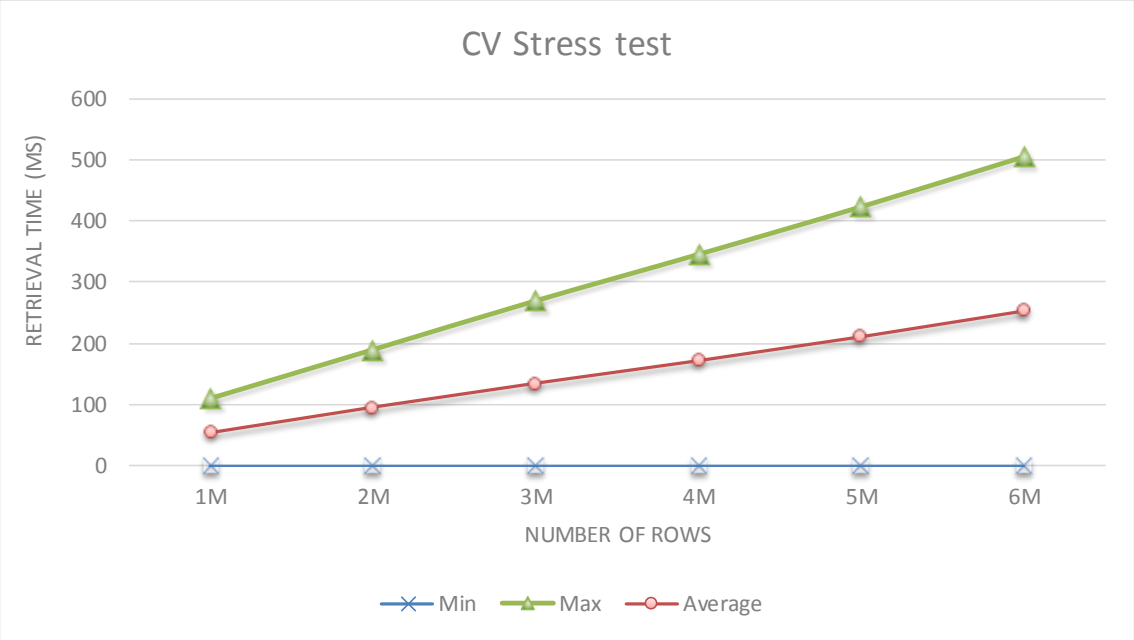

Figure 42. CV stress test row retrieval result

As expected, the minimum time for the search was practically 0, which corresponds to the case when the element being searched is the first one to be analyzed. For the opposite case, that is, when the value is placed in the last position of the table, the complete hash map has to be traversed, therefore obtaining the maximum search delay. As the number of rows increased, the search by value delay increased as well, in a linear way. For the worst possible case, i.e. the maximum delay for 6 million rows, the total delay was around 500 ms, a decent value. The average delay obtained ranged from 50 ms up to 250 ms.

Since the data is stored sequentially in the CV, the probability density function (PDF) that models the probability of having a search delay between the minimum and the maximum values is uniform. That means that any delay value in the whole range has the same probability of being produced than any other.

5.1.2. SC connection

The moment a SC is switched on, it sends a connection request to the SCM, which updates the CV information and answers with the assigned SC identifier. At this moment, the SC is considered to be connected and active in the Small Cell Cloud.

In case the connection is broken during the connection message interchange, the SC will try to reconnect waiting a fixed time period before the new attempt to avoid network congestion and the saturation of the SCM.

This test is carried out to obtain the average delay it takes a SC to be connected to the SCM when a high number of them are switched on in a short time period. To do so, SCs were started sequentially and both the accumulated delay and the time to stablish the connection were measured. As described before, the connection through which the tests were done was a corporate LAN Gigabit Ethernet with an average of 1.122 milliseconds of RTT. Figure 43 below shows the results obtained in the test.

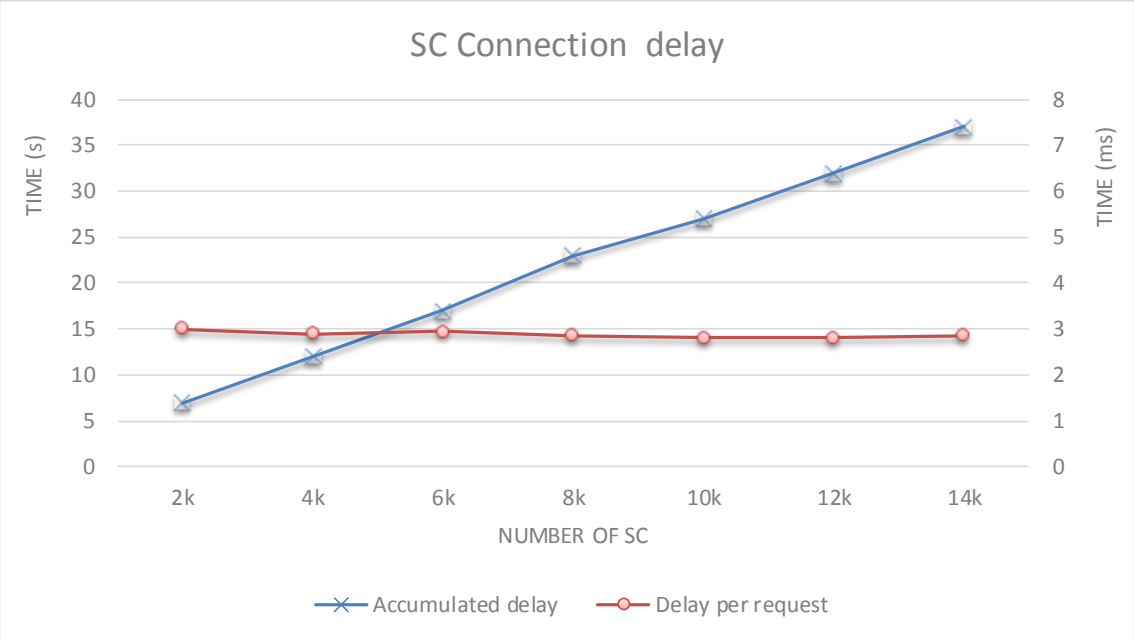

Figure 43. SC connection delay stress test result

As it can be seen, the accumulated delay for the connection increased linearly with the number of SCs used for the connection test, which ranged from 2k up to 14k. For 2k SCs the total delay obtained was 6 seconds, an acceptable value considering the amount of SCs. The average time for a SC to connect was kept at 2.9 milliseconds approximately for the whole range.

5.1.3. Primary VM deployment

One of the stress tests was designed to evaluate the performance of the SCM when primary VMs had to be deployed at SCs. The SCM uses the deployment algorithm defined in the Service Model to decide in which SC the VM should be deployed to keep a balanced load among SCs.

When a user connects to a SC, a primary VM associated to him or her is deployed in a SC that can be the one to which the connection was stablished or a different one. The test measured the time it takes to deploy the primary VM, after the correct SC is selected, depending on the time difference between user connection requests. For the test, 100 SCs were initially deployed and 500 user requests were used.

The results of the test are shown in Figure 44. For a delay bigger than 10 seconds between connections, the SCM was able to process each request without waiting for thread synchronization constraints, obtaining a deployment time of 10 seconds.

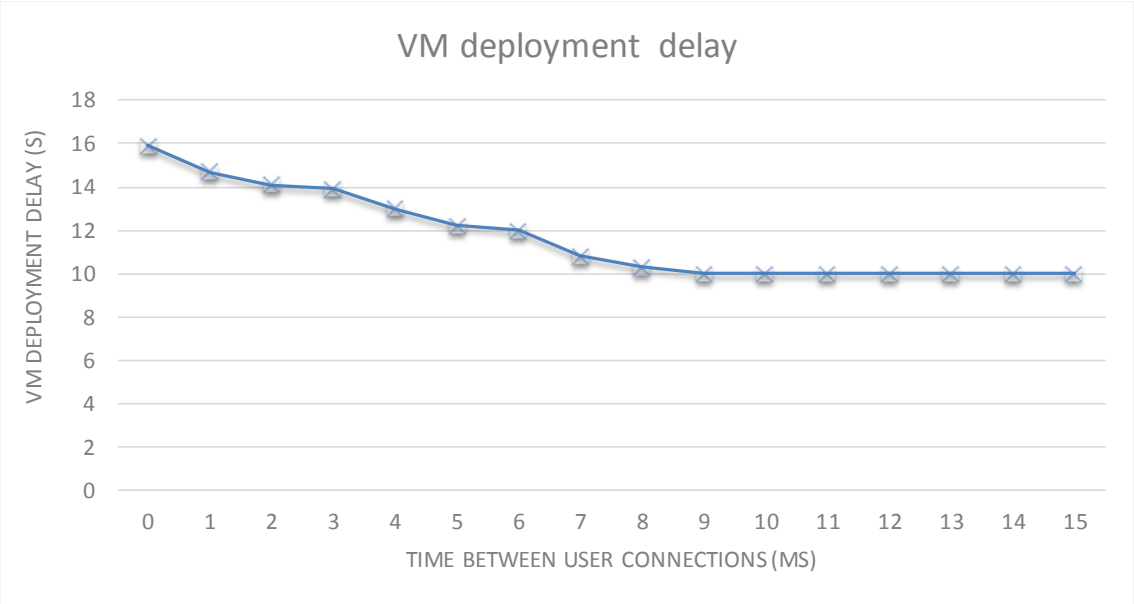

Figure 44. VM deployment delay stress test resutl

Despite the number of available threads in the thread pool was enough to process each request without creating new threads and therefore reducing that delay, the synchronization required to access the CV produced a delay that increases the total time for the primary VM deployment up to 16 seconds. In a real case scenario the probability of having 500 user connection requests at exactly the same time is almost negligible so it can be safe to assume that the behavior of the system is appropriate for the primary VM deployment.

