

Crowdsourcing Photos in Edge-Clouds with Panoptic

Tadeu Augusto Leite Freitas

Mestrado Integrado em Engenharia de Redes e Sistemas Informáticos

Departamento de Ciência de Computadores

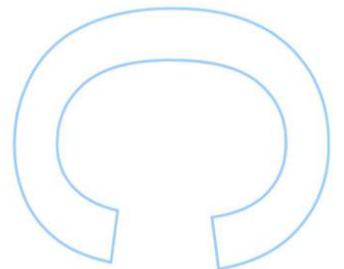
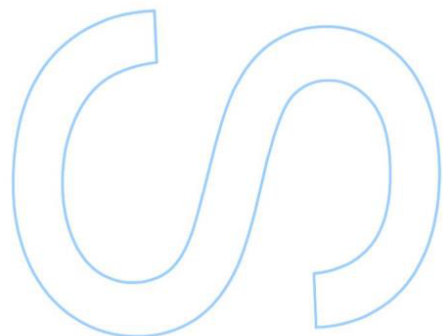
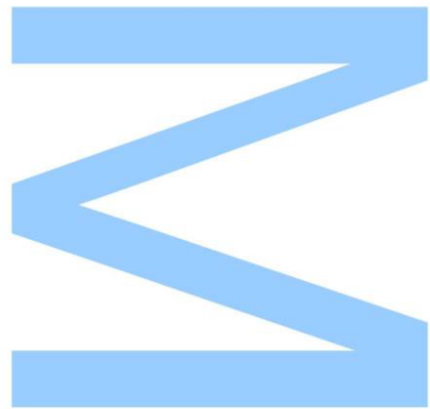
2017

Orientador

Rolando Martins, Professor Auxiliar Convidado,
Faculdade de Ciências da Universidade do Porto

Co-Orientador

Miguel Coimbra, Professor Auxiliar,
Faculdade de Ciências da Universidade do Porto



U. PORTO

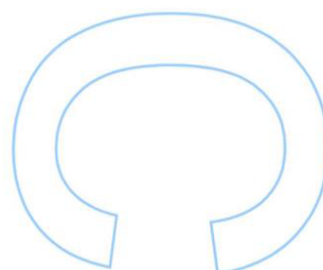
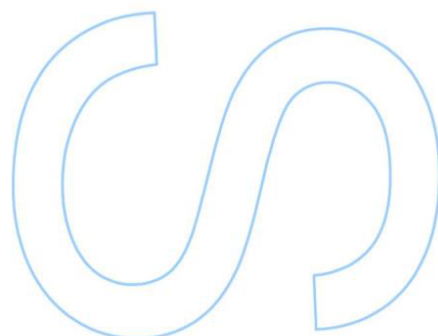
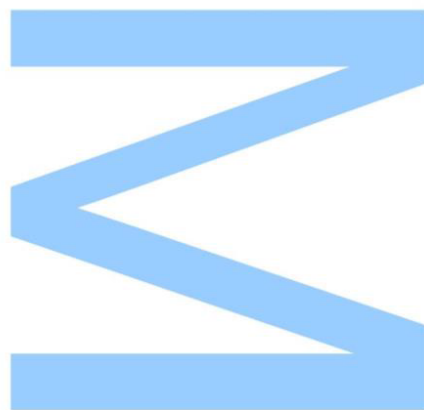


FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Todas as correções determinadas
pelo júri, e só essas, foram
efetuadas.

O Presidente do Júri,

Porto, ____/____/____



To my brother...

“Home is behind, the world ahead,
And there are many paths to tread
Through shadows to the edge of night,
Until the stars are all alight.”

J.R.R. Tolkien

Acknowledgements

First, I would like to thank my family for the support they gave me, specially in my lowest moments, and for their counsel. With a very special gratitude to my brother that has always been at my side.

I would also like to appreciate my supervisors, Professor Rolando Martins and Professor Miguel Coimbra, for their advice and for their patience. They were always present, guiding me with a helping hand.

I am grateful to Nuno Silva and Diogo Machado, always challenging me to do better and always aided me when I needed the most.

I would like to thank the research grant provided by the HYRAX project (CMUP-ERI/FIA/0048/2013) at CRACS/INESC-TEC.

A special mention to João Rodrigues and Sérgio Faria. Their support was essential for my work to be completed.

And finally, last but by no means least, I want to thank Ana Amaro Ferreira, for her support, for her belief in me and for her capacity to see the best of me, until the end.

Thanks for all your encouragement and support!

Abstract

The increasing capabilities of smartphones is paving way to novel applications through the crowd-sourcing of these untapped resources, to form hyperlocal meshes commonly known as edge-clouds. While a relevant body-of-work is already available for the underlying networking, computing and storage facilities, security and privacy remain second class citizens. In this thesis we present Panoptic, a decentralized distributed system, with support for edge-clouds, that enables the search of missing people, similar to the commonly known Amber alert system, in high density scenarios, e.g. concerts, while featuring privacy and security by design. The system makes uses of both computer vision, to identify people, and security techniques to enforce privacy and data security. Since the limited resources present in the mobile devices, namely battery capacity, Panoptic offers a computing offloading that tries to minimize data leakage while offering acceptable levels of performance. Since it access sensitive data, i.e., private photos, to search for potential matches, this work has the oversight of a certified/trusted authority. In order to keep the information private, every communication channel is secured and all caches remain encrypted. The use of an edge-cloud is a key factor as it allows for faster responses, making use of data locality, and avoids the shipping of private data to public clouds. Our results show that it is achievable to run these algorithms in an edge-cloud configuration and that it is beneficial to use this architecture to lower data transfer through the wireless infrastructure while enforcing privacy.

Resumo

O aumento das capacidades dos smartphones estão a abrