UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

AN IMPLEMENTATION OF TASK PROCESSING ON 4G-BASED MOBILE-EDGE COMPUTING SYSTEMS

A Master's Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Circe Romero Uruñuela

In partial fulfillment

of the requirements for the degree of

MASTER IN TELECOMMUNICATIONS ENGINEERING

Co-Advisor: Jordi Pérez-Romero

Co-Advisor: Yusheng-Ji

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

telecom BCN

Barcelona, June 2019

TITLE OF THE THESIS: An Implementation of Task Processing on 4G-based Mobile-Edge Computing Systems.

AUTHOR: Circe Romero Uruñuela.

CO-ADVISORS: Jordi Pérez-Romero (Polytechnic University of Catalonia, UPC) and Yusheng- Ji (National Institute of Informatics, NII).

ABSTRACT

Mobile Edge Computing (MEC) is a new technology that facilitates low-latency cloud services to mobile devices (MDs) by pushing mobile computing, storage and network control to the network edge, thereby prolonging the battery lifetime of MDs. Besides, MEC aims to reduce latency and permit delay-sensitive applications in 4G communications. There is, therefore, a push to test MEC performance on existing cellular systems.

With the recently available mobile platform for academia SINET, NII can now connect MDs to ESs through 4G. This project focuses on the implementation of a physical 4G-based MEC System for task offloading, in which with the goal of achieving face detection, MD partially offload tasks to the ES under the instructions dictated by the offloading algorithms.

Accordingly, the objectives of this thesis are to prove the efficiency of LTE based MEC systems in the real world focusing on its performance in terms of latency and battery consumption.

Acknowledgments

First, I wish to show my greatest appreciation to Yusheng Ji for giving me the opportunity of coming to the National Institute of Informatics in Tokyo, for her support and mentoring during these months.

Moreover, I would also like to thank everyone in charge of the NII International Internship program for letting me participate in this amazing program and for their help and support prior and during my stay in the institution.

On the other hand, I would like to thank everyone in KLab at NII for their support and assistance special Yi-Han Chiang and Tian Zhang for guiding me along with the project with their deep knowledge and experience with Mobile Edge Computing. It has been an honor working with everyone here and learning from their methodology and experience.

Finally, I would like to express my deepest gratitude to Jordi Pérez-Romero who has been enormous support from Spain since the beginning of the project, not only for all his help but also for believing that I could do it and encouraging me to live this experience.

Revision history and approval record

Revision	Date	Purpose
0	02/02/2019	Document creation
1	26/06/2019	Document first revision
2	03/07/2019	Document last revision

Written by:		Reviewed and approved by:	
Date	03/07/2019	Date	03/07/2019
Name	Circe Romero Uruñuela	Name	Jordi Pérez-Romero
Position	Project Author	Position	Project Supervisor

Table of contents

List of Figures

List of Tables

List of Acronyms

AP	Access Point
CNTRL	Controller
DHCP	Dynamic Host Configuration Protocol
ES	Edge Server
eNB	Evolved Node B
KLAB	K Laboratory
LTE	Long Term Evolution
MD	Mobile Device
MEC	Mobile Edge Computing
NII	National Institute of Informatics
REQ/RESP	Request/response
RSLT	Result
RTT	Round Trip Time
SINET	Science Information NETwork
TTT	Total Task Time
UPC	Polytechnic University of Catalonia
WADCI	Wide Area Data Collection Infrastructure
WP	Work Package

1 INTRODUCTION, OBJECTIVES, AND REQUIREMENTS

1.1 BACKGROUND OF THE RESEARCH

As part of undertaking my final year of the master's degree in Telecommunications Engineering at the Polytechnic University of Catalonia (UPC) in Barcelona, I got the opportunity to participate in an International Internship[1] as a researcher at KLab[2] at the National Institute of Informatics (NII) of Tokyo, Japan.

This thesis is part of a collaboration project between the National Institute of Informatics and SINET-WADCI[3] (Wide Area Data Collection Infrastructure) platform. SINET is an academic information base that supports research activities of a wide range of universities and research centers throughout Japan establishing an academic information-sharing network nationwide.

Figure 1 SINET network configuration. Source SINET.

In 2018, a new access environment was added to SINET, a wide-area data collection base that uses the mobile network as a platform to connect to SINET that. on the other hand, allows in its wired section of speeds up to 100 Gbps. With this platform, mobile devices can reach our servers at NII through SINET and Softbank 3G/4G network, one of the largest telecommunications companies in Japan. Besides, there is a high chance that by September of 2019, they will gain access to a 5G network.

At KLab, Professor Yusheng- Ji, Yi-Han Chiang, and Tianju Zhang are focusing their studies on Mobile Edge Computing (MEC). My role at NII was to join KLAB's Mobile Edge Computing team and, by making use of the SINET-LTE link, implement a physical testing environment capable of testing offloading MEC algorithms. Moreover, this platform has been built to be scalable and to remain in KLAB for future experiments.

1.2 INTRODUCTION

Thanks to cloud computing, mobile applications can now offload their computation-intensive jobs to the cloud instead of relying on resource-limited mobile devices. However, sometimes offloading to the cloud can result in large delays which is a big problem for real-time-low-latency applications. To solve this, Mobile Edge Computing aims to migrate this solution to smaller-scale servers at the edge of the network.

MEC is an emerging technology that facilitates low-latency cloud services to mobile devices (MDs) by pushing mobile computing, storage and network control to the network edge, closer to mobile devices and thereby prolonging their battery lifetime. The key components of a MEC network include the mobile devices and MEC servers. Mobile devices and servers are separated by the air interface where reliable wireless links can be established using advanced wireless communications and networking technologies.

[1]https://www.nii.ac.jp/en/about/international/mouresearch/internship2019-1/
[2] http://klab.nii.ac.jp/
[3] https://www.sinet.ad.jp/en/top-en

MEC servers are smaller size data centers deployed somewhere closer to the end-user, usually they are co-located with wireless Access Points (i.e. the base station in cellular networks or the WIFI APs). Through a gateway, these servers are connected to the data centers via the Internet.

At KLab, thanks to the collaboration between NII and SINET-WADCI, Mobile Devices (MDs) can now through the LTE-SINET platform to Edge Servers (ESs) deployed on the SINET. Previously, other researchers in KLAB had centered their research in theoretical studies around MEC, and there was a motivation to also understand the performance of MEC in a real-case scenario. For this matter, my role at NII was to develop a physical LTE-based MEC environment where several edge computers and mobile devices could coexist.

As previously stated, one of the main objectives of MEC is to reduce latency and permit delay-sensitive applications in 4G and in the future, 5G communications. Thus, it was key that the mobile devices overcame a computation-intensive challenge. With the goal of achieving face detection, mobile devices offload tasks to the edge servers, under the instructions dictated by the offloading algorithms, in this manner this project has been able to evaluate the performance of both the implemented MEC algorithms and LTE-based MEC systems.

Accordingly, the objectives of this thesis are to create the test environment and to implement simple MEC offloading algorithms in the real world in order to have a better understanding of the benefits of using MEC technology on cellular networks.

1.3 Requirements and specifications

In order to deploy, Mobile Edge Computing in the platform it was necessary to build two separated planes. On one hand, the control plane, used for signally, in which mobile devices would send some information to the controller in exchange for an offloading MEC decision. On the other hand, the data plane, in which tasks (pictures in this case) would be offloaded according to what was dictated by the controller in the control plane, to achieve face detection.

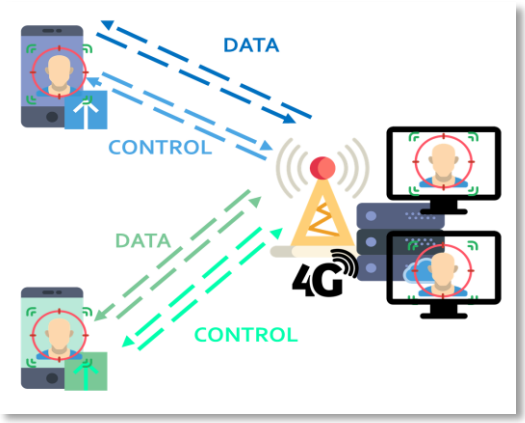

Figure 2 Simple project schematics

In consequence, the requirements for the project were:

1. Build up a 4G-based MEC system where multiple MDs and ESs coexist.
2. Measure the uplink/downlink delays in terms of varying data sizes.
3. Implement the request and response messages network for signaling in the control phase.

4. Collect a library of images for processing.
5. Achieve face detection in both mobile devices and edge servers.
6. Implement simple MEC offloading algorithms.
7. Study the data collected through the different tests and extract conclusions on MEC's performance in LTE.

Hence, the three key components of the testing network include:

1. **Mobile Devices** equipped with SoftBank SIM cards that provide LTE connection.
2. A **controller** in charge of running the offloading algorithms in the control phase and communicating the results to the mobile devices.
3. **"Edge Servers"** located in the SINET Network (at NII) and in charge of performing computationally intensive operations according to the offloading and scheduling decisions. As it has been discussed, MEC servers are typically co-located with wireless APs. However, because of the limitations in the collaboration between NII and SINET, in this project the edge servers have been placed in the laboratory and connected to the Base Station by a 100Gbps link though SINET.

1.4 STATEMENT OF PURPOSE

All of the facts stated in the introduction have set the reasoning on why this project was chosen to be developed. MEC is believed to be one of the technologies to be developed in the near future and even though initially it was thought to be only an only 5G feature, LTE-based MEC system is attracting considerable interest due to the widespread use of this cellular networks.

Moreover, it is well known that the use of smartphones has changed the way we interact with the world and numerous mobile applications are emerging every day. At the same time, these applications are becoming more battery and resource hungry and facing these two problems is set to become a vital factor in their growing evolution.

This project is an implementation of an LTE-based MEC system that integrates MEC offloading algorithms to perform the battery and resource-hungry task of face to detection. The goal of the project was to use this platform to evaluate the performance in a real case scenario of the implemented MEC offloading algorithms and most importantly, 4G-based MEC networks. Accordingly, the objectives of this thesis are to:

Objective 1. Build a network where multiple Mobile Devices and Edge Servers coexist.

Objective 2. Test the network and measure the transmission delays.

Objective 3. Implement a computation-intensive face detection application that motivates the use of MEC.

Objective 4. Measure the processing time of the face detection application and evaluate its performance.

Objective 5. Leverage the SINET-WADCI platform to evaluate the mobile-edge computing (MEC) offloading algorithms.

Objective 6. Acquire insights into the performance of Mobile Edge Computing for Long Term Evolution cellular networks in terms of latency and battery consumption.

1.5 METHODS AND PROCEDURES

This section moves on to present the software previously developed by other authors that are used to complete the project.

In accordance with what was presented in the objectives and requirements of the project, this project consists of the creation of the test environment including both, the establishment of the communications and the implementation of the face recognition library in all network devices. As it will be later be explained in section 3.2 of this document for the face recognition applications in both the server and the phones Dlib is used. In particular, for Android, I have used as a baseline the Android application that was created by *Tzutalin* available on GitHub [1].

On the other hand, for the evaluation of the test consumption a free Android application, *Simple Battery Graph,* available in Google Store was used. This application measures the battery consumption over time and permits to export the data to handle it over Microsoft Excel.

1.6 WORK PLAN WITH TASKS, MILESTONES AND GANTT DIAGRAM

The next step was to subdivide the pending tasks into smaller work packages (WP) and to define a project planning strategy. Work packages form the lowest level of planning and consist of smaller activities that if well-structured, help to distribute the work over time and facilitate the completion of the project. Milestones, on the other hand, are smaller goals that show an important achievement in the project.

This section moves on to describe the project execution divided into eight work packages and all the milestones that have set important accomplishments during these months.

WP1. PROJECT KICK-OFF.

This work package was essential for the correct development of the project since it set the requirements and objectives of the project and defined a work plan in order to achieve them within the available time. Moreover, it included taking an introductory Android course to get familiar with the platform and Java.

- Study and understand the state of art of the current technology.
- Understand the requirements of the project.
- Set up the project objectives and plan.
- Introductory Android course.

Milestone 1. Project kick-off and project plan definition.

WP2. NETWORK SET-UP

The next step after the initialization of the project was the hardware implementation where all the devices required in the network were interconnected and evaluated for correct functioning.

- Network implementation.
- Routing table configuration and ping test.

Milestone 2. Hardware set-up.

[1]https://github.com/tzutalin/dlib-android-app

WP3. CLIENT-SERVER COMMUNICATIONS

Once the checking that the network was correctly set-up, WP3 focused on developing the client/server software in order to send a larger amount of data.

- Selection of the communication protocol.
- Android Client implementation on Mobile Devices (MDs)
- Node.js Server development on the Edge Servers (ESs)
- Client-server communication and message exchange.
- Network testing.

Milestone 3. Successfully achieve client-server communications at an acceptable rate.

WP4. FACIAL DETECTION IMPLEMENTATION

Since it was required to equip MD and ES with facial detection capabilities, work package 4 is in charge of adding this feature. A computation-intensive application was needed in both the server and client in order to see the benefits that using edge computing could bring.

- Mobile devices facial detector implementation.
- Server facial detector implementation.
- Test the performance and processing time in both mobile devices and servers.

Milestone 4. Achieve face detection in all devices that form the network and evaluate the processing delays.

WP5. ALGORITHM IMPLEMENTATION AND FINAL SET-UPS

The next step was to implementation of very simple offloading algorithms in the controller which has a full view of all the devices in the network.

Milestone 5. Implementation of MEC offloading algorithms.

Finally, it was time to install the client application and server code in all the devices of the network, modify the message been sent and do final fixes to ensure the correct functioning of the project.

- Install android code in MD
- Install sever in both ES and controller

Milestone 6. Integration of control and data plane and final set-ups.