5.1.4. Communication failure

One of the tests was dedicated to check the answer of both SCM and SC components when a failure in the communication occurs. This type of failure may be produced by the network (e.g.

broken link, traffic congestion) or by the system itself (e.g. software crashing, component saturation).

The test was carried out starting both components using the threshold-based proactive monitoring scheme and keeping them in a steady state. Then the network wire was disconnected, producing an error in the communication. As soon as one of the components try to reach the other end, an exception in the socket is raised and cached appropriately in the code, so that both the CV for the SCM and the LV for the SCs can be updated setting the other end status parameter as *DISCONNECTED*.

Whenever this situation happens, each component will try to recover the connection. The waiting time until a new request is done can be configured before the start of the SCM or the SC.

5.2. Proactive monitoring simulation analysis

In this section the proactive monitoring approach is simulated under specific system conditions in order to compare the performance of the periodic and the threshold-based methods in terms of the achieved accuracy and the amount of generated network traffic.

The reactive monitoring scheme has been left out of the analysis due to it being an on-demand monitoring and therefore providing always the best accuracy (no error in the measure) at expenses of a much higher network overload. Proactive monitoring is considered as an optimized mechanism over the reactive one, being able to provide a good compromise between accuracy and network load and making it more suitable to be implemented in real-case scenarios.

For the simulation, the chosen metric to be monitored is the SC CPU due to its highly dynamic behavior. Memory consumption can be expected to change at a slower rate and in more predictable steps, since the amount of reserved memory for new VMs is always the same.

Several CPU patterns have been extracted for actual CPU data under different conditions in order to test the performance of the monitoring mechanisms on each of them. The data have been generated using an adjustable CPU stress tool and the monitoring software included in the OS. A total of 400 sample metrics monitored every second have been extracted for all the patterns, an equivalent of almost 7 minutes of monitoring.

In order to obtain accurate results in the long term, the extracted CPU data has been extended until 100,000 samples, using repetition and adding a normal random value around the each point with standard deviation of 1 to induce controlled variability. The following figures are sample sections for each pattern.

- **Low variability pattern**: this pattern shows a CPU that is stable around the same value all the time. This kind of CPU can be expected when the SC is active but no users are connected to it so it remains during a certain amount of time with the same CPU usage.

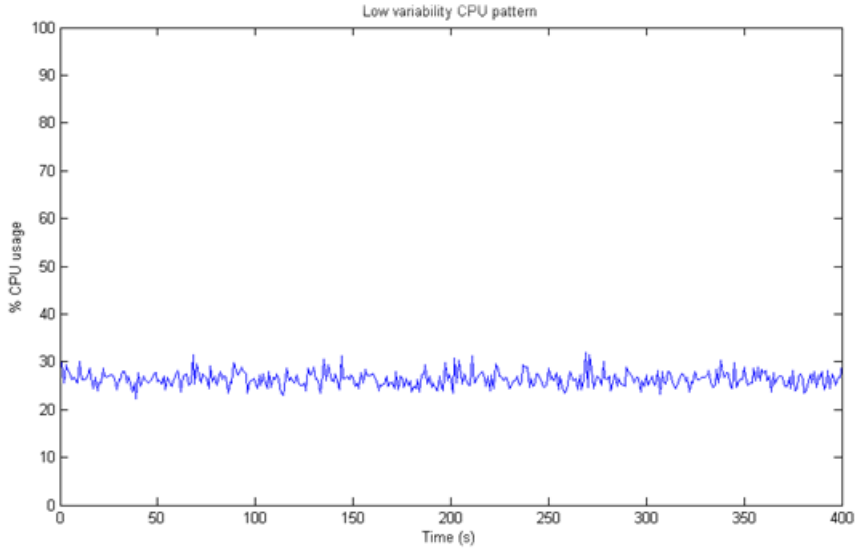

Figure 45. Low variability CPU pattern

- **Periodic with low variability pattern**: this pattern corresponds to a CPU usage that varies between low and medium workload periodically resulting in a metric with moderated variability.

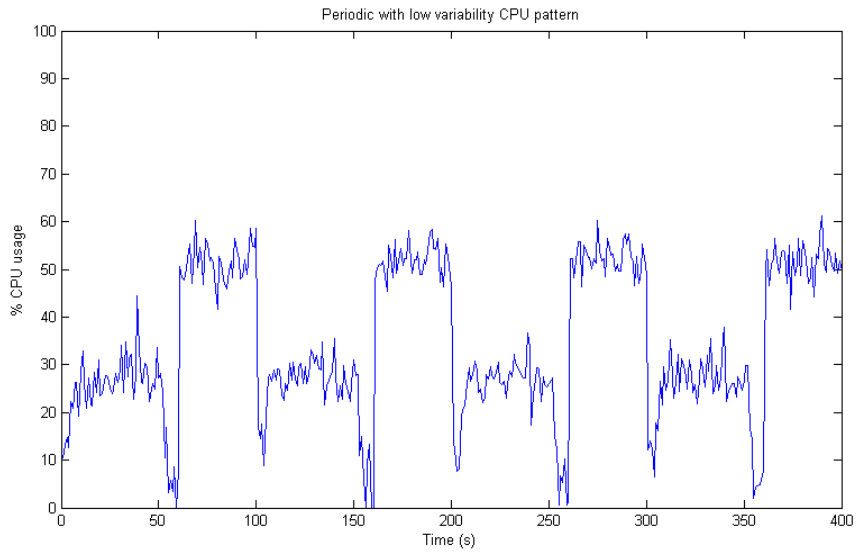

Figure 46. Periodic with low variability CPU pattern

- **Periodic with high variability pattern**: this pattern represents a CPU in which highly demanding tasks are periodically executed in short periods of time. The metric variability of this pattern is high.

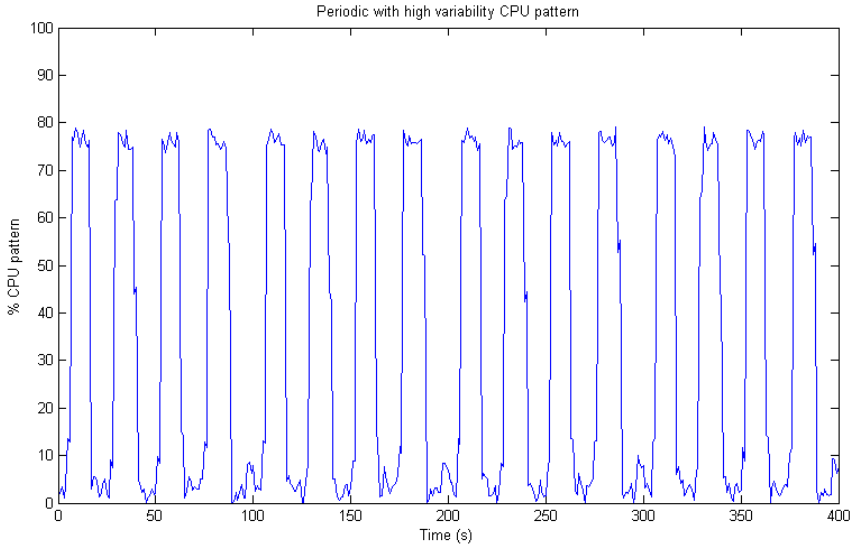

Figure 47. Periodic with high variability CPU pattern

- **Pseudo-random pattern**: this pattern is the result of a CPU working under random workload, when there can be peaks of computation and moments of low processing, maintaining a medium-low average workload. It contains the highest metric variability.

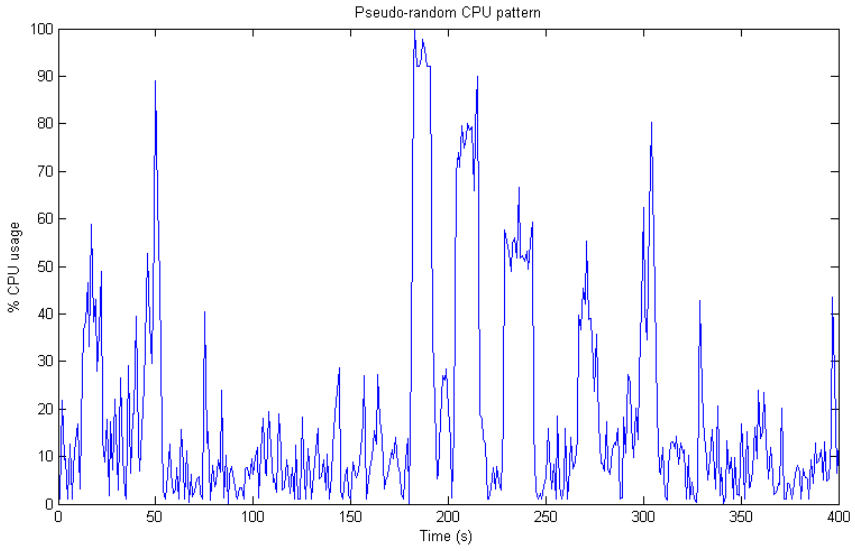

Figure 48. Pseudo-random CPU pattern

5.2.1. Accuracy comparison

The monitoring mechanisms described in Chapter 3 have been tested using the previous CPU models. The analysis of this section is focused on the accuracy obtained for each case and monitoring scheme. Individual results for each of the four CPU patterns have been obtained,

computing the average value to get a better approximation of the overall performance of the monitoring mechanism.

5.2.1.1. Periodic monitoring

First, the periodic proative monitoring has been tested. Figure 49 shows the obtained results for the CPU patterns. The horizontal axis represents the period, in number of samples, between each metric sent to the SCM.

As it is expected, increasing the period between metrics sent to the SCM leads to a lower accuracy in average. The best performing pattern is the stable CPU, the one with the slowest variance, whereas the periodic pattern shows the worst performance.

Due to the periodicity that some patterns have it is possible to achieve higher accuracy approximating the sampling period with the period of the pattern itselft. This is why several increment peaks on the accuracy can be observed in the graph for specific periods, for example at 10 sample period for the high-low CPU pattern.

The smallest accuracy value obtained with this monitoring scheme is around 0.75 for the highest periods.

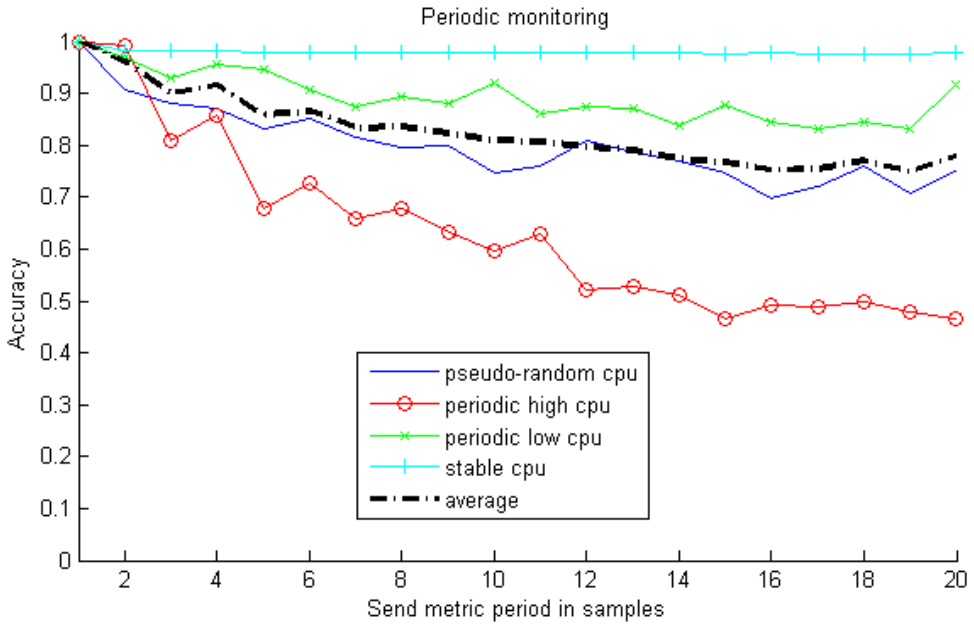

Figure 49. Periodic monitoring accuracy simulation results

5.2.1.2. Threshold-based monitoring: mean metric value

Next the analysis of the threshold-based proactive monitoring is presented. For this monitoring scheme two different methods of updating the central value for the thresholds can be considered. The first one is the mean metric value, for which the amount of metrics to consider for the mean value can be adjusted to obtain different results.

The following graphs show the simulation results for the mean metric value threshold update considering a set of 2, 5 and 10 previous metrics to compute the mean. The horizontal axis represents the threshold distance to be added and subtracted to the middle value, measured as a fraction of the total range value of the metric being monitored, e.g. in the case of monitoring the % CPU usage and assuming a middle value of 50% CPU, a 0.3 threshold distance would mean that the upper threshold is at 80% CPU, 30% higher than the middle value, and the lower threshold is at 20% CPU, a 30% lower. Using a threshold distance higher than the 0.5 would decrease the accuracy due to the distance from the upper and lower threshold being too large.

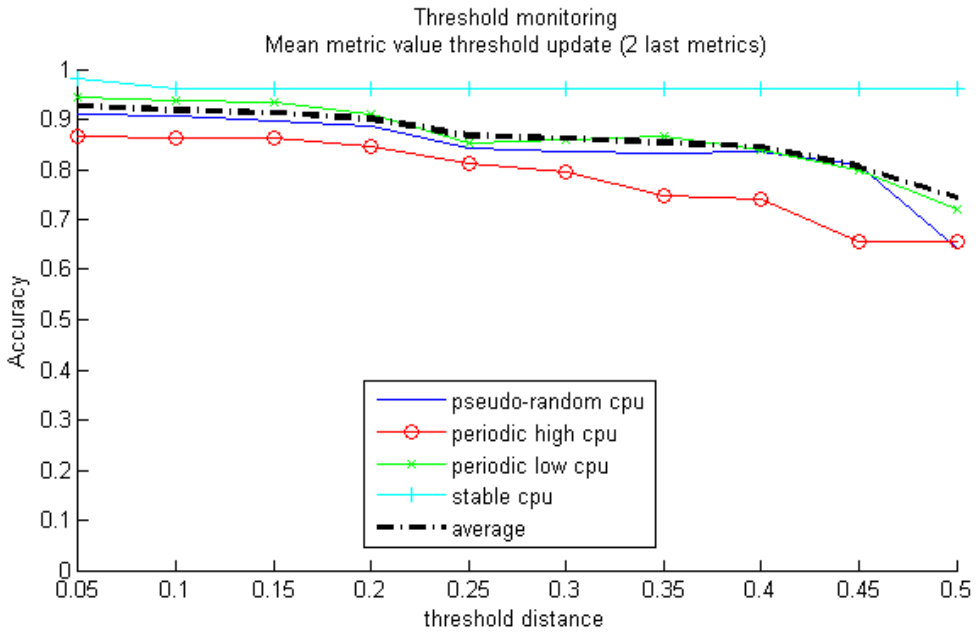

Figure 50. Threshold monitoring, mean metric value (2 last metrics) accuracy simulation results

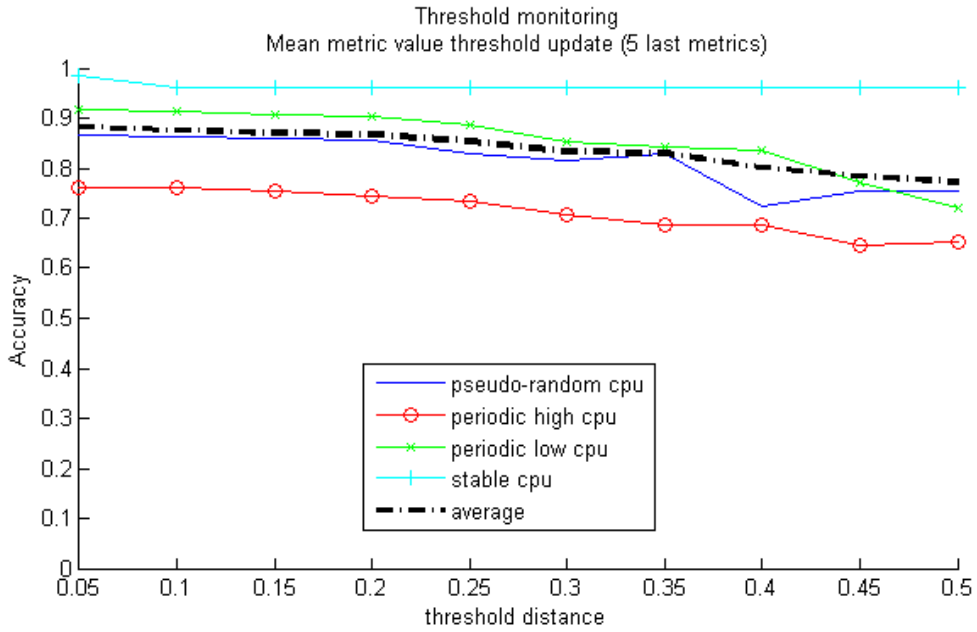

Figure 51. Threshold monitoring, mean metric value (5 last metrics) accuracy simulation results