WP7. MEC TESTING

WP7 consisted of performing the last tests, focusing on the evaluation of the MEC algorithms and LTE-MEC systems. One of the main steps of this work package was to obtain insights from the feedback in the final presentation at NII. This presentation was planned before the project finalization in order to do any possible improvement according to the comments.

- Perform the tests of the MEC platform and conclusion extraction.
- Present results at NII and obtain feedback.
- Repeat any necessary test and do last improvements according to the feedback of the final presentation at NII.

Milestone 7. Test LTE-MEC platform and extract conclusions.

WP8. THESIS DOCUMENTATION

Finally, WP8 constitutes all the presentations and papers needed to hand throughout the course of the project. Both UPC and NII required different documentation thus WP8 encompasses all such procedures.

Milestone 8. Initial presentation. Definition of the project and the objectives.

Milestone 9. Mid presentation. Presentation of the progress half-way through.

Milestone 10. Final presentation. Summarizing the work done, conclusions, possible improvements, and continuation.

Milestone 11. Thesis document.

Milestone 12. Presentation at UPC.

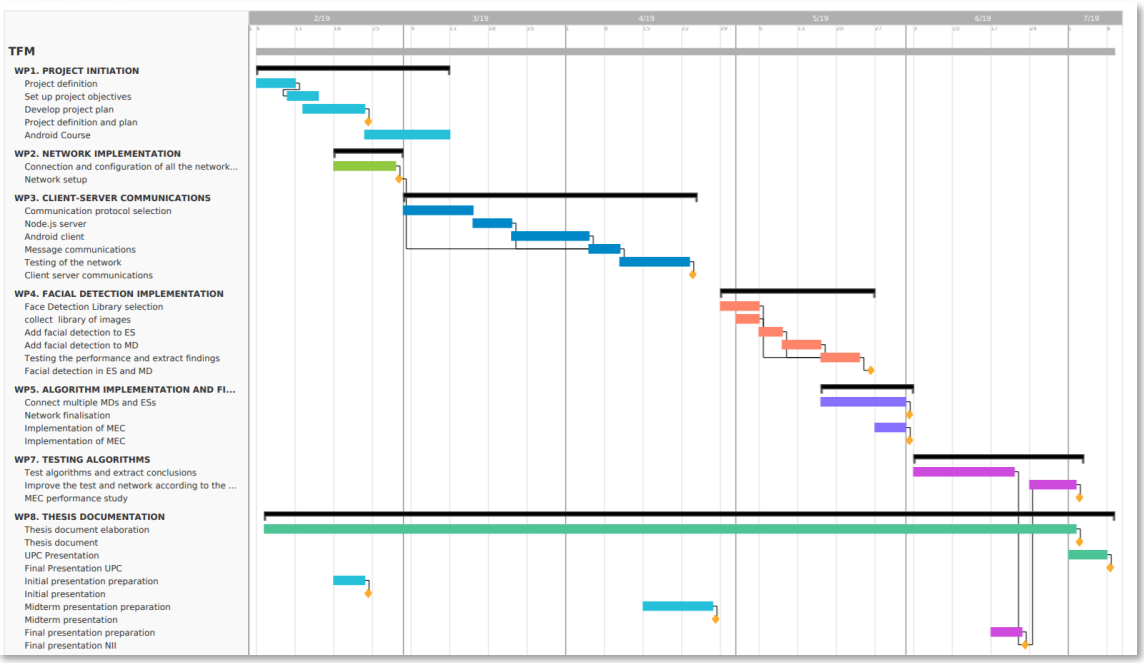

Figure 3 Gantt Chart. Made with: https://prod.teamgantt.com.

1.7 DESCRIPTION OF DEVIATION FROM THE INITIAL PLAN

As presented, this master thesis is part of a collaboration project between SINET and KLAB's MEC team. Even though the project plan was created to be quite flexible and it was open to deviations and changes there were some that were not taken into account.

The main deviation of the plan that has the biggest influence in the outcome of the project was the change in the algorithms to be tested. Initially, this project was meant to be a collaboration project with another researcher at NII that was working in the design of offloading and scheduling algorithms. Unfortunately, due to circumstances that are out the scope of this project these algorithms were not ready during the development of the project and the project made a turn into another direction.

The idea was to spend from week 14 to week 16 understanding and implementing the algorithms provided, however, this was postponed up to week 16 when I was informed that the algorithms were not going to be ready on time and had to reschedule other areas of the plan to fit this complication. In the end, the project finalization was possible by the implementation of the random and round-robin offloading algorithms and FIFO (First In First Out) as the scheduling policy.

Aside from the change in the algorithms to be implemented, no big issues or problems have occurred during the project. However, and due to my inexperience in research and software development I think I have underestimated the many small deviations that could happen for very small things and I would have liked to plan this small deviation in my initial plan to avoid the high stressed level of the last two months of the project.

After this experience, I have learned that in the same manner as the project plan was perfectly modeled with the objectives and requirements it would have been very useful to do a project risk study by identifying what could have gone wrong, characterization and prioritization of these risks and to create a handling and monitoring plan for each of these cases according to its possible repercussions.

1.8 STRUCTURE OF THE DOCUMENT

Overall this thesis summarizes from start to end the development of the 4G Based MEC platform and the testing of the different components focusing on the performance of Mobile Edge Computing for LTE for the special case of Face Detection in a network composed by two edge servers and two mobile devices.

For this, this document is been divided into six chapters. In chapter 2, a brief introduction will cover the state of art of MEC technology and a small introduction of the theory behind the implemented face detection library. Next, chapter 3, will move on to present the development of the project that has been divided into three phases: control phase deployment, data phase implementation and offloading algorithms implementation and final setups.

Afterwards, chapter 4, focuses on presenting the different tests performed during the development of the project. And, in the same manner, as the project development, it has divided into three parts: first, it focuses on the evaluation of the delay due to data transmission, next it puts its attention in the processing delays, finalizing with the study the behavior of Long-Term Evolution. LTE has been proposed as one of the cellular technologies that could benefit from Edge Computer, therefore there is a push to study its performance in a real case scenario. By the use of simple algorithms, insightful results about the two main focuses of MEC have been performed: latency and battery consumption.

Finally, chapters 5 and 6 include the project's budget and the conclusions and possible project continuation respectively.

2 STATE OF THE ART OF THE TECHNOLOGY

Chapter 2 moves on to present the state of art of the technologies used in this investigation, it will be divided into two very well distinguished parts.

First, this theoretical framework will explore the current state of Edge Computing. A historical background will present the evolution of cloud computing and its limitations, followed by a detail explanation of the role of Edge Computing in cellular networks. Next, edge computing will be presented, an overview of the importance of offloading and scheduling algorithms.

To finalize, the technology behind the face detection library will be briefly introduced, focusing on the HoG and Support Vector Machine.

2.1 EDGE COMPUTING

The evolution of information technology has been accelerated by the growth and expansion of mobile devices. Smartphones have gained an important role in society and they are becoming more sophisticated. This evolution has been especially important in the areas of cloud computing and wireless communications. Mobile devices are equipped with various network interfaces, such as Long-Term Evolution (LTE) and Wi-Fi, and can connect to "the cloud" anytime and anywhere. At the same time, Cloud Computing has become a very important paradigm of computing delivering elastic computing power and storage to resource-constrained end-user devices.

As a result of all these technological breakthroughs, developers worldwide are building complex applications, such as face recognition, interactive gaming, natural language processing and virtual reality to augment even more the capabilities of these gadgets. However, all these emerging applications are usually resourced hungry and require high amounts of computational power and energy which is an immense challenge for resource-constrained mobile phones. At the same time, 5G has brought the concept of ultra-low latency to the table, and it is widely agreed that relying on cloud computing is inadequate to realize the ambitious millisecond-scale latency for computing and communications.

For all of that, a new concept known as Mobile Edge Computing is emerging. The main feature of MEC is to push mobile computing, network control, and storage to the edge of the network, reducing the latency that usually involves using a centralized cloud. By offloading the computation partially or totally to the cloud infrastructure, MEC is envisioned as the most promising to technology to overcome these challenges.

2.1.1 From Cloud to Edge Computing

Cloud computing is currently one of the fastest-growing sectors of the IT industry. It is a general term that refers to anything that involves delivering hosted services over the Internet or in other words, it is the practice of storing and processing information in remote servers on the Internet instead of locally.

Cloud services differ from traditional local servers in two main aspects: its scalability and the easiness in deployment for the service's consumer. The ability to scale computing capacity up or down on-demand basis gives the opportunity to share the resources according to the real demand. On the other hand, that the service is fully managed by the provider, gives the opportunity to any kind of enterprise, big or small, to deploy these services according to their necessities without the initial investment that buying the infrastructure requires. Companies are increasingly aware of the business value that cloud computing brings and have already taken steps towards the transition to the cloud.

Although there have been massive improvements and development around cloud computing, this technology still faces the challenge of delivering reliable and available services [1]. An evident weakness is the long data-exchange latency that mobile users' experiment which makes cloud computing inadequate for a wide range of emerging mobile applications that are latency-critical. Although this long latency may seem irrelevant for applications like emails or web browsing, it becomes critical for others such as video streaming or gaming, in which long delays would hurt the interactive response because of human's sensitivity to delay and jitter.

To overcome these limitations, over the last few years cloud computing is going throw a fundamental shift in which the traditional highly centralized model [2] would be replaced by a distributed decentralized one, with the function of the clouds being moved towards the network edges. This new concept is known as edge cloud computing and aims to bring closer to the user, cloud capabilities such as computing, networking, and storage, reducing significantly the latency roundtrip to the cloud. The concept of MEC was first defined by the European Telecommunications Standard Institute and was defined as "*a new platform that provides IT and cloud computing capabilities within the radio access network (RAN) in close proximity to mobile subscribers*". [2]

Since Edge Computing is still an emerging technology some of its characteristics are to be defined but for simplicity, edge cloud computing could be subdivided into two types: FoG, that pushes technology down to the local area network, and Edge Computing, that as the name suggests, pushes it to an edge gateway or access point.

In terms of applications, a wide range of new applications and services are emerging with the Internet of Things and 5G. Edge computing has already been proven to be beneficial for many computationally intensive applications such as gaming, healthcare, Internet of Things or video streaming [4] Edge servers allow for delay-sensitive applications to be executed externally but closer to the end-users. This paradigm is crucial for enabling low-latency, high-bandwidth, and agile mobile services. [5] This technology is an important target for the business since it will give mobile phones higher computational power, with a reduction in battery consumption and without the need for any extra hardware within the devices.

Some of the benefits of adopting MEC will bring include:

1. **Low latency**. Compare to cloud services approaching the servers closer to the end-user at the edge of the network would result in a huge enhancement regarding latency and will allow for new latency concerned applications.
2. **Mobile energy savings.** By transferring computation to the network, it is expected that mobile devices will significantly reduce battery consumption when using these computation-intensive applications.
3. **Context-awareness.** By tracking user real-time information such as behavior and location, more accurate context-aware services could be delivered to the end-users.
4. **Privacy enhancement,** by distributing the information from big unique data centers to smaller distributed ones.

For all these reasons, edge computing technologies have attracted much attention to cellular networks.

2.1.2 Edge computing for Cellular Communications

In the pre-smartphone era, before IP communications, when voice quality was the highest necessity, the edge of mobile networks was a place for only specialist processing and that was designed to perform a specific function. The complete network, from the edge to the core, was optimized for an only purpose resulting in equipment very little scalable. With the widespread adoption of over IP technologies such as LTE, new applications have emerged, and the industry has seen a whole transformation in the network design. Single vendor radio network solutions

are evolving into modular open solutions capable of being integrated into a changing environment. [2] Mobile-access edge computing is seen as a key technology to be incorporated into these new more flexible networks in order to bring application-oriented capabilities to the heart of carrier's networks to broad the horizons of what we now know as cellular network and, in order to explore a wide range of new use cases, especially those with low latency requirements.

Sometimes MEC is considered a 5G only feature, nevertheless, since 4G is still far from disappearing, research is being done towards adding MEC to existent 4G networks. Actually, according to the ETSI [1], the defined MEC reference architecture is independent of whichever cellular network it is deployed in. Products based on current MEC specifications can be smoothly migrated to support 5G networks through a software update. This way, flexibility in the deployment architecture allows planning for the introduction of MEC services as the milestone to build the edge cloud, which is key for the success of 5G services. In this manner, MEC for LTE would not only improve the current network but it will act as a driver to motivate 5G adoption. [6]

There are a few potential deployment options available for operational 4G systems explored by ETSI. However, the requirements for deployment of MEC in LTE are clear: it must suppose the minimum investment, it must be standard-compliant, and it must not introduce any modifications on eNB or the core network. The key idea is that the MEC platform sits within the eNB architecture and some traffic is routed to the MEC and others pass through the MEC to reach the Internet. Edge servers act as a middlebox transparent to the network architecture.

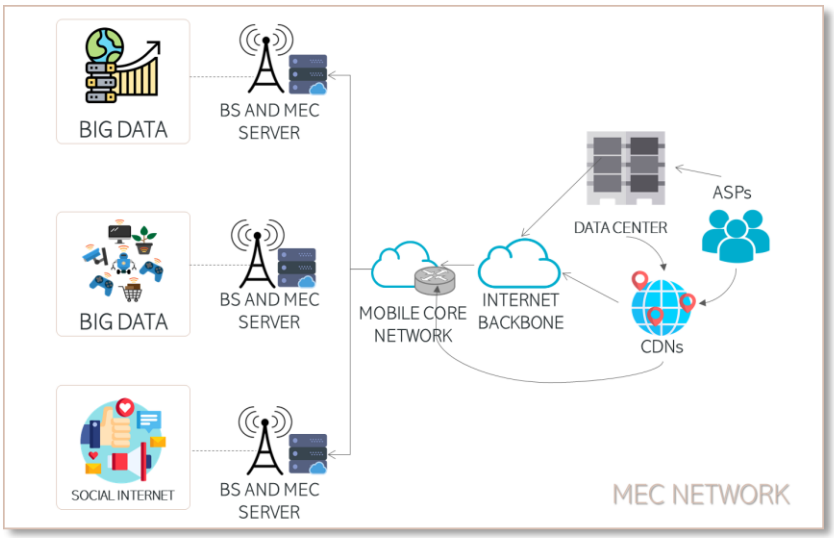

Figure 4 MEC Network concept.