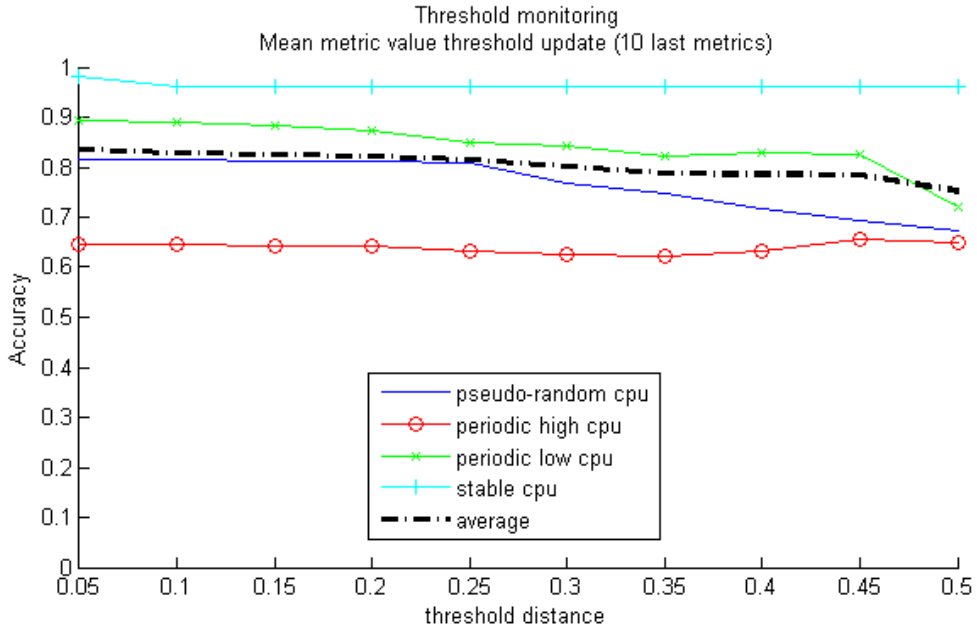

Figure 52. Threshold monitoring, mean metric value (10 last metrics) accuracy simulation results

The graphs show that taking a smaller number of previous metrics to compute the middle value for the thresholds yield a higher average accuracy. The more the number of metrics considered for the middle value, the slower it changes which result in worse performance for CPU patterns with more variability such as the periodic high and the pseudo-random ones. As before, the stable CPU pattern achieves the highest accuracy for the whole range of tested threshold distances due precisely to its low variability, always keeping the metrics under the thresholds.

It can be seen as well that using the two last metrics for the central value of the monitoring, the accuracy decreases to reach a similar value than the other simulations. This implies that the number of metric values to compute the mean is not so relevant when the threshold distance is very large.

It is also worth noting that selecting just one value for computing the mean, i.e. setting the last metric value as the mean itself, would not show the same results than in the case of using the periodic monitoring approach with one period sample. This is so due to the changing middle value in the case of the threshold-based monitoring, that would result in metrics not being notified to the SCM. The periodic monitoring with one period sample would send each obtained metric value to the SCM regardless of their value, and therefore obtaining a 100% accurate result.

5.2.1.3. Threshold-based monitoring: last sent metric value

Finally, the next figure shows the accuracy results for the threshold-based monitoring case in which the middle value is chosen to be the last metric notified to the SCM. For this graph the same limit of 0.5 used for the previous simulation is also applied for the same reasons.

The obtained results reveal the highest average accuracy from among the analyzed simulations, achieving more than 90% accuracy until a threshold distance of 30%. Contrary to the mean metric value threshold-based monitoring, this approach proves to get better results for CPU patterns with more variability. On the other hand, from threshold distances higher than 30%, the accuracy for the stable CPU pattern falls to a minimum value. Since the first monitored metric is always sent to the SCM, it can occur that the threshold distance is so large that no more metrics are sent apart from that one and therefore the difference between the monitored metrics in the SC are subsequently different from the one the SCM has stored, lowering the accuracy.

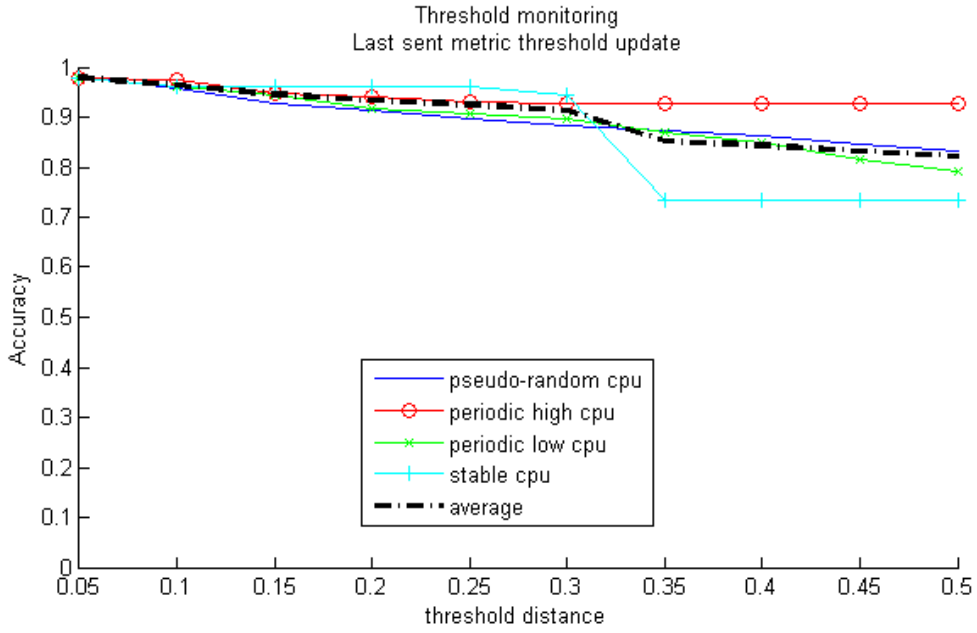

Figure 53. Threshold monitoring, last sent metric accuracy simulation results

From the previous analysis it can be extracted that the best monitoring approach in terms of accuracy of the metrics is the threshold-based scheme using the last sent metric as the middle value around which set the upper and lower thresholds.

5.2.2. Generated traffic comparison

The previous analysis has been done to compare the accuracy that the different proactive monitoring schemes can achieve. However, it is also important to consider the amount of network traffic each one generates. The final purpose is to find a trade-off between the accuracy and the amount of metrics that have to be sent from the SC to the SCM.

5.2.2.1. Periodic monitoring

Due to the characteristics of the periodic monitoring, a dedicated analysis on the amount of metrics sent has not been performed since they are inversely related to the sample period at which the metrics are sent. Reducing the period at which the samples are sent to the SCM increases the monitoring traffic by the same amount. Although the accuracy increases as well for small periods, the amount of traffic generated could result excessive and usually a compromise between the two would be preferred.

With the periodic monitoring the SCM is able to adjust with precision the amount of monitoring traffic it receives by just telling the SCs to increase the period at which they send the metrics. For low congested situations, the accuracy in the metric values of the SCM can be increased demanding more frequent monitoring messages from the SCs. This mechanism allows the SCM to keep a constant monitoring traffic flowing through the Small Cell Cloud.

5.2.2.2. Threshold-based monitoring: mean metric value

The same procedure has been followed for this monitoring scheme, using a different number of previous metrics to compute the middle value around which the thresholds are established. In order to compare the number of metrics that are sent to the SCM, the scale has been defined to range from 0 to 1 representing the fraction of monitoring metrics sent with respect to the total metrics obtained in the SC, 0 meaning that no metrics are notified to the SCM and 1 that the total of metrics taken in the SC have been sent to the SCM. Figure 54, Figure 55 and Figure 56 show the traffic generated for the cases where 2, 5 and 10 last metrics are considered to compute the mean respectively.

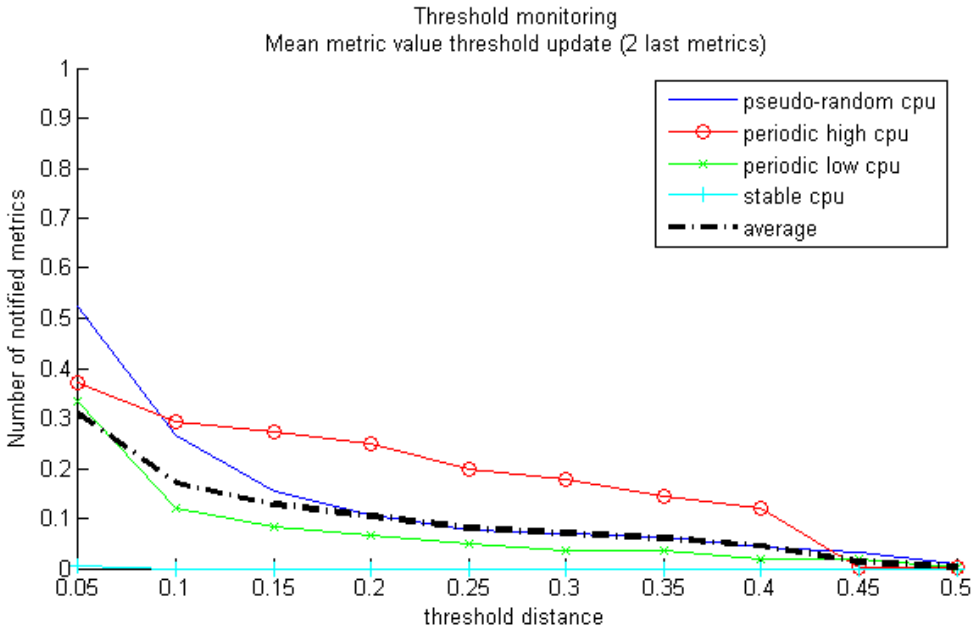

Figure 54. Threshold monitoring, mean metric value (2 last metrics) number of metrics sent simulation results

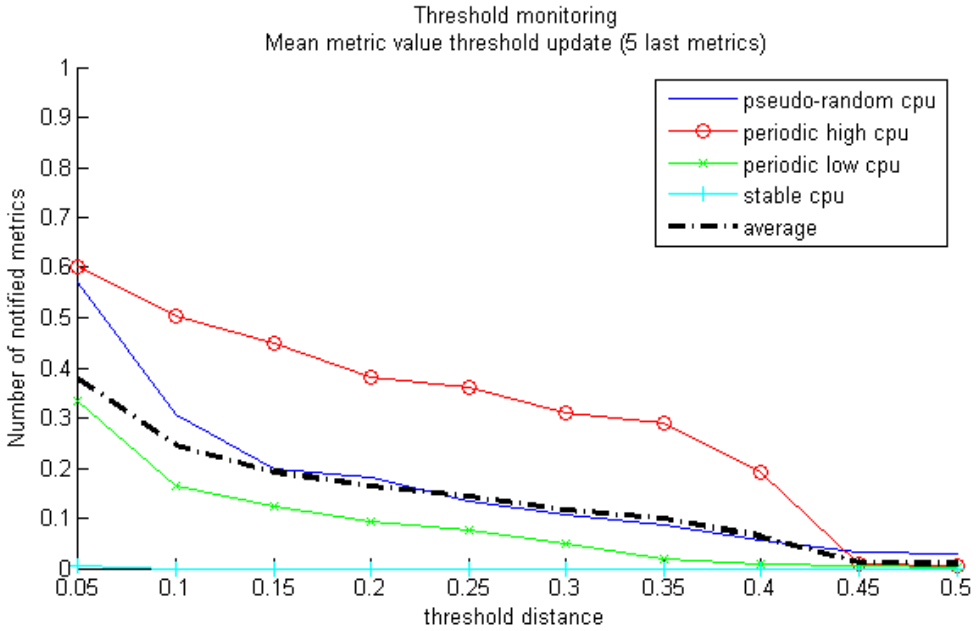

Figure 55. Threshold monitoring, mean metric value (5 last metrics) number of metrics sent simulation results

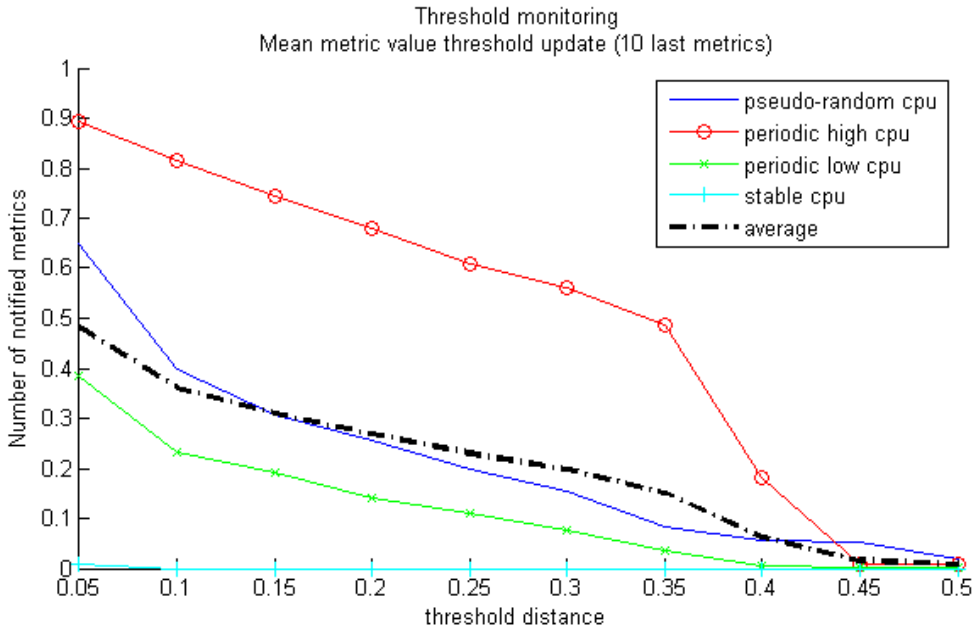

Figure 56. Threshold monitoring, mean metric value (10 last metrics) number of metrics sent simulation results

As it can be seen, the lowest number of monitoring metrics in average is sent using just the 2 last metric values, which from previous section is the one that achieves the highest accuracy as well. Similarly, the maximum number of metrics sent is for the case of the periodic with high variability CPU pattern using 10 last metrics for the mean, notifying the SCM with 90% of the total metrics obtained in the SC. Due to the slow change of the mean value, metrics with high variability fall frequently out the threshold and therefore are notified to the SCM.

On the other hand, for the stable CPU pattern no messages are sent for threshold values higher than 0.1. In this case, and as it has been suggested when showing the accuracy results, the metric values for the CPU pattern with lowest variability fall inside the thresholds so no further metric notification to the SCM are required, maintaining the accuracy to nearly 1.

Reducing the threshold distance achieves a better accuracy at expenses of increasing the number of sent metrics. The smallest average number of metrics is a 30% of the total measured using the 2 last values for the mean and a threshold distance of 5% CPU usage. That value is increased up to half the number of monitored metrics for the same threshold distance but using 10 last metrics. Keeping the mean to be computed using just two previous values, an average 90% accuracy can be obtained notifying only the 10% of the monitored metric, which improves the scalability of this monitoring approach.

The results obtained with this monitoring approach are better than the ones that obtained with the periodic monitoring. To notify only the 10% of the metrics using the periodic approach, the sending period would have to be sent at 1 every 10 samples, which results in an average accuracy a little higher than the 80%, 10% less than the value achieved by the threshold-based monitoring using 2 values for the mean. In the case of a stable CPU pattern with very low variability, the threshold-based monitoring does not require to send messages to the SCM

without further configuration. Whereas similar results can be achieved using the periodic monitoring approach, the period would have to be constantly extended because of the low variability, but that would require exchange of adjustment messages from the SCM and sporadic metric notifications from the SC, resulting in a lower efficient monitoring scheme.

5.2.2.3. *Threshold-based monitoring: last sent metric value*

Finally, the results for the threshold-based monitoring using the last sent metric as the middle value for the thresholds are presented in Figure 57. In this case, the maximum number of metrics sent to the SCM is around the 60% of the total monitored metrics for the pseudo-random CPU pattern. Similar to the previous threshold-based monitoring configuration, no metric notification messages are required for a threshold distance above the 10%.