Besides the challenges that deploying MEC in LTE supposes, also the wireless nature of the channels could be a big challenge. [7] The wireless channel condition plays a big part in the performance of MEC: bad channel conditions increment battery consumption and latency and could make the use of MEC disadvantageous. For that is very important to build efficient MEC algorithms and use advanced techniques such as interference cancelation and adaptive power control.

2.1.3 MEC: offloading and scheduling algorithms

Two of the biggest issues in the spotlight of the academic MEC researches are task offloading and scheduling. The former means transferring resource-intensive computational tasks to an external platform, the edge server. The latter refers to the order the server decides to process the arriving tasks.

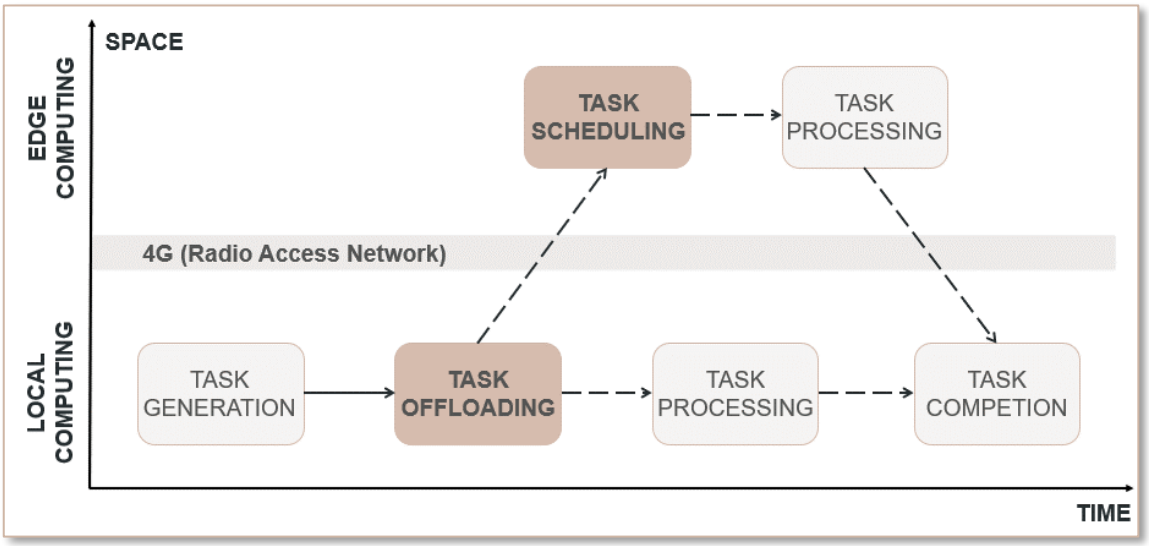

Figure 5 Offloading and scheduling in MEC

Task offloading incurs extra overheads in terms of delay and energy consumption due to the communication required between the devices and the MEC server in the uplink wireless channels. Because of this time-varying channel condition, it is not necessarily optimal to always offload the tasks to the server, for example when the channel is in deep facing conditions. This hence can lead to low energy efficiency and long data transmission time and offloading would result detrimental.

Additionally, in a system with a large number of offloading users, the finite computing resources at the MEC servers considerably affect the task execution delay. How the tasks are prioritized within the server would be crucial to minimize the processing time of the overall system.

Therefore, offloading and scheduling decisions become a critical problem toward enabling efficient computation. Thus, the two main goals in MEC are:

1. For the **mobile users** to determine whether or not to **offload**, select an edge server and the amount of the computation to offload
2. For the **servers** to dynamically **schedule** the offloaded tasks to optimize the network performance.

As it will later be presented in section 3, this project focuses on the performance of multi-user multi-server MEC systems and low-complexity offloading algorithms (Round Robin and random) are implemented to minimize the average total task time. Concerning scheduling on edge-clouds, an online scheduling scheme without any assumption about the distribution of the job releases is desired. First-Come-First-Serve scheme has been selected as the server's strategy to perform the tasks.

2.2 FACE DETECTION

There is an increasing number of emerging mobile applications that could benefit from MEC one of them is face detection. An image detection algorithm is a computer vision-based algorithm that is capable of detection because it has been trained to learn the difference between different elements. Therefore, a face detection algorithm takes images as input and after some computation, outputs whether a face is present or not. The method of face detection in pictures is complicated because of the high variability in human faces including expression, hair, facial hair, skin color, lighting conditions…

The library that has been used for this experiment is called Dlib and works by computing HOG features and classifying them with an SVM (Support-Vector Machines). This section moves on to describe an overview of these theoretical concepts to have a better understanding of the detection library.

2.2.1 Histogram of Oriented Gradients (HOG)

As it has been presented Dlib and uses a concept called Histogram of Oriented Gradients (HOG) which is an implementation of the original paper by Dalal and Triggs [11]

HOG extracts feature into a vector which is then classified by Support Vector Machine (SVM). The features that are extracted are histograms of directions of gradients of the image that indicate where the edges and corners are in each picture region. [12] [13] [14] This feature descriptor is obtained in 5 simple steps:

1. Pre-processing to normalize gamma and the color of the input image.
2. Gradient calculation. Calculate the x and y gradient images from the original image. The gradient image removes a lot of non-essential information (e.g. constant colored background) and highlights outlines.
3. Divide the image into cells. Then, for each of these cells, compute the HoG directions or edge orientations for the pixels within the cell.
4. Grouping and normalization of histograms. Grouping the cells into blocks according to their weighted gradients.
5. Descriptor or feature vector. A normalized group of histograms represents the block histogram. The set of these block histograms represents the descriptor.

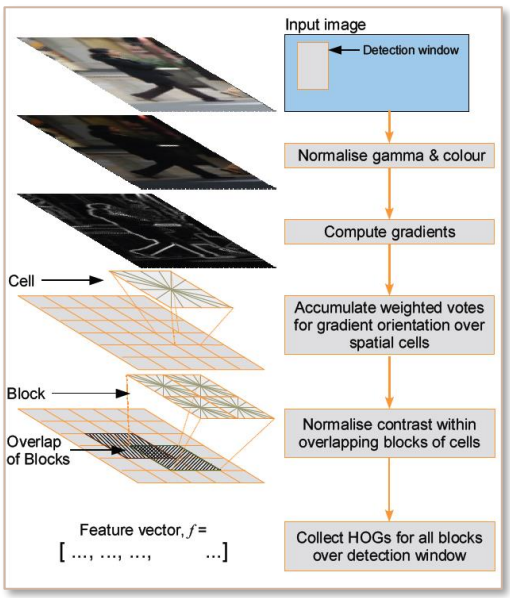

Figure 6 Histogram of Oriented Gradients working scheme. [15]

2.2.2 Support Vector Machine (SVM)

DLIB uses Linear SVM and is been trained with a dataset of 2825 images. This algorithm performs a simple supervised binary classification. [16]

To simply understand the basic concept of the algorithm we can simplify by thinking of a result feature vector from HOG of just two dimensions, which can be represented as the two colors (black and white) in *Figure 7.* These two features could be, faces and background.

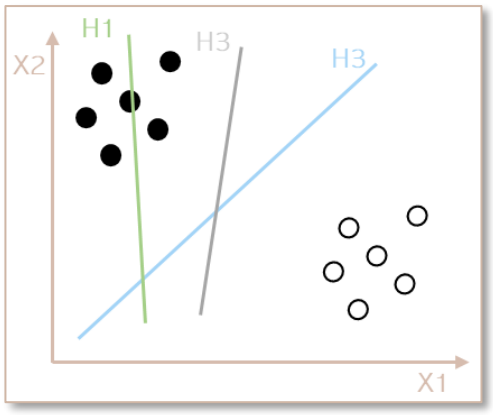

Figure 7 Support Vector Machine for two dimensions

Then, we provide the algorithm with examples of the two classes and then we tell the algorithm to organize the dots as black or white. Different learning algorithms separate these two classes in different ways, in the case of Linear SVM tries to find the best line that separates the two classes. As we can see in *Figure 7*, the best line would be H3, as it perfectly separates the members of the two classes and it is at maximum distance from both types.

In the same manner, if the feature vectors are in 3D, SVM will find the appropriate plane that maximally separates the two classes. And in the case of DLIB, where the feature vector is in a 3780-dimensional space, SVM will find the appropriate hyperplane.

3 METHODOLOGY PROJECT DEVELOPMENT

The present chapter aims to explain the methodology followed during the thesis development. The projects prime objectives were to leverage the SINET-WADCI platform to evaluate the benefits of mobile-edge computing (MEC) through the implementation of Round Robin and Random MEC-offloading algorithms. Thus, the first step is to construct a practical MEC system which ultimately, would allow the MDs and ESs to realize the missions of face detection, following the offloading decisions dictated by the controller, and the computational results would finally be sent to hosting MDs through result messages.

As reported in the work plan presented in the first chapter of this document, the development of the project was divided into three phases:

- **Phase one** main focus was to initiate the **development of the Control Plane**. The objectives or this phase were to set up a simple network environment that connects mobile devices and the controller and enabled the exchange of request and response messages between them.

- **Phase two** consisted of the **deployment of the Data plane.** As previously presented, in order to prove the efficiency of the algorithms, both edge servers and mobile devices need to be equipped with computationally intensive applications. Bearing this in mind, a face detection application was deployed in both MD and ESs.

- Then, **phase three** consists of the deployment of the algorithms in the platform the final setups. This last phase of the project will be complemented with Chapter 4 and the evaluation of the applicability and effectiveness of MEC by summiting the simplistic designed solution through extensive experiments on the SINET-WADCI platform.

Therefore, the project is clearly been divided into three phases. First, the implementation and testing of the network and control plane. Second, the implementation and testing of the data plane and lastly, the deployment and evaluation of simple MEC offloading algorithms to extract insightful conclusions.

3.1 PHASE 1: IMPLEMENTATION AND TESTING OF THE NETWORK AND CONTROL PLANE.

The initial step in the development of the project was to build the network, including the hardware configuration and the software implementation.

3.1.1 Hardware configuration: Set-up of the network environment

The very first step to take was to interconnect and properly install the main components of the network. The system comes complete with the use of the following devices:

- *Two PCs, Lenovo Legion T730 with Intel Core i9-9900K processor.* These PCs are used as the Controller (which also hosts Edge Server 1) and Edge Server 2.

- *Two mobile devices (MD), Samsung Galaxy S9 with Exynos 9810 processor,* equipped with LTE Softbank SIM cards that connect to the SINET-WADCI network.

- *NII Ethernet cable[13B-OL-008],* connected to the SINET-WADCI network. It is important to note that this cable is the connection between the NII laboratory and

SINET and therefore it had already assigned an IP address and a gateway within the SINET network.

- IP address: 10.1.1.17.
- Netmask: 255.255.255.252.
- Gateway: 10.1.1.18.

• *CE Router, AirMac Extreme A1408*. Connects to the Ethernet cable and creates a private local network within the laboratory.

Thanks to *Softbank* SIM cards, the mobile devices can connect through LTE to the SINET-WADCI network. On the other hand, the Controller and ESs, located in the laboratory are connect to the network through CE router which is the laboratory's gateway to the SINET network. *Figure 8* is a schematic representation of the network configuration end to end.

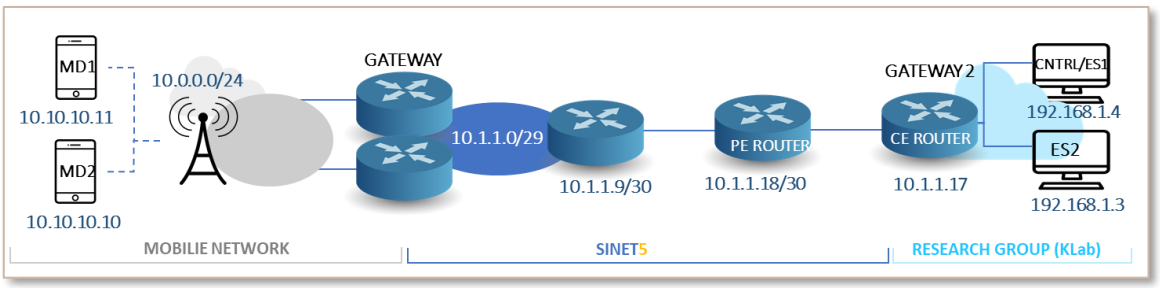

Figure 8 Network configuration

The procedure followed to interconnect the network elements consisted of three simple steps:

1. Installation of Ubuntu 16.04 in the PCs: T730 (1) which acts as edge server 1 and the controller and T730 (2) which is the edge server 2.

2. Plug the NII ethernet cable [13B-OL-008] into the router [AirMac Extreme A1408] and ping from the Mobile Devices to the PE and CE routers to ensure the connection worked smoothly.

3. Since KLAB was provided with only one IP from SINET, the final steps consisted of using CE router to set up a Local Network (LAN) and to configure the routing tables to let any information from outside the LAN reach our PCs. CE router is connected to the ethernet cable 10.1.1.17 and creates a LAN for the controller (also hosting ES1) and the ES2 in the domain 192.168.1.x. CE router receives the information sent by the phones towards a specific port of 10.1.1.17 and then, thanks to routing configuration, that information is redirected towards the corresponding device. The routing tables were configured as represented in table 1.

   Thus, T730(1) hosts the controller and the ES1 at 192.168.10.4 and T730(2) serves as ES2 at 192.168.10.5.

Packets sent	Redirected to	Network	Device
10.1.1.17:8004	192.168.10.4:8004	Controller/Edge	T730
10.1.1.17:8005	192.168.10.5:8005	Edge Server 2	T730

Table 1 Routing table configuration

As can be seen in *Figure 8*, this is not a pure edge computing deployment. In a real future MEC scenario, the edge servers would be deployed within the eNB (as explained in section 2), however, due to the limitations of NII agreement with SoftNET the Edge Servers are deployed within the SINET network. Still, this positioning of the edge servers is considered to be in a much closer location than a Cloud Server, which could be hundreds of kilometers away from the end-user. To have a better understanding of all the hops the message goes through from the devices to the edge servers we can look to a traceroute of a simple ping message.

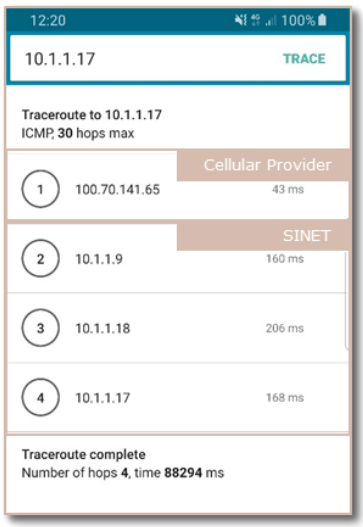

Figure 9 Ping Traceroute from 10.10.10.10 (MD) to 10.1.1.17 (CE Router)

3.1.2 Software implementation: Client-server communications

The experiment proceeds with the software implementation that was required to achieve client-server communications. Before taking the decision of what technology would be used as the communication protocol it was important to understand the network's main characteristics and requirements:

- The network is formed by edge servers acting as servers and mobile devices, acting as clients.
- Both, the mobile devices and the edge servers must be able to send information at any time.
- One of the key objectives of mobile edge computing is to reduce latency, therefore, the link must allow real-time communications.

As can be seen the software implementation had to be split into two parts, an Android app hosted by the Mobile Devices and Servers, hosted by the PCs. These components must had been able to communicate real-time at any time. To achieve this feat, Socket.io was chosen and the following subsection moves on to present the motives behind this decision.

3.1.2.1 Socket.IO vs HTTP

Socket.IO is a library that enables real-time, bidirectional and event-based communication between the client and the server. One of the main goals of Socket.IO is reliability for this aim it first establishes a long-polling connection, then tries to upgrade to better transports, usually WebSocket. Socket.IO allows using WebSockets very easily, making real-time, bi-directional communication between clients and server possible. [17] The main characteristics of Socket.IO are:

1. Multiplatform. Same as HTTP, Socket IO has two parts: a client-side library that runs in the Android phone, and a server-side library for Node.js.

2. Bilateral communications. Socket.io also provides the benefit that both the server and client can push messages to the other at any given time, allowing for full duplex client-server communication. This protocol goes beyond the typical HTTP request/response paradigm. Unlike HTTP, Socket.io allows a sort of communication that remains open between the client and the server. They stay connected to each other and can exchange messages from any side node at any time.

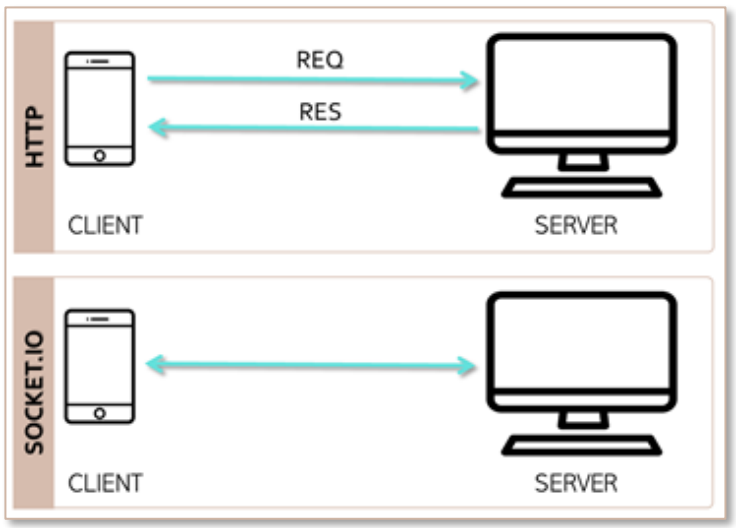

Figure 10 HTTP request-response vs Socket.io bidirectional communications

3. Event-driven model. Socket.io has the ability to work with an event-driven model: both the server and the client can react to events and messages.
4. Lower overhead. Excluding this initial connection setup (over HTTP), a Socket.io connection does not have to send headers with every message resulting in lower overhead.
5. Transfer speed. Due to all of the previously stated characteristics Socket.io allows for a faster transmission rate.

These characteristics have been summarized in the following table:

	HTTP	SOCKET.IO
Client/server	Supported.	Android and Node.js libraries.
Communication	Unsynchronized. Client request-server response.	Full duplex synchronized bilateral exchange. Event-driven comm.
Connection	TCP connection. New connection every new message (open/close).	Single TCP connection for each client.
Data transfer	Data attached to every packet (headers).	Lean protocol after the HTTP establishes the connection.

Transfer speed	Slow.	Created to support real-time over TCP applications.

Table 2 Summary of the differences between HTTP and Socket.io

An easy way to understand the performance difference between HTTP and Socket.io is by measuring the Round-Trip Time over time when using Socket.io. As it has just been explained Socket.io initiates the communication with HTTP moving on to an open bidirectional WebSocket communication afterward. *Figure 11* shows how the RTT using HTTP reaches 300 ms while the rest of the time this latency is below 100 ms.

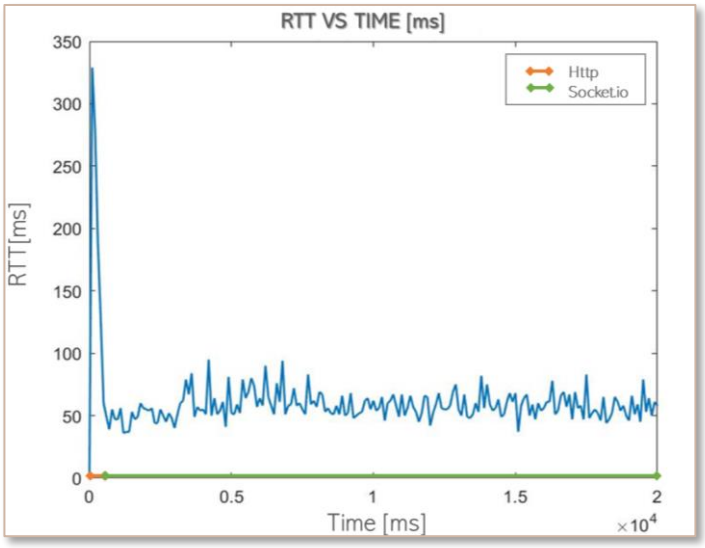

Figure 11 Socket.io performance. HTTP vs Socket.io

3.1.2.2 Socket.IO for Node.JS and Android.

Once the decision of using Socket.IO as the communication protocol was made, the next step was its implementation. As it has been stated, Socket.IO is available for both Node.js and Android. Communications are possible following two simple steps:

1. Establishing communications: connect/disconnect.

   - **Node.js Multi-client server,** the server in Node.js is written in JavaScript. *Figure 12* shows how Socket.io is initiated.

```
var express = require ('express');
var app = express ();
var server = require('http').createServer(app);
var io= require ('socket.io').listen(server);
//SERVER LISTENING TO THE PORT CONFIGURE IN THE ROUTING TABLE
server.listen(process.env.PORT || 8004);
io.sockets.on ('connection', function(socket){
//ALL THE EVENTS THAT WE NEED TO HANDLE ARE GOING TO BE IN HERE
```

Figure 12 Node.js Socket.io Server connection establishment

Socket.IO servers are initiated listening to the port 8004 (for ES1) and 8003 (for ES2) , which is where all the packets directed to PC1 and PC2 will be redirected. Socket.IO has some reserve events.

- **Io.sockets.on ('connection', ...)** is used to establish the connection. Every time a new user connects the server will show a message of the number of users connected. Within this connection, every user name is assigned with an ID.

All the events that we need to emit are going to be inside this function.

- **Socket.on ('disconnect', ...).** Every time a socket is disconnected this event would be called.

▪ **Android Client.** Socket.io client will connect to 10.1.1.17 and to the port we want to redirect the information to.

```
//Connect to 17 and the port that redirects to the
local network
{ try { mSocket = IO.socket("http://10.1.1.17:8004");
    connected= true;
} catch (URISyntaxException e) {
    Log.d("myTag", e.getMessage());}
```

Figure 13 Android Socket.io Client connection establishment

2. Sending and receiving events. The main idea behind Socket.IO is that it is event-driven communications, so any device can send and receive any events at any time. Any objects and data that can be encoded as JSON can be sent over Socket.IO.

▪ **Node.js Multi-client server.** Node.js server receives information through **Socket.on ('event', ...).** It is important to understand that there are two ways of sending information from the server.

- **Io.emit ('event', ...).** This event and its information will be sent to every connected client.

- **Socket.emit ('event', ...).** This will be sent to a specific socket, it can also be done by using **io.to**(#socketID**). emit ('event', ...).**

```
//TEXT MSG (RECEIVED AND RESEND) TO TEST THE CONNECTION
socket.on('send message', function(data){
    socket.emit('new message', {msg:data});});
```

Figure 14 Receive and send the same message to the device that sent it. Server side.

```
//TEXT MSG (RECEIVED AND RESEND) TO TEST THE CONNECTION
socket.on('send message', function(data){
  var id= socket.id;
  io.to(id).emit('new message', {msg:data});});
```

Figure 15 Receive and send the same message to the device specified by the id. Server side.

▪ **Android Client** The client event emission and reception have the same structure as in the server, however, it can seem a little bit more complex due to the nature of Java itself.[18]

- **Socket.emit ('event', ...)** is used to send information from the client. In the case of the control plane, it would be a JSON object containing all the necessary information for the controller to make the offloading decisions.

- **Socket.on ('event',** *EmitterListener***).** As it has been constantly stressed, Socket.IO permits bidirectional communications and thus, every time the server sends an even to the client, the second must be listening. In Java this can be implemented through an event listener. An event listener is designed to processes an event and it is always listening waiting for this event to respond accordingly.

```
//Send message where send text is an JSON object
mSocket.emit("send message", sendText);
...
//Receive a message through a listener
mSocket.on("new message",handleIncomingMessages);
...
private Emitter.Listener handleIncomingMessage = new
Emitter.Listener(){
    @Override
    public void call(final Object... args){
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
             JSONObject data = (JSONObject) args[0];
             try {
               String result = data.getString("msg")}
               catch (JSONException e){ return;}});}};
```

Figure 16 Send and receive a message. Client side.

Once communications were achieved and before the moving on to the implementation of the data plane, some tests were performed to understand the performance of the implementation. Once, it was proved that the round-trip values were within the acceptable delays for the application, the next phase's implementation could start. These results will be presented in *Section 4* of the document with all the tests related to the experiment.

3.2 Phase 2. Implementation and testing of the Data Plane

The goal of this thesis is to understand the performance of Edge Computing for LTE communications. One of the candidate applications to be suitable for this type of platform is Face Detection. Accordingly, phase two consists of the implementation, as part of the data plane, of the face detection in both servers and clients.

3.2.1 Face-detection implementation

Face Detection is a computer vision application that consists of spotting the number of faces present in a photo. Picture processing is usually slow and computationally intensive and therefore, it makes sense to offload some or all of the tasks to the edge servers in order to reduce latency and improve the overall performance.

There are a few libraries available that can implement face detection however the criteria for selecting which one was more suitable for this experiment was based in three essential requirements:

- The processing delay has to be as little as possible.
- It must be able to efficiently detect faces with a relatively low error rate.
- It must be able to implement in both the Android phones and the Edge Servers.

One algorithm that suited all these requirements was DLIB's HOG face detection model.[19] As stated in [20] it is one of the fastest open library methods on CPU and works very well in frontal and slightly non-frontal faces.

The ultimate reason for DLIB being chosen as the Face Detector library was its availability for both Python and Android. Having the same library in both smartphones and servers removes the uncertainty on whether or not the difference in performance in-between libraries has an influence on the overall system behavior and allows a more accurate and easier analysis of the difference between using or not using MEC.

Face detection in DLIB is a model, based on HoG features and SVM. This model was built out of 5 HOG filters (front, left and right looking, front looking but rotated left, and a front looking but rotated right) and it was trained by his author with a dataset of 2825 images.[20]

3.2.1.1 *Edge Servers.*

First, when it comes to the deployment of Face Detection on the Edge Computers the solution was to implement Dlib in Python. Following the principles dictated by the example "*FaceExample.py"* provided by Dlib (Figure 17), face detection was achieved.

```
import sys
import cv2
import dlib
import time

detector = dlib.get_frontal_face_detector()
img = cv2.imread(f)
dets = detector(img, 1)
out=format(len(dets))
```

Figure 17 FaceRecognition.py [21] and an example of a detection

However, it is important to notice that since the servers were deployed in JavaScript some type of communication protocol was needed to connect the back and front end within the edge servers and the immediate solution was to call the Python program with a child process. Nevertheless, this solution was lowering the performance of the library resulting in delays of over a second.