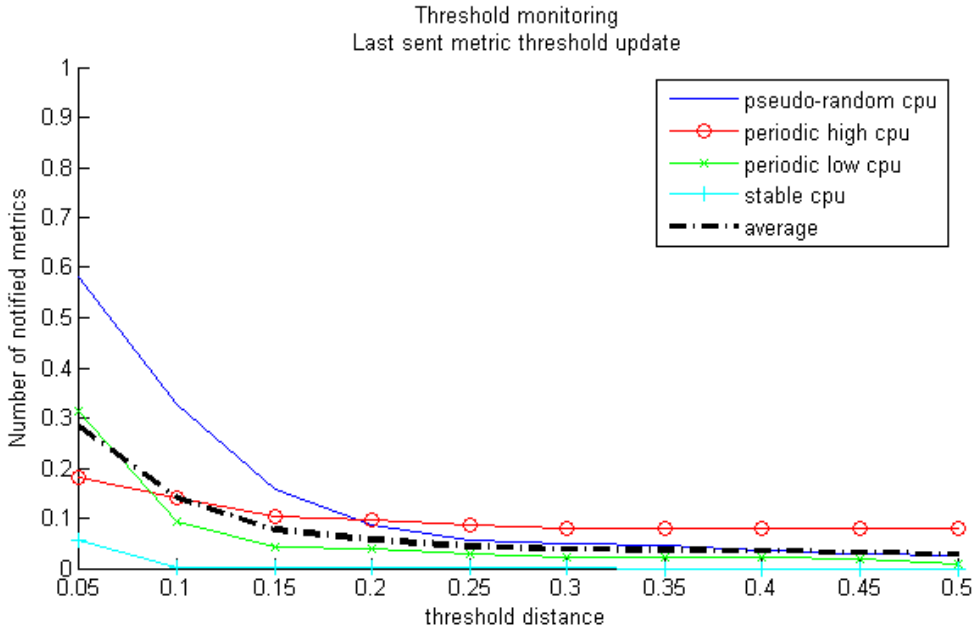

Figure 57. Threshold monitoring, last sent metric number of metrics sent simulation results

The average number of metrics sent to the SCM is slightly lower than for the best case using the mean metric value for the middle point between the thresholds. However, the results show that for a periodic CPU pattern with high variability, the required number of metric notifications is significantly lower; around half of the notification messages are needed in this case.

Despite achieving similar results when compared with the other threshold-based monitoring approach, looking at the achieved accuracy values the last sent metric method gets better results. Notifying 10% of the total monitored metrics can be reached with 0.15 threshold distance, 0.05 lower, and more than 95% accuracy is obtained, considerably better than in the previous case.

Due to the stability in high accuracy values shown for this monitoring approach, still 90% of accuracy can be obtained notifying just the 5% of the total metrics for a threshold distance of 0.3.

The simulation analysis for the different proactive monitoring schemes show that the best results are obtained with the threshold-based one using the last sent metric for the threshold middle point. High values of accuracy, around 90%, can be obtained notifying a small amount of the total monitoring metrics in the SC, only 5%, which allows an efficient way of monitoring the SC parameters without saturating both the network and the SCM with high loads of monitoring traffic.

Section 7.1 presents the final conclusions that can be derived from the previous results.

CHAPTER 6
PLANNING AND COSTS

6.1. Project planning

The work employed in this project extends from 1st September, 2014 until 16th February, 2015. With an average work per day of 2-3 hours without taking into account holidays and occasional free days, the project duration has an estimated amount of 300 hours.

Throughout the whole project a logical planning has been followed: The first steps carried out were the problem analysis and the study of involved technologies. This was followed by the State of the Art documentation and, in parallel, a first approach of the requirements definition.

Once the components of the system were defined, the implementation work began. This phase was divided in two parts: a first basic implementation of both components which contained the main functionality and a reduced set of the defined commands; and a second implementation where the components were extended to achieve the defined capabilities adding an optimized version of each module and support for the whole set of commands. Due to its simplicity, the reactive monitoring was included using the basic implementation of the component, whereas the proactive monitoring schemes were introduced with the optimized modules.

The unit tests were being done in parallel with the component's code verifying that each module and function was working properly. Stress tests were done once the system was completely implemented to obtain its actual performance.

During the implementation and tests phases, some missing design features were realized (e.g. new fields in the CV or hypervisor configuration improvements), leading to minor requirement adjustments.

Documentation work has been done progressively since the beginning of the project, both literature and code, being assembled and organized during the last part of the project.

The corresponding Gantt diagram with the approximate planned start and duration of each task of the project is shown in Figure 58 next page.

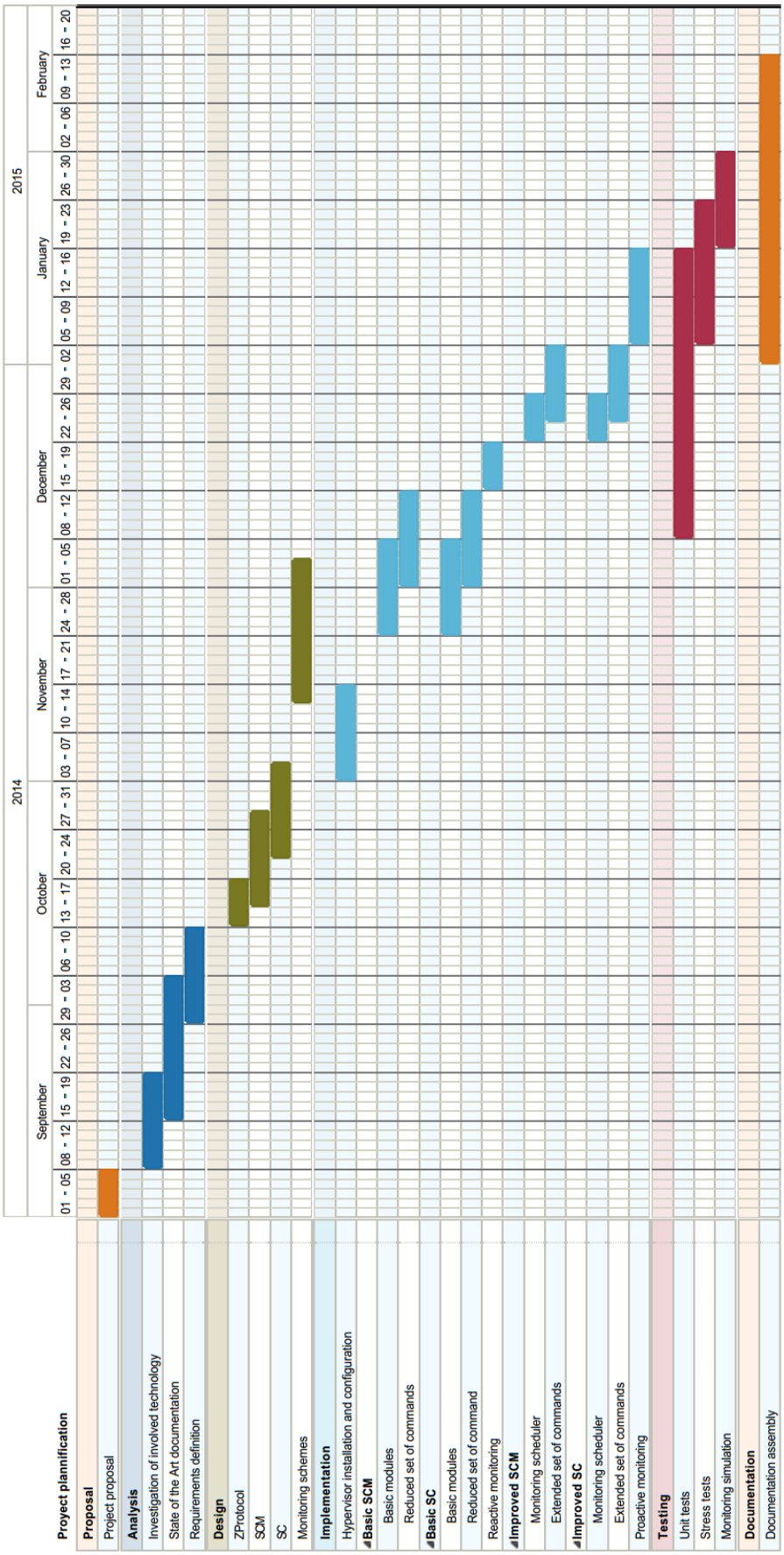

Figure 58. Project planning Gantt diagram

6.2. Project costs

This section presents the estimated budget required for the development of the present project divided into categories.

6.2.1. Personnel cost

The personnel costs associated to this project have been calculating taking into account the previous planning.