This processing delay was unacceptable and after some investigations, the problem was spotted. Since, Dlib is a quite large library, loading it every time a picture is processed would increase the overall delay significantly. To overcome this issue, the solution was to run the Python program independently and call it every time that a face recognition it was needed without the need to restart it.

As it was explained in *Section 3.1*, the real-time communications between the mobile devices and the edge servers are possible thanks to Socket.IO. Since the same kind of bidirectional connection was needed between the Node.js fronted and the Python Face Detector backend, using Socket.IO seemed like the best manner to achieve real-time data transfer between the programs. As it is represented in *Figure 18*, thanks to Socket.IO every time the Node.js frontend receives a new picture over Socket.IO, it redirects the picture to the Face Recognition Python client (also over Socket.IO) , which after doing the detection responds with the amount of faces detected and the name of the picture it detected them from.

Furthermore, by using Socket.IO, the problem of loading the library every time the program is called is solved since the python backend is continuously running from the program's start-up.

Figure 18 Communications within ES between server.js and FaceDetection.PY application

3.2.1.2 *Android client*

On the other hand, as explained *in Section 1.5* of this document, to achieve Face detection in the Android devices the application of Face Detection Android created by Tzutalin was selected [1]. This application is a simple implementation of the C++ Dlib library which is the same one used in the Edge Servers.

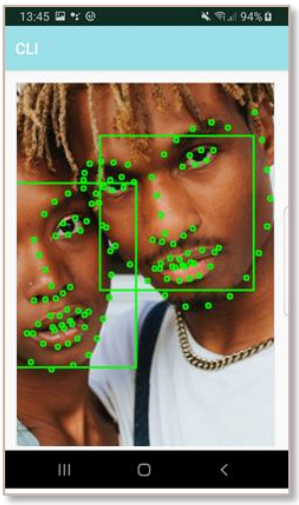

Figure 19 Tzutalin Phone face detection detecting an image with two faces.

This application was combined with the client explained in *Section 3.1* and the face detection software is called in the background every time the offloading decision dictates that the picture needs to be processed internally by the mobile device.

For this project, it was very important to test every part before moving on to the next to avoid any unforeseen consequences of an inefficient implementation. Thus, as it will later be presented in Section 4, experiments were run to understand the processing delay profile.

3.3 PHASE 3. MEC AND FINAL SETUPS

Finally, once communications and computationally-intensive task of face detection were achieved, phase three oversees the deployment of the MEC algorithms and the final setups.

3.3.1 Mobile Edge Computing

In Mobile Edge Computing tasks are released by the phone to be computed by the edge servers. In order to understand the offloading process some things must be considered:

- Independently on from which mobile device a task is released from, it goes throw two phases, the control and the data phase.
- In this experiment, there are 3 devices that can process the tasks: the mobile device itself MD, edge server one, ES1 or edge server two, ES2.
- Two tasks cannot be processed in parallel within a device. Therefore, the waiting time of a task depends on the number of tasks queued in that processing device.

As introduced in Section 1.6 of this document there were some modifications from the initial plan that caused the modification of the final approach to the project. In the end Mobile Edge Computing has been deployed using the following scheduling and offloading algorithms:

- First, as the **scheduling** policy**,** *FIFO* (first in first out) was considered.
- On the other hand, when it comes to **offloading** two low-complexity sub-optimal solutions have been used:
  - *Round robin*. That assigns the task to one of the devices of the network respectively. So, first ES1, second ES2, third on-device processing and then again to ES1, etc.
  - *Random*. Randomly assigns which device process each picture.

[1]https://github.com/tzutalin/dlib-android-app

Finally, it is important to remember that this platform was built from the start of the project to be scalable so that any kind of algorithm could be deployed with little modifications. For some very simplistic algorithms, the control phase may not be necessary, however, bearing in mind this platform has the intention to remain in the laboratory and be used for other future algorithms and even 5G cellular networks. it makes sense, therefore, to build the infrastructure in a scalable manner.

3.3.2 Final set-up and scale up

When it comes to the final set up and the scale up some things were considered.

- Since, Socket.io is present for Android, Node.js, and Python all the elements of the network can be interconnected through sockets.
- The two PC needs to be equipped with the Node.js Server to be able to redirect the images to Python and handle several users (MDs).
- Socket.io is an event-oriented platform therefore for each of the "conversations" an event will be in charge of handling each type of information.
- Each socket has a specific socket ID and each connection between a client and a server will have its specific socket identified by its ID. There are two servers, one deployed on each PC. And 6 clients: the two mobile devices in Android and the three python clients.

As a summary *Figure 20,* presents a schematic of the main message exchange between the Mobile devices' application and the Node.js ES and the Python applications.

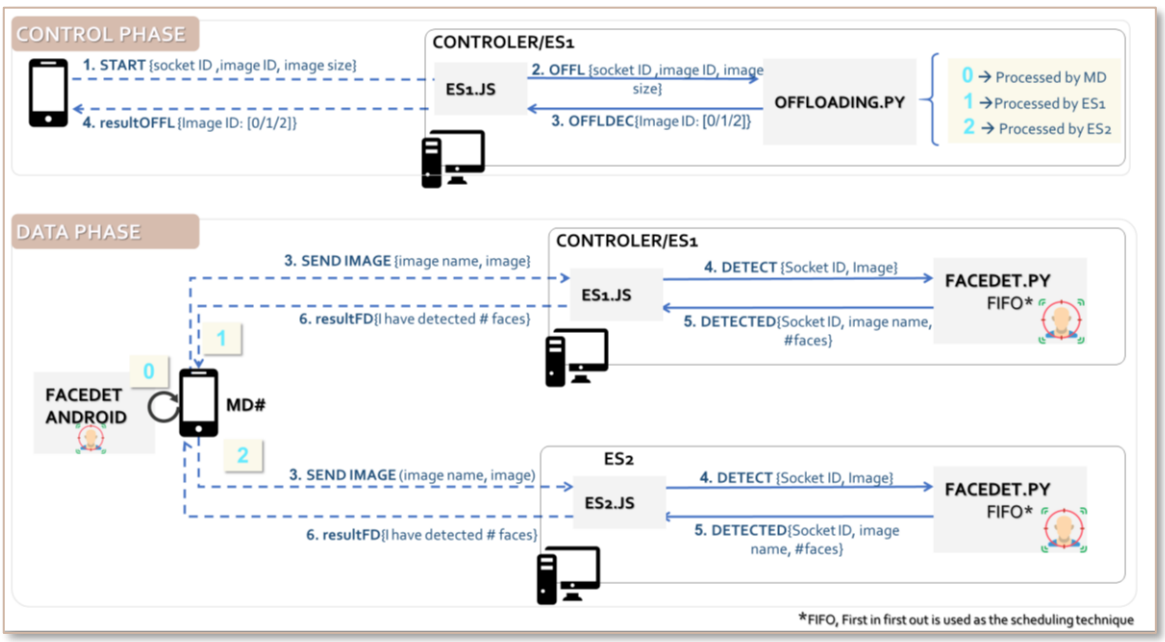

Figure 20 Summary of the application and servers' communications

There are two phases, first the control phase and then the data phase.

- CONTROL PHASE. There are four main events:

    1. ***Start.*** Mobile devices indicate with start the name of the image they want to upload, its size and the socket ID that characterizes the connection between the MD and the CNTR. This socket ID would be useful for later redirect the offloading decision to this mobile device.
    2. ***Offl.*** The server redirects to Python offloading program the information it needs to make the offloading decision.

3. ***OffIDec.*** Depending on the offloading algorithm, the offloading decision will be made in different ways. However, this event will always respond with the name of the image and with:
    - **0:** If the decision is that the task must be processed by the MD.
    - **1:** When it must be processed by ES1.
    - **2:** If, on the other hand, ES2 has to process it.

    Again, it redirects the socket ID for the server to know who sent the request.
4. ***ResultOff.*** through this event, the offloading decision is redirected to the MD.

Figure 21 represents Example 1. The mobile device [1] sends the control information which is received by the controller (also hosting ES1). The JavaScript front end (server.js) is in charge of resending this information to the back end[2]. The python offloading program (MECRR.py) makes the offloading decision [3] and then sends back this information to the front end that redirects the information again back to the mobile devices. [4]

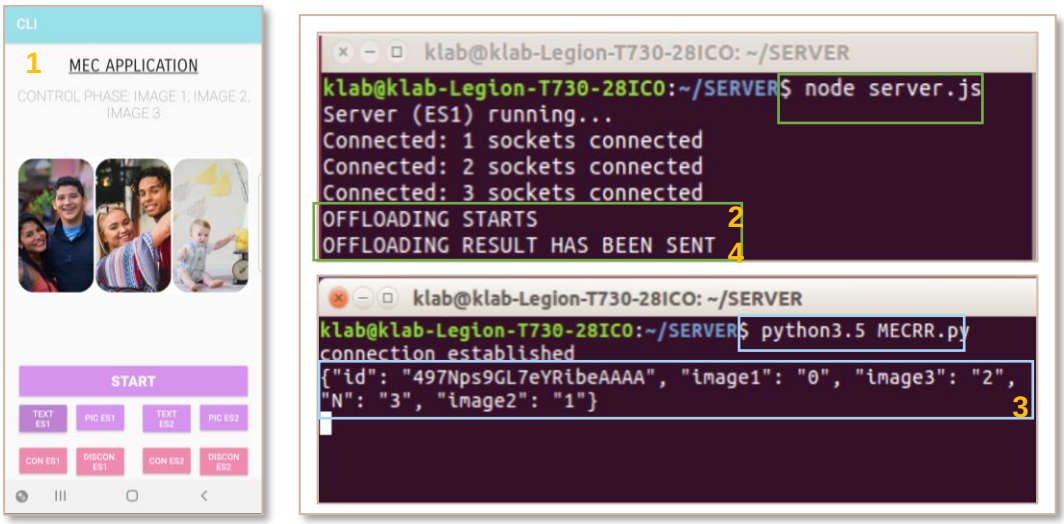

Figure 21 Example 1. Simple demonstration with 3 pictures: Control phase.

- DATA PHASE. The data in this phase can go through three different actions according to the decision dictated by the controller. As stated, the controller indicates with a 0,1 or 2 whether the picture needs to be processed internally, by edge server one or two respectively.

  Whenever the decision is to offload whether it is to ES1 and ES2, four events dictate the message exchange to share the information:

  5. ***SendImage.*** According to the offloading decision, MD sends the image to the corresponding server (1 or 2) through this event.
  6. ***Detect.*** Mobile devices redirect the picture and the Socket ID for the Python program to detect the faces.
  7. ***Detected.*** Once the detection is completed the Python program sends back the name of the image and the number of faces in each picture.
  8. ***ResultFD,*** the face detection result is redirected to the MD.

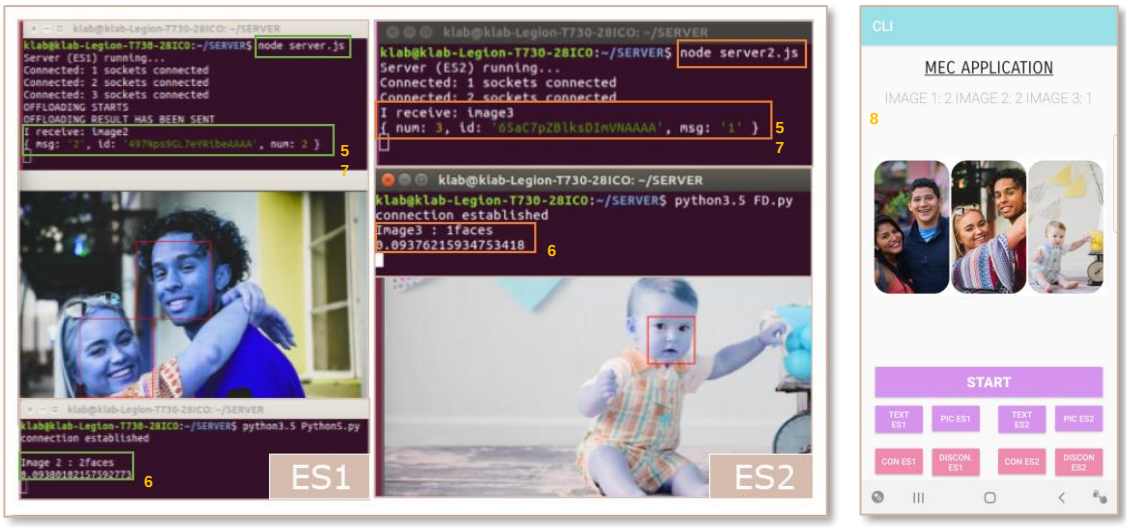

Figure 22 Example 1. Simple demonstration with 3 pictures: Data phase.

Continuing with the example represented in *Figure 21* in which the mobile device wanted to offload 3 images, *Figure 22* represents the data phase. According to the offloading decision MD1 processes image 1 internally and sends the picture 2 to ES1 and picture 3 to ES2. [5] The python backends in both computers process it [6] and finally, the result is redirected to the MD [7] that shows the result on screen[8].

4 RESULTS

Focusing now on the project's outcome, Chapter 4 of this thesis centers its attention on carrying out experiments and retrieving results. For this section, it is essential to have in mind the motivations of adopting Mobile Edge Computing.

Particularly, for computational-intensive battery hungry applications, the main aspects for which MEC could be especially beneficial are task processing time and mobile device battery consumption. This experiment focuses on evaluating these two main concerns that the face detection application has to overcome: latency and battery consumption.

As it has been stressed during chapter three, this project's development was divided into three phases, each of which, had to be tested prior to continuing with the next. It is very important to understand, that each of these tests were not only to ensure the correct flow of the project and to avoid unexpected deviations but also, the former serves to supports the understanding of the latter. Consequently, chapter 4 is divided into three sections:

1. **Data transmission performance**. This section focuses on the testing of the infrastructure built in phase one of the project development. The goal of these measurements is to estimate the communication delay linked to each data size.