According to the publication of the *Colegio Oficial de Ingenieros de Telecomunicaciones* (COIT, Official School of Telecommunication Engineers) in 2013 where an analysis of the telecommunications sector was presented, the average salary of a telecommunications engineer is around 42,000€/year [53]. However, this quantity is reduced to 25,700€/year for engineers with less than 5 years of experience. Because of this research project does not necessarily require more than 5 years of experience on the field, the cost estimation has been done using the lower salary value.

As described in the previous section, the duration of this project has been determined to be 300 hours. This is equivalent to $300/8 = 37.5$ days or 7.5 weeks in a full time job. Doing the proper calculations, the total personnel cost results in:

$$\frac{25{,}700\ €}{year} * \frac{7.5\ weeks}{52\ \frac{weeks}{year}} = \mathbf{3{,}706}\ €$$

6.2.2. Hardware cost

The hardware used for this project has consisted in two identical enterprise laptops, one to deploy the SCM and the other to run the hypervisor and the SCs, and a personal computer to write the documentation and run the simulations. Both laptops contained an Intel i5-3320M CPU@2.60GHz, 4 GB of RAM and a 300GB HDD, whereas the personal pc had an Intel i5-4440 CPU@3.10GHz, 8 GB of RAM and 1TB of HDD.

The cost computation for these elements has been taking into account the depreciation period, which corresponds to the expected useful life of the device. For personal computing resources a 3 year useful life is considered. Therefore, the costs can be calculated in the following way:

$$Imputable\ cost = Cost\ without\ taxes / depreciation\ period * utilization\ time$$

The hardware cost results in:

Table 5. Hardware cost detail

Asset	Cost w/o taxes	Depreciation period	Utilization time	Quantity	Imputable cost
Laptop	600€	3 years	16 weeks	2	123.08 €
PC	700€	3 years	20 weeks	1	89.74 €
TOTAL					**212.82 €**

6.2.3. Software cost

Detailed costs regarding the software employed in this project are described below.

The Windows 7 Professional license came preinstalled in the corporate laptops. However a student license can be obtained for free, using the Microsoft Developer Network Academic Alliance (MSDNAA) program. For Microsoft Visio 2013 software, used to create most diagrams in the project, a license can also be obtained for free due to the MSDNAA program.

Ubuntu OS, the Xen Hypervisor and the Libvirt libraries are completely free and Open Source software.

The Matlab Student license to perform the simulations can be obtained for 35 €. Finally, the Office suite used to document the project and represent various data, can be purchased for 119 € for the Home version, which contained enough features for developing the project.

Table 6. Software cost detail

Software	Cost
Windows 7 Professional License	Obtained free
Ubuntu 12.04 LTS	Free
Xen Hypervisor	Free
Libvirt	Free
Matlab 2014a Student License	35 €
Microsoft Office Home 2013	119 €
Microsoft Visio 2013	Obtained free
TOTAL	**119 €**

6.2.4. Other costs

This section reflects the indirect costs on electricity and office supplies, including a cost margin on the previously described costs due to planning changes. A 15% of the budget is destined to the electricity cost whereas 50 € are dedicated to the office supplies. Because of the scope of the project the cost margin has been stablished at a 15% of the direct costs.

This project represents the first step towards a more complete Small Cell Cloud system so a 20% benefit over the costs is considered.

6.2.5. Total costs

The following table provides a summary of the estimated total project cost. An important remark is that a 21% of taxes have been applied to the total computed cost, following to the current legislation.

Table 7. Total project cost detail

Description	Cost
Direct cost	4087.82 €
Personnel cost	3,706 €
Hardware cost	212.82 €
Software cost	119 €
Office supplies	50 €
Indirect cost	613.17 €
Electricity	613.17 €
Partial total	4700.99 €
Risk margin	705.15 €
Benefits	940.20 €
Total (no taxes)	6346.34 €
Taxes (21%)	1332.73 €
TOTAL	**6033.72 €**

The total cost of the project results in SIX THOUSAND, THIRTY-THREE AND SEVENTY-TWO HUNDREDTHS EURO.

CHAPTER 7
CONCLUSIONS AND FUTURE WORK

7.1. Conclusions

Taking the basic component definition envisioned in FP7 TROPIC project as a starting point, the work done in this project has contributed to extend some design concepts and provide an actual software implementation of the system. Moreover, performance results based on the implementation have been also obtained.

Several goals have been achieved:

- This project has been able to improve the definition of both the SCM and SC modules by considering more specific requirements related to the VM management and monitoring aspects.
- New ZProtocol messages have also been included to deal with the monitoring aspect as well as modifying already defined messages within TROPIC to better adapt them to the system behavior.
- A real proof of concept implementation of the Small Cell Cloud has been developed as well, based on the updated design and using a simplified own version of Service Model algorithms, which has allowed to evaluate both its viability and performance thus giving a first step into the Small Cell Cloud paradigm.
- Furthermore, different possible mechanisms for SC monitoring have been provided and thoroughly compared to each other, based on the particular definitions of metric accuracy and network load, in order to gather their benefits and drawbacks and obtain the best suitable scheme for a real world scenario.

The conclusions that can be extracted from the performance evaluation of the Small Cell Cloud implemented in this project are the following:

- The implemented SCM is able to manage a very large number of SCs in an equipment with medium specifications.

- In-memory databases implemented as a Java variable, such as the one used in this project for the Context Vector in the SCM, show a good response for a quite large number of entries, i.e. few millions of rows, though present significant limitations when the amount of stored entries exceeds the available memory of the equipment.

- The time needed to deploy a VM in a SC is in the order of the 10 seconds, even when a high number of SCs have to be attended simultaneously, which is an acceptable amount of time for users to wait.

- Proactive monitoring schemes lead to better results in both metric accuracy and network load than reactive monitoring. Although proactive monitoring in SCs still requires the SCM to send occasional configuration messages, the metric accuracy obtained is closer to the 100% for considerably lower interchanged messages than with reactive monitoring.

- Among proactive monitoring schemes, the threshold-based approach achieves similar or even better metric accuracy with less notification messages than the periodic approach. The best results using the threshold-based monitoring approach are obtained using the last sent metric as the parameter around which stablish the thresholds. With this method, an average of 95% metric accuracy can be achieved just notifying around the 5% of the total monitored metrics in the SC.

- The SCM is able to detect a network problem when communicating with a SC and updating that information in the CV so that it can be taken into account in the VM deployment and migration algorithms.

- Despite being a proof-of-concept implementation, the Small Cell Cloud developed in this project results in a viable approach to merge the Small Cell with the Cloud Computing and leverage the benefits of both technological paradigms.

Finally, an important remark to be pointed out is that the work developed in this project will be included in the contribution of Atos SE to the next TROPIC deliverables to the European Commission.

7.2. Future work

Because of the main objective of this project has been to implement and analyze a proof of concept version of a Small Cell Cloud, more research work remains to be done to improve and complete the overall system.

The following is a list of possible extensions to the previous work that have been left out of the scope of the project:

- Study the adaptation of the SCM and SC software to the current LTE eNodeB base stations in order to achieve a complete merge of the presented system with current broadband mobile networks.

- Analyze communication mechanisms for very wide deployment scenarios where several SCM can be running. Two approaches can be considered for the distributed CV information that arises when several SCM are deployed: distributing the SC information among the SCM using for example a hierarchical structure to reduce network traffic or having a central CV from which the SCMs concurrently obtain and store the information.

- Evaluate the system performance in a more realistic scenario with dedicated equipment and infrastructure, instead of using parallel execution with threads to simulate several SCs in a single computer.

- Develop security mechanisms to protect and guarantee the integrity of the data and measuring their impact on the overall system performance. Examples of security actions can be: adding a security layer to the Z Protocol implementation, including authentication and access control methods for the VMs and provide encryption mechanisms to protect the data inside a SC.