2. **Face detection library performance**. Within phase two of the development of the project, the work focused on achieving face detection. Before moving on to the implementation of MEC it was very important to understand what was the processing delay and the main factors that would increase this value.

3. **MEC performance.** Since the ultimate goal of this project is to measure the performance of MEC in 4G systems, section 4.3 is dedicated to measuring the average total task time for the different algorithms and evaluating battery consumption.

4.1 NET COMMUNICATIONS DELAY

The system-level delay performance of our interests is composed of communication and computation aspects. In this stage, the testing was focused on the former one by measuring the round-trip time (RTT) of request-response (REQ-RESP) message exchanges in the Control and Data plane. To do this, an MD periodically sent a REQ message to CTRL, and on receiving the REQ message, CTRL can promptly respond to the MD via a RESP message. These tests consist in comparing the performance of using the SINET-LTE network to a small local WIFI network.

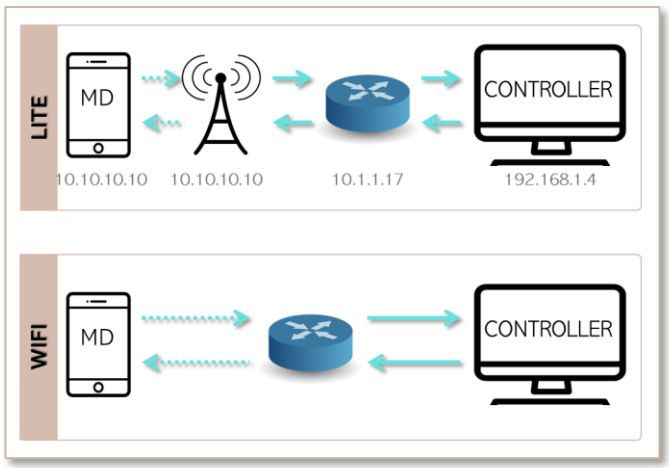

Figure 23 LTE and WIFI network configurations. In the WIFI network, DHCP IP assignment has been used.

4.1.1 TEXT DATA TRANSMISSION

The first network's performance test consisted of sending a small text message of 132 bytes over the SINET-LTE network continuously every 100 ms. The two main insights to be taken from this test were to understand the time a short message would take to be sent from the client, resent from the server and received by the client.

```
{"ID": "1",          //PHONE ID
 "N": "2",           //NUMBER OF IMAGES
 "size": "50,100",   //SIZE OF THE IMAGES
 "RTT": "50ms" }     //FROM HISTORICAL DATA
```

Figure 24 Testing message that resembles what the Control Message would look like.

By looking at the networks' performance in *Figure 25*, it can be observed how the LTE network presents sudden peaks during some time periods. It is important to remember, that even though, LTE communications aim to be very fast and reliable networks, the signal transmitted from MD to the base station has to travel a long distance, sometimes under congested or high fading situations and these peaks are normal. Moreover, since the ESs are located within the SINET network instead of the edge (eNB), an extra delay in the average RTT was expected.

On the other hand, it can be noted how WIFI communications are much more stable over time and have in lower average Round Trip Time, this result was expected due to the proximity of the mobile device to the router, the low congestion of the Access Point and the deployment of the Edge Servers in the Edge of the network.

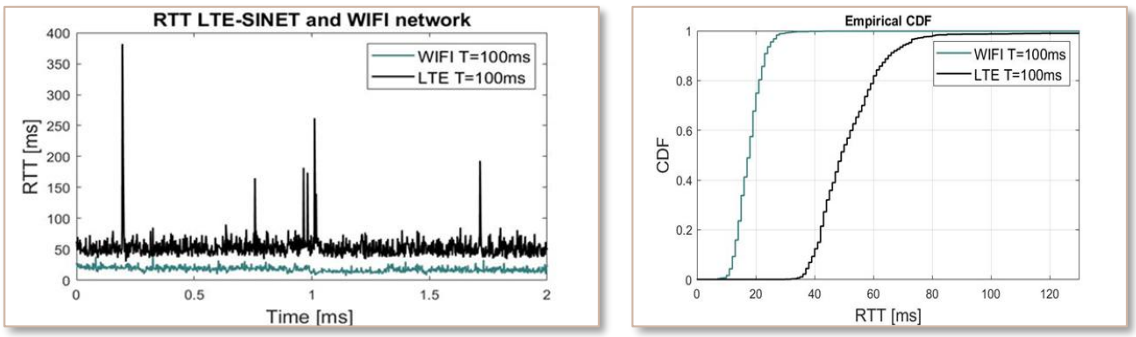

Figure 25 Small text performance LTE vs WIFI.

Also, with the cumulative distribution function (CDF), that calculates the cumulative probability for a given x-value it is easy to understand the differences in performance between both networks. The WIFI network RTT is lower or equal than 20 ms for 60 percent of the samples taken. Whereas in the LTE network the same number of samples have a value of lower or equal than 50 ms.

- **Observation 1:** The WIFI networks presents lower RTT and more stable connectivity. Since we can deploy the server closer to the access point, the WIFI link can be used as a lower bound of how the MEC system could perform in an ideal case scenario.

- **Observation 2:** The overall performance of the 4G-based SINET-WADCI platform is good since it can mostly provide mean RTT below 100 ms, despite sometimes larger RTT.

4.1.2 LARGER DATA, IMAGE TRANSMISSION

Secondly, in order to have more insights on how the data plane's transmission delay profile would behave and whether or not the built infrastructure during phase one was sufficient to support big data file transferring, the second set of tests measured the performance of sending round trip (client to server and server to client) three different-dimension images. These tests were performed for both WIFI and SINET-LTE network infrastructures.

From *Figure 26* it is easy to see how the round-trip delay has a maximum of 500 ms Round Trip Delay in LTE and 300 ms for WIFI, even for the large image. In the case of the medium image, the RTT is around 300 ms in LTE and 200 ms for WIFI. Finally, for the smaller image, the RTT has a minimum, being around 200ms for LTE and 50 ms for WIFI.

It is important to notice that both networks have the same behaviors as in the test presented in *Section 4.1.1*, in which LTE presents sudden peaks in the RTT, with the only difference that the average round trip time increases for both of them for the obvious reason that the amount of data transmitted is larger.

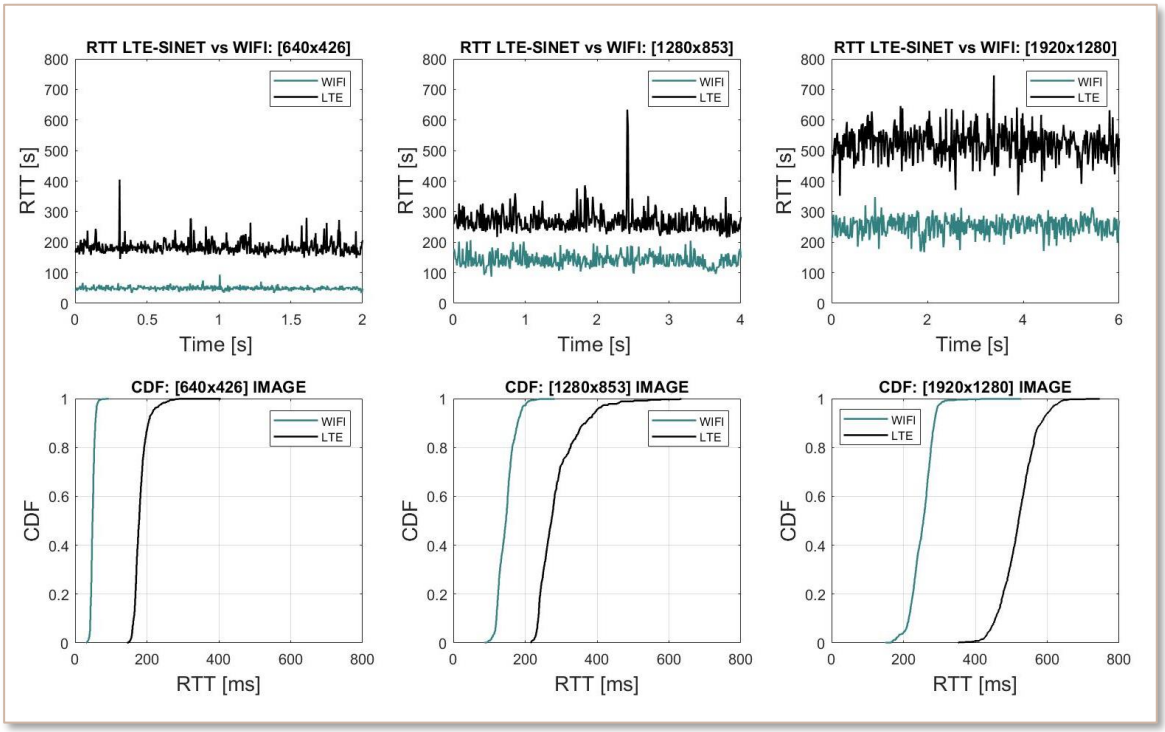

Figure 26Performance of the network sending round trip [640x426], [1280x853] and [1920x1280] images.

- **Observation 3:** The 4G-based platform can mostly provide mean RTT :

  - Below 600 ms for large images (300ms for the WIFI case).
  - Below 300 ms for medium-size ones (200 ms for WIFI).
  - As low as 200 ms for small dimension pictures (below 50ms for WIFI).

  RTT delay in WIFI is approximately half than LTE.

Just a side note, it is important to remember that in the case of this thesis face detection application, the delays related to the image transmission would be lower (around half of this value) due to the fact that the image only needs to be sent from client to server, and the server replies accordingly. These results lead to the next phase of the project which is the Data Plane implementation.

4.2 COMPUTER VISION LIBRARY PERFORMANCE: PROCESSING DELAY

The initial studies had proven that the platform's performance was suitable or the experiment, this section moves on to perform some experiments to understand the overall performance of the implemented library. Hence, the following tests help to understand what the maximum and minimum picture processing times are, and what are the factors influencing these values.

4.2.1. FACTORS INFLUENCING THE PROCESSING TIME

The following tests were made with 100 picture data set for each image size, i.e. a total of 300 pictures from 1 to 14 faces within the pictures taken from Pexels Website [1].

Two aspects were considered as the two main potential factors increasing processing time: face count and image size.

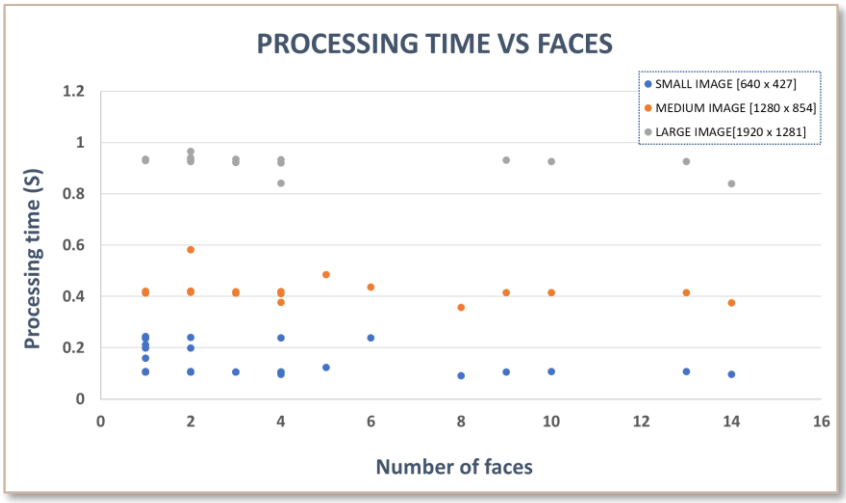

Figure 27 Processing time vs the number of faces.

By plotting the processing time vs, the number of faces (*Figure 27*) it can be seen that there is no direct correlation between a greater amount of people in a picture and greater processing time.

This graph shows three different groups of image dimensions classified by color. Within these pictures, there are from one to fourteen people, however, comparing the processing time of the picture with fourteen people to the one with only one the processing time is equal.

Figure 28 Processing time vs Image dimensions

According to *Figure 27,* it could already be seen that the image dimension actually plays a big role in the picture's processing time. Thus, it made sense to study the average processing time per image size and its standard deviation (*Figure 28*). Thanks to both *Fig.27* and *Fig.28*, it is very easy to see how all the images with the same size are processed in almost the same amount of time.

Thus, from these tests two observations can be extracted:

- **Observation 1:** The greater the size of a picture, the larger the processing time.

- **Observation 2:** There is no correlation between a greater amount of people in a picture and a larger processing time.

Besides, these results can be combined with the ones extracted in the communication delay experiment in which it was proven how dimensions of the picture have an impact on the data transmission delay. Even though it is beyond the scope of this project, picture size could be used as an insightful input in an optimization offloading algorithm, that dictates which pictures are offloaded to the network, to minimize the total task processing time.

4.2.2. PROCESSING TIME: MOBILE DEVICE VS SERVER

Finally, the last performed before moving on to the last phase of the project was measuring the difference in performance between the servers and mobile devices when performing face detection. This graph shows the difference in processing time between the mobile devices and the edge servers for the data set mentioned before.

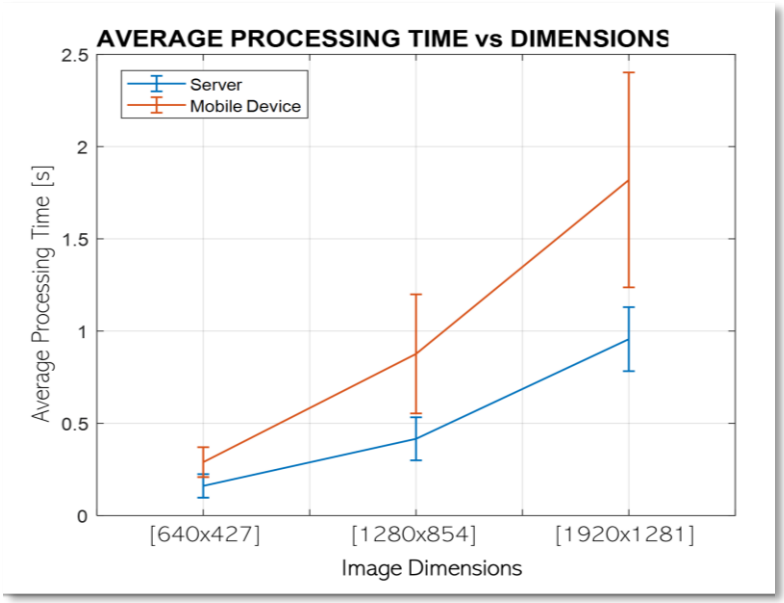

Figure 29 Processing time vs Image dimensions

Figure 29 demonstrates that the edge servers have a better processing performance that the mobile device. ES performs the detection in less at most one second for the worst case, whereas the mobile device's delay can reach 2.5 seconds.

- **Observation 3:** The overall performance of the Edge Servers is better. This difference in performance is especially important in medium-size and large images.

This simple observation is in fact one of the most important findings of the project since it shows the motivation on why Edge Computing could be a good solution for this application, here we can see that actually the delays in the mobile phone could be reduced almost in half if the condition of the communication link is good.

[1] https://www.pexels.com/

4.3 MEC PERFORMANCE

One of the main concerns of computationally intensive applications is that in order to perform each task and due to the limitation in phone's processors, there is usually a considerable delay linked to every individual task. Just as it was introduced in *Chapter 2,* humans are very susceptible to glitter and reducing this delay would broaden the horizons on the kind of applications mobile devices can be equipped with. On the other hand, this kind of applications imply a rather high battery consumption.

These two aspects are two of the main motivations behind using MEC and this section is going to focus on testing the performance of latency and battery life in a real MEC-LTE case scenario. For that, the experiments conducted in this part aim to measure the improvements Round Robin and random algorithms could bring just by partially offloading tasks to the network.

4.3.1 Latency: Average Total Task Time

The Total Task Time (TTT) is the total time, i.e. communication plus processing time, that takes a task to be finalized. The test has been performed in a dataset of 100 images that is sent repeatedly every 20 milliseconds and therefore the average TTT means what is the total time that takes in average for each task to obtain the result of how many faces it contains, whether it has been processed internally or offloaded.

As it has been stressed the goal of this project was to test the advantages and disadvantages of using simple algorithms and edge computing itself by comparing the following cases:

- Everything is processed by the mobile devices themselves.
- Round Robin.
- Random.

Moreover, similarly to *Section 4.1,* when testing the communication delay these experiments have been conducted for WIFI and LTE, considering that in the case of WIFI the server is located in the real edge, its performance appears like a good reference value of what could potentially be attained when locating the servers in the core of the base station.

4.3.1.1 Algorithm comparison

This first test focuses on testing the differences between the deployed algorithms: Round Robin and random.

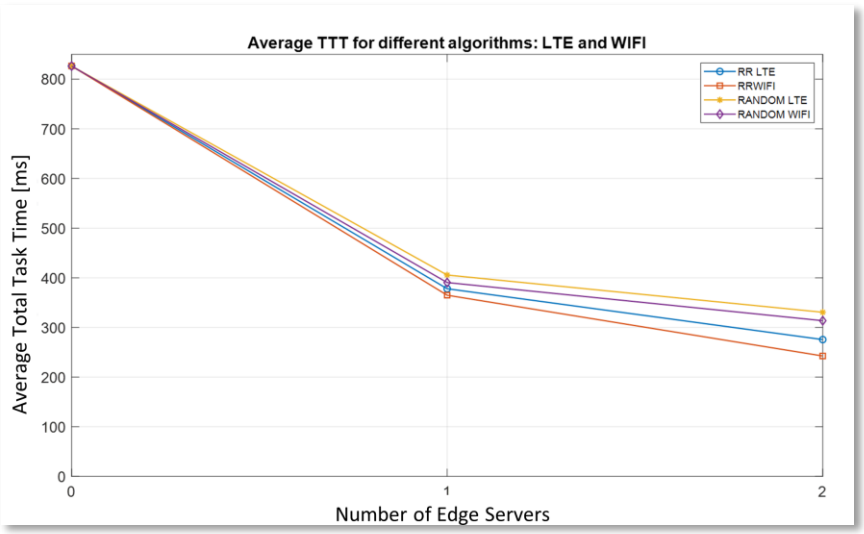

Figure 30 Average Total Task Time for Round Robin and Random algorithms for both LTE and WIFI communications.

The first thing to look at when seeing these results is the huge improvement using two edge servers has on the average Total Task Time. By looking at the case when Edge Computing is not used, i.e. zero edge servers are used and the phone processes all internally, the average TTT is greater than 800ms. On the contrary, it can be seen how for the best-case scenario (Round Robin- WIFI), this value goes all the way down to 242.5 milliseconds, which is a reduction in more than half of the average TTT just by using a simple algorithm and two edge servers.

It is very important to understand that the primary goal of the project was to have evidence of the benefits of MEC for cellular LTE networks. With this simple test we have proven the advantages Mobile Edge Computing could bring for latency concern applications.

Moreover, as can be seen, WIFI outperforms LTE in both cases, the reason behind this appears to be because of the communication delay studied in *Section 1.1*. However, further sections are specifically dedicated to this study.

Finally, it can be observed that Round Robin has a better overall performance than the random algorithm because it evenly distributes the tasks between the devices. Therefore, from these section three main observations can be highlighted:

- **Observation 1:** By using MEC the average Total Task Time [ms] can be reduced more than to half in the best of the cases.

- **Observation 2:** Round Robin has a better performance than the Random Algorithm because of the evenness in its task-sharing assignment.

- **Observation 3:** WIFI presents slightly best performance than the LTE network due to lower transmission delays.

In order to get more insights, the same data is being plotted in bar form including the error bar. This is very useful since it helps us see how diverse the data.

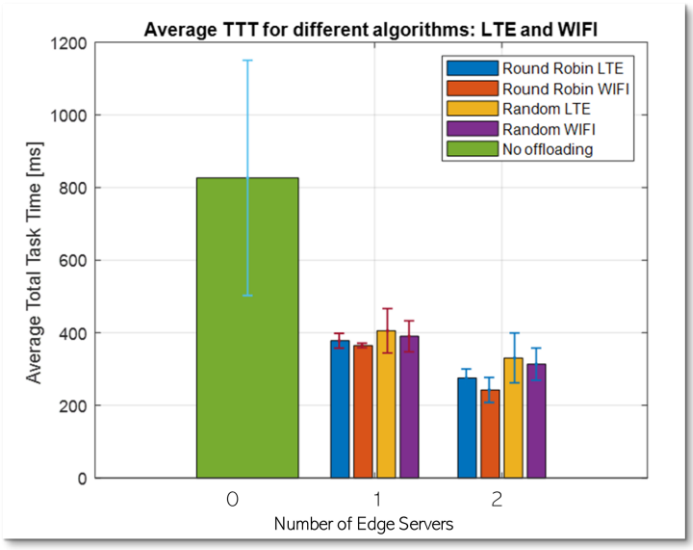

Figure 31 Average TTT and standard deviation for RR and Random algorithms for Average Total for both LTE and WIFI communications.

- **Observation 4:** Round Robin with WIFI presents a very stable performance because of both the stability of WIFI communication and the non-variability of the Round Robin offloading decision outcome.

- **Observation 5:** Both LTE, because of its transmission characteristics and the Random Algorithm because of its own nature introduce randomness to the Total Task Time.

4.3.1.2 Average Total Task Time and Processing Time

From the previous test, it was still unclear on what are the reasons for an increment in the average Total Task Time. Since Round Robin is been proved to be the most stable algorithm, this test measures how much of the total task time is due to the processing time.

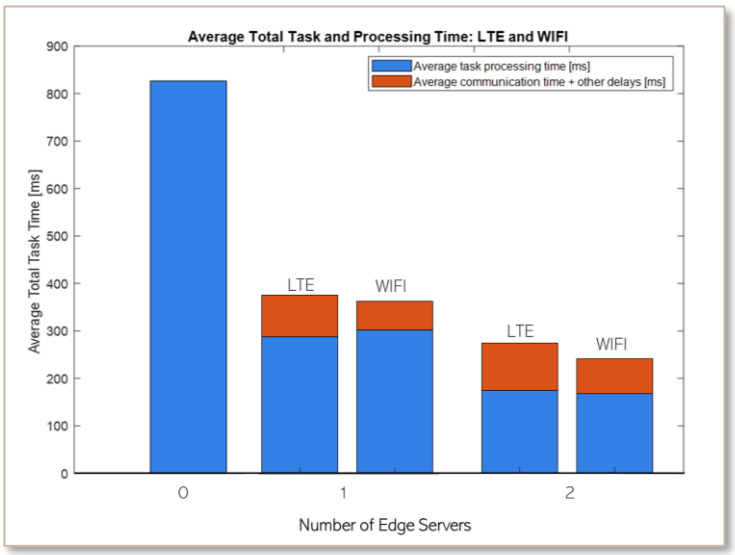

Figure 32 Average processing time with respect to average TTT [ms]

This figure shows the average total task time composed by the average processing time and some additional delays. These other delays are in its great majority due to communication delays however, some other factors not considered could also take part in this extra delay. As it was expected the majority of the delays are caused by the processing time.

Again, it can be observed how the two edge servers outperform, one and using MEC supposes a great advantage over on-device processing. Furthermore, some other conclusions can be drawn:

- **Observation 1:** Processing time is quite stable but in case of slower performance, it increases the TTT.

Processing time is a variable value, even though it is quite constant over time it can sometimes experiment a larger delay. This, therefore, is translated in larger average processing time, which affects the total task time. This is reflected more significantly for the case of one edge servers, in which the difference between WIFI and LTE has been reduced because of this increase in the processing delay, while the proportion of the communication delay remains the same from one to two servers.

- **Observation 2:** The percentage of TTT required for transmission is higher for two edge servers, but the overall result is the best for this case.

For obvious reasons when using Round Robin, the communication delay is higher for the case of two edge servers. Since the tasks are evenly distributed two every three tasks would be offloaded versus one every two in the case of one edge server, which in the end increases the communication delay.

Moreover, the transmission delay for the WIFI case is half of the transmission delay for LTE. This result goes in accordance with the conclusions extracted at the early stages of the project when testing the communication performance.

4.3.2 Battery consumption

Finally, the last parameter evaluated in the network was the battery consumption. It is important to remember that one of the biggest motives to use edge computing is the potential reduction in battery consumption for battery-hungry-computationally intensive applications. Since the mobile devices used for the experiment have the same characteristics and were purchased at the same time, it is quite easy to perform an accurate consumption test on what would be the difference between Round Robin with two Edge Servers and on-device-only processing with no MEC.

For this test the app "Simple Battery Graph" that measures the battery consumption over time and permits to export the data to a computer to analyze the data.

Therefore, the test *environment:*

- Two phones Samsung S9.
- Same characteristics
- Starting from 100% charge.
- Not connected to WIFI, but both connected to LTE (only one of them offloads the tasks but both have the data activated).
- Same data set: 100 different medium-size pictures, sent at repeatedly over the 3 hours and 15 minutes.

Moreover, the test was performed simultaneously, and the image was being processed continuously, the moment one is terminated (whether by the phone or the response is received from the edge server), the next image is sent or starts processing.

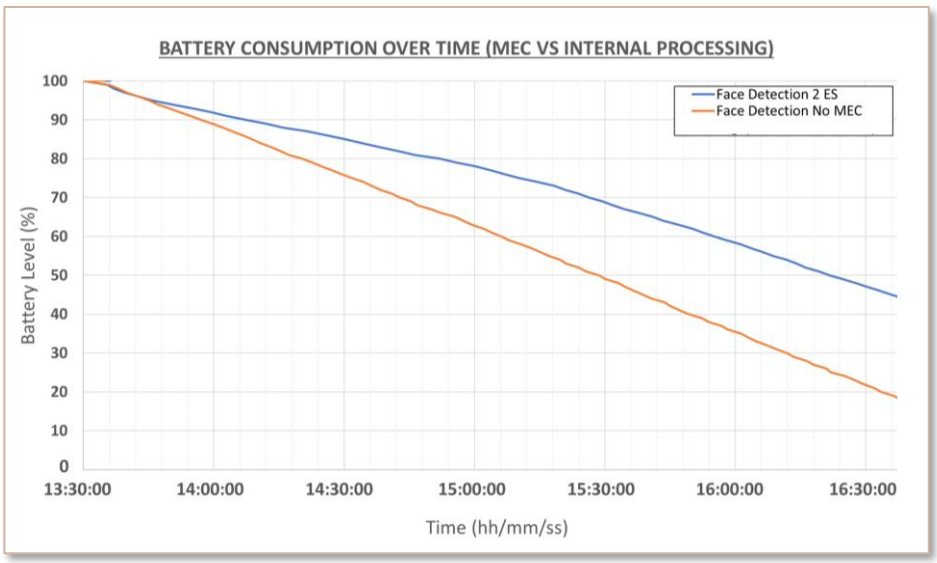

Figure 33 Time consumption over time: LTE based MEC vs Internal Processing.

For a three hour and fifteen-minutes test the phones with the same characteristics ended up with 16% for the internal processing vs 43% for the one using 4G based MEC with two servers.

- **Observation 1:** Battery consumption can be reduced by using Mobile Edge Computing in computationally intensive-battery hungry applications.