List of abbreviations and acronyms

3GPP	3rd Generation Partnership Project
API	Application Programming Interface
AWS	Amazon Web Services
BBU	Baseband processing Unit
BS	Base Station
C-RAN	Cloud Radio Access Network
CEO	Chief Executive Officer
CMP	Cloud Management Platform
CPU	Central Processing Unit
CV	Context Vector
DSL	Digital Subscriber Line
EU	European Union
FP7	Seventh Framework Programme
Gpbs	Gigabits per second
GRUB	GRand Unified Bootloader
GUI	Graphical User Interface
Het-Net	Heterogeneous Networks
HVM	Hardware-assisted or Full Virtualization
I/O	Input/Output
IaaS	Infrastructure as a Service
ICI	Inter-cell Interference
IoT	Internet of Things
IP	Internet Protocol
IPSEC	Internet Protocol security
JDBC	Java Database Connectivity
JVM	Java Virtual Machine
LAN	Local Area Network
LTE	Long Term Evolution
LTE-A	Long Term Evolution - Advance
MB/GB	Megabyte/Gigabyte

MiB/GiB	Mebibyte/Gibibyte
MIMO	Multiple Input Multiple Output
NIST	National Institute of Standards and Technology
OS	Operating System
PaaS	Platform as a Service
PC	Personal Computer
PV	Paravirtualization
QoS	Quality of Service
RAM	Random Access Memory
RAN	Radio Access Network
RMI	Remote Method Invocation
RMSE	Root-mean-square Error
RPC	Remote Procedure Call
RRD	Remote Radio Head
RTT	Round Trip Time
SaaS	Software as a Service
SC	Small Cell
SCM	Small Cell Manager
SQL	Structured Query Language
TCP	Transmission Control Protocol
TLS/SSL	Transport Layer Security/Secure Socket Layer
UDP	User Datagram Protocol
UE	User Equipment
VDI	Virtual Desktop Infrastructure
VIM	Virtual Infrastructure Manager
VM	Virtual Machine
VMM	Virtual Machine Manager
VPN	Virtual Private Network

REFERENCES

1. *Research & Innovation*. European Comission. [cited September 2014]; Available from: http://ec.europa.eu/research/index.cfm.
2. *TROPIC: Distributed Computing, Storage and Radio Resource Allocation over Cooperative Femtocells*. [cited September 2014]; Available from: http://www.ict-tropic.eu.
3. Luis M. Vaquero, L.R.-M., Juan Caceres, Maik Linder, *A Break in the Clouds: Towards a Cloud Definition.* ACM SIGCOMM Computer Communication Review, 2009.
4. Peter Mell, T.G. *The NIST Definition of Cloud Computing*. 2011.
5. *Amazon Web Services (AWS) - Cloud Computing Services*. Amazon Web Services, Inc. Available from: http://aws.amazon.com.
6. *The Cloud for Modern Business*. Microsoft. Azure: Microsoft's Cloud Platform; Available from: http://azure.microsoft.com.
7. *IBM Cloud / IaaS*. IBM. Cloud Computing: Infrastructure as a Service (IaaS); Available from: http://www.ibm.com/cloud-computing/us/en/iaas.html.
8. *Google App Engine*. Google Developers; Available from: https://developers.google.com/appengine.
9. *Heroku*. Available from: https://www.heroku.com.
10. *Develop, Host, and Scale Your Apps in the Cloud*. Red Hat Incred. OpenShift by Red Hat; Available from: https://www.openshift.com.
11. *Akamai Web Page*. Akamai. Cloud Services, Enterprise, Mobile, Security Solutions; Available from: http://www.akamai.com/index.html.
12. *Cloud9*. Cloud9. Online IDE; Available from: https://c9.io.
13. *Business ByDesign Software Solutions*. SAP. Available from: http://www.sap.com/pc/tech/cloud/software/business-management-bydesign/overview/index.html.
14. Qi Zhang, L.C., Raouf Boutaba, *Cloud computing: state-of-the-art and research challenges.* Internet Serv App, 2010.
15. Borja Sotomayor, R.S.M., Ignacio M. Llorente, Ian Foster, *Virtual Infrastructure Management in Private and Hybrid Clouds.* IEEE Internet Computing.
16. *OpenNebula | Flexible Enterprise Cloud Made Simple*. [cited September 2014]; Available from: http://opennebula.org/.
17. Jiang, Q. *CY13-Q4 Community Analysis — OpenStack vs OpenNebula vs Eucalyptus vs CloudStack*. [cited September 2014]; Available from: http://www.qyjohn.net.
18. *OpenStack*. 2014 [cited September 2014]; Available from: https://www.openstack.org.
19. Apache. *Concepts and Terminology*. CloudStack 4.3.0 Documentation; Available from: http://docs.cloudstack.apache.org/en/latest/concepts.html.
20. Brockmeier, J. *It's Not Highlander: There Can Be More Than One Open Source Cloud*. 2012; Available from: http://readwrite.com/2012/04/05/its-not-highlander-there-can-b#awesm=~oIiz1m2QWQ0lHs.
21. *Hypervisor - Xen*. [cited September 2014]; Available from: http://www.xenproject.org/developers/teams/hypervisor.html.
22. *KVM Hypervisor*. KVM Official Site [cited September 2014]; Available from: http://www.linux-kvm.org.

23. Dubuque, C. *KVM is Linux. Xen is Not.* 2012; Available from: http://www.chucknology.com/?p=15.
24. VMware, *VMware vSphere Basics*. http://www.vmware.com/support/pubs.
25. Microsoft. *Hyper-V Architecture*. Developer Network; Available from: http://msdn.microsoft.com/en-us/library/cc768520%28v=bts.10%29.aspx.
26. *3GPP TR 36.839 V11.0.0 (2012-09)*. 3rd Generation Partnership Project.
27. *Small Cell Forum*. Available from: http://www.smallcellforum.org.
28. *Small cell market status*. Small Cell Forum. February 2013 2013.
29. Niroshinie Fernando, S.W.L., Wenny Rahayu, *Mobile cloud computing: A survey.* Future Generation Computer Systems, 2013.
30. Mahadev Satyanarayanan, P.B., Ramón Cáceres, Nigel Davies, *The Case for VM-Based Cloudlets in Mobile Computing.* Pervasive Computing, 2009.
31. Marinelli, E.E., *Hyrax: Cloud Computing on Mobile Devices using MapReduce*. 2009, Carnegie Mellon University.
32. *Fog Computing*. Cisco. Available from: http://www.cisco.com/web/solutions/trends/tech-radar/fog-computing.html.
33. Flavio Bonomi, R.M., Jiang Zhu, Sateesh Addepalli, *Fog Computing and Its Role in the Internet of Things.* Proceedings of the first edition of the MCC workshop on Mobile cloud computing, 2012.
34. Eduardo Cuervo, A.B., Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, Paramvir Bahl, *MAUI: Making Smartphones Last Longer with Code Offload.* Proceedings of the 8th international conference on Mobile systems, applications, and services, 2010.
35. Byung-Gon Chun, S.I., Petros Maniatis, Mayur Naik, Ashwin Patti, *Clonecloud: elastic execution between mobile device and cloud.* Proceedings of the sixth conference on Computer systems 2011.
36. Satyanarayanan, M., *Pervasive computing: vision and challenges* Personal Communications, IEEE, 2001.
37. Kristensen, M.D., *Scavenger: transparent development of efficient cyber foraging applications.* IEEE International Conference on Pervasive Computing and Communications (PerCom), 2010.
38. *C-RAN The Road Towards Green RAN - White Paper*. 2013, China Mobile Research Institute.
39. Rayal, F. *Cloud RAN vs. Small Cells: Trading Processing for Transport Cost*. 2012 [cited Dec 2014]; Available from: http://frankrayal.com/2012/03/17/cloud-ran-vs-small-cells-trading-processing-for-transport-cost/.
40. Parker, T. *C-RAN: Plotting next-generation wireless from inside the base station hotel*. 2014; Available from: http://www.fiercewireless.com/tech/special-reports/c-ran-plotting-next-generation-wireless-inside-base-station-hotel.
41. Hesham ElSawy, E.H., Dong In Kim, *HetNets with Cognitive Small Cells: User Offloading and Distributed Channel Access Techniques.* IEEE Communications Magazine, 2013.
42. *Android-x86 - Porting Android to x86*. [cited September 2014]; Available from: http://www.android-x86.org/.
43. *Libvirt - The virtualization API*. [cited October 2014]; Available from: http://libvirt.org/.
44. Morris, J. *Hash Tables*. Data Structure and Algorithms [cited October 2014]; Available from: https://www.cs.auckland.ac.nz/software/AlgAnim/hash_tables.html.
45. *Hashmap - Java Platform SE 7*. [cited October 2014]; Available from: http://docs.oracle.com/javase/7/docs/api/index.html?java/util/HashMap.html.

46. *ConcurrentHashMap - Java Platform SE 7*. [cited October 2014]; Available from: http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ConcurrentHashMap.html.
47. *HSQLDB - HyperSQL DataBase*. [cited November 2014]; Available from: http://hsqldb.org/.
48. *What is MySQL?* [cited November 2014]; Available from: http://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html.
49. HyperSQL. *PolePosition Results*.
50. *Xen Project Best Practices*. [cited September 2014]; Available from: http://wiki.xen.org/wiki/Xen_Project_Best_Practices.
51. *JUnit*. [cited October 2014]; Available from: http://junit.org/.
52. *Cisco Licensed Small Cell Solution: Reduce Costs, Improve Coverage and Capacity*. Cisco. Available from: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/licensed-small-cell/solution_overview_c22-726686.html.
53. *El Ingeniero de Telecomunicación: Perfil Socio-Profesional*. Colegio Oficial de Ingenieros de Telecomunicación.