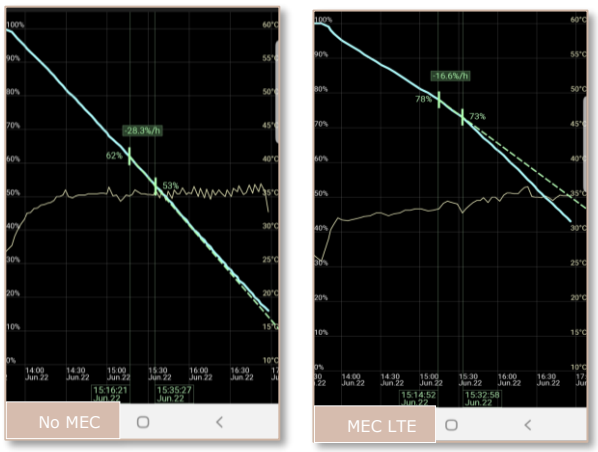

Figure 34 Screenshot of Simple Battery Graph for phone battery consumption (test duration: 3 hours and 15 minutes).

Moreover, this test was repeated under the same conditions for comparing MEC for LTE and WIFI. After three hours and fifteen minutes, the battery of the phone offloading over WIFI to two Edge Servers was 45%, which is slightly higher than for LTE that resulted this time in 38%.

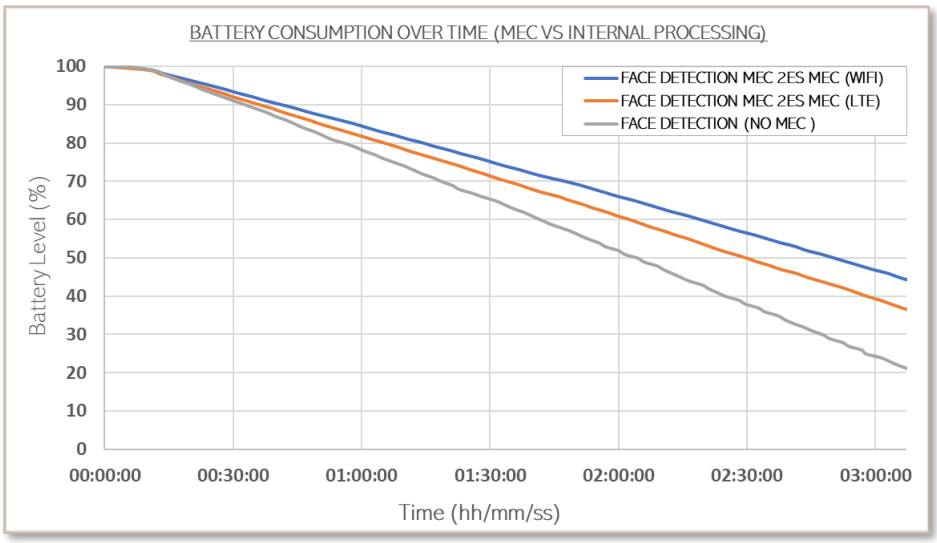

Figure 35 Time consumption over time: LTE based MEC, WIFI based MEC and Internal Processing.

Therefore, an observation can be extracted:

- **Observation 2:** Battery consumption has a better performance using Mobile Edge Computing on WIFI than in LTE but this difference in consumption it is no comparable with the differences between using or not MEC.

This higher consumption in LTE is the same that us as mobile device consumers experience every day. It is very common as consumers to turn off the data to reduce battery consumption. This outperformance in the case of WIFI is due to the high battery consumption that implies the connection to different base stations and the long-distance data transmission in LTE.

To sum up, this section has put its attention on testing, both the testing environment and the technology. First, by testing the communication delay, it has been understood what the maximum and minimum delays the application could suffer due to the task transmission. Later

on, this study has evaluated the performance of the deployed library to have a better understanding of what could be the potential benefits of using MEC.

Finally, the last section of *Chapter 4* has focused on the technology itself. Mobile Edge Computing, has been proved to be the promising technology that is said to be, reducing latency and improving battery performance in mobile devices hosting computationally intensive battery hungry applications.

5 BUDGET

Since NII is a public administration and there was no government subvention particularly specific for my project there is no purchase, besides my salary, that was specifically made for the project, but I used the components that were already available, however, an approximate estimation of their value can easily be done. Therefore:

1. **Prototype component budget**
   - 2 Samsung S9 with an approximate cost of ¥ 53,800 x 2 = ¥ 107600
   - 2 PC Lenovo Legion T730: ¥ 264,708 x 2= ¥ 529416
   - Router Airport by Apple: ¥ 18,000

2. **Engineering budget:** On the other hand, the designing and prototyping cost or personnel cost was of 151 days x ¥ 5,700 = ¥ 860700 (approximately 37000 euros).

Besides, there could be some additional costs related to the collaboration between SiNet, NII and KLab however, that information was confidential.

6 CONCLUSIONS AND FUTURE DEVELOPMENT

This thesis has centered its attention on the development of a task processing LTE based mobile edge computing platform. The goal of the experiment was to evaluate the performance of 4G-based MEC for task offloading in computationally intensive applications. Both the development of the project and the testing of the platform have been done progressively over time in order to understand the limitations and to avoid possible cumulative errors.

In phase one, the initial set up was done by building a 4G-based system where multiple Mobile Devices and Edge Servers could coexist. Moreover, in this phase, some preliminary tests were performed to understand the uplink and downlink delays that different sizes of data sets would experience. It was concluded that the overall performance of the 4G-based SINET-WADCI platform is good since it can mostly provide mean RTT below 100 ms for text messages and below 600 ms for any image size, despite sometimes larger RTT.

Besides, the performance of the 4G-based SINET-WADCI platform was compared to WIFI which resulted in a reduction in half of the RTT delay. Since we can deploy the server closer to the access point, the WIFI link is been used the whole project as a reference lower bound of how the MEC system could perform when locating the servers closer to the base station. Finally, in phase one, the request and response messages for the signally in the control plane were implemented.

Next, phase two consisted of the deployment of the data plane, in which the mobile devices following the instructions of given in the control plane would or would not offload the pictures to be detected to the network. For that, it was necessary to achieve face detection in all the devices of the network, which was possible thanks to Dlib. In the same manner, as in phase one, this library was submitted to an evaluation in order to understand its performance.

It was observed how the main factor increasing the processing time was image dimensions. Furthermore, this phase also included the comparison in performance between processing images internally by the MD versus when these were processed in the ES. *Figure 29* demonstrated that the edge servers have a better processing performance that the mobile device. Outperforming mobile devices, the edge servers present processing delays lower than one second even for the largest images, whereas the mobile device's delay can reach 2.5 seconds. This difference in performance was the ultimate motivation to continue with the project.

Finally, the last step of the project included integrating all the components of the experiment and the deployment of Mobile Edge Computing offloading algorithms. Round Robin and random algorithms were deployed in the network dictating how to offload the task. Both of these algorithms were evaluated through multiple tests that compared its performance with the case where no MEC is possible, i.e. everything is processed internally on the device.

After various latency and battery tests, MEC for LTE has been proven to give mobile phones higher computational power, with a reduction in battery consumption and without the need for any extra hardware within the devices, since all smartphones nowadays are equipped with LTE capabilities. Thus, two main aspects have been proven to be achievable:

- **Low latency.** By approaching the servers closer to the end-user, it results in a huge enhancement regarding latency. By using MEC the average Total Task Time [ms] can be reduced more than to half in the best of the cases being going from 843 ms when is processed internally to less than 300ms for round robin algorithm.

Moreover, because of the evenness in its task-sharing assignment Round Robin has a better performance than the Random Algorithm. Also, WIFI presents slightly best performance than the LTE network due to lower transmission delays.

But most importantly MEC always outperforms internal processing, which will allow for new latency concerned applications such as the tested face detection, virtual reality or even other still to be discovered applications.

- **Mobile device energy savings.** By transferring computation to the network mobile devices significantly reduce battery consumption when using these computation-intensive applications.

  Even though the battery consumption in the LTE based MEC network is a slightly higher than in the WIFI based. MEC reduces in more than half the battery consumption.

Ultimately, the experiment has achieved it is the goal of not only testing but proving the efficiency of LTE based MEC systems for the battery hungry latency concerned application of Face Detection.

Notwithstanding, several future milestones have been left for the future. The immediate next step to be taken is to test optimization algorithms on the platform and in the 5G network that will be available from September. This would enable to see how MEC optimization algorithms could increase the overall performance of MEC systems and measure the range of improvement they could bring. Aside, it would be compelling to measure the performance of MEC in dense networks, test MEC under mobility and test the performance of Mobile Edge Computing in heterogeneous networks.

Bibliography

[1]. F. Sabahi, "Cloud computing reliability, availability and serviceability (ras): Issues and challenges," International Journal on Advances in ICT for Emerging Regions (ICTer), vol. 4, no. 2, 2012

[2]. European Telecommunications Standards Institute, "Mobile-Edge Computing - Introductory Technical White Paper." Sept. 2014.Available at: https://portal.etsi.org/Portals/0/TBpages/MEC/Docs/Mobile-edge_Computing_-_Introductory_Technical_White_Paper_V1%2018-09-14.pdf. [Accessed 1 Jul. 2019].

[3]. CISCO, "Edge computing vs. fog computing: Definitions and enterprise uses". Available at: https://www.cisco.com/c/en/us/solutions/enterprise-networks/edge-computing.html. . [Accessed 1 Jul. 2019].

[4]. European Telecommunications Standards Institute, "Mobile-Edge Computing (MEC); Service Scenarios "Available at: https://www.etsi.org/deliver/etsi_gs/MEC-IEG/001_099/004/01.01.01_60/gs_MEC-IEG004v010101p.pdf. Last seen 1/07/2019.

[5]. T. X. Tran, A. Hajisami, P. Pandey and D. Pompili, "Collaborative Mobile Edge Computing in 5G Networks: New Paradigms, Scenarios, and Challenges," in IEEE Communications Magazine, vol. 55, no. 4, pp. 54-61, April 2017.

[6]. European Telecommunications Standards Institute, "MEC Deployments in 4G and Evolution Towards 5G" Available at: https://www.etsi.org/images/files/etsiwhitepapers/etsi_wp24_mec_deployment_in_4g_5g_final.pdf. [Accessed 1 Jul. 2019].

[7]. C.-Y. Chang, K. Alexandris, N. Nikaein, K. Katsalis, T. SpyropoulosMec architectural implications for lte/lte-a networks. Proceedings of the Workshop on Mobility in the Evolving Internet Architecture, ACM (2016).

[8]. Mao, Yuyi & You, Changsheng & Zhang, Jun & Huang, Kaibin & B. Letaief, Khaled. (2017). Mobile Edge Computing: Survey and Research Outlook.

[9]. A. van Kempen, T. Crivat, B. Trubert, D. Roy and G. Pierre, "MEC-ConPaaS: An Experimental Single-Board Based Mobile Edge Cloud," 2017 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud), San Francisco, CA, 2017, pp. 17-24.

[10]. S. Yang, Y. Tseng, C. Huang and W. Lin, "Multi-Access Edge Computing Enhanced Video Streaming: Proof-of-Concept Implementation and Prediction/QoE Models," in IEEE Transactions on Vehicular Technology, vol. 68, no. 2, pp. 1888-1902, Feb. 2019. doi: 10.1109/TVT.2018.2889196

[11]. N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, CA, USA, 2005, pp. 886-893 vol. 1.

[12]. Anon, (2019). A full guide to face detection. [online] Available at: https://maelfabien.github.io/tutorials/face-detection/# [Accessed 1 Jul. 2019].

[13]. Consulting, A. (2019). Image Recognition and Object Detection : Part 1 | Learn OpenCV. [online] Learnopencv.com. Available at: https://www.learnopencv.com/image-recognition-and-object-detection-part1/ [Accessed 1 Jul. 2019].

[14]. Consulting, A. (2019). Histogram of Oriented Gradients | Learn OpenCV. [online] Learnopencv.com. Available at: https://www.learnopencv.com/histogram-of-oriented-gradients/ [Accessed 1 Jul. 2019].

[15]. Cs.brown.edu. (2019). Face detection with a sliding window. [online] Available at: http://cs.brown.edu/courses/csci1430/2011/results/proj4/hangsu/ [Accessed 1 Jul. 2019].

[16]. P. F. Felzenszwalb, R. B. Girshick, D. McAllester and D. Ramanan, "Object Detection with Discriminatively Trained Part-Based Models," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 32, no. 9, pp. 1627-1645, Sept. 2010.

[17]. Nebra, M. (2019). Socket.io: let's go to real time!. [online] OpenClassrooms. Available at: https://openclassrooms.com/en/courses/2504541-ultra-fast-applications-using-node-js/2505653-socket-io-let-s-go-to-real-time [Accessed 1 Jul. 2019].

[18]. Socket.IO. (2019). Socket.IO — Native Socket.IO and Android. [online] Available at: https://socket.io/blog/native-socket-io-and-android/ [Accessed 1 Jul. 2019].

[19]. Consulting, A. (2019). Face Detection - OpenCV, Dlib and Deep Learning | Learn OpenCV. [online] Learnopencv.com. Available at: https://www.learnopencv.com/face-detection-opencv-dlib-and-deep-learning-c-python/ [Accessed 1 Jul. 2019].

[20]. Dlib.net. (2019). [online] Available at: http://dlib.net/face_detector.py.html [Accessed 1 Jul. 2019].

[21]. Y. Mao, C. You, J. Zhang, K. Huang and K. B. Letaief, "A Survey on Mobile Edge Computing: The Communication Perspective," in IEEE Communications Surveys & Tutorials, vol. 19, no. 4, pp. 2322-2358, Fourthquarter 2017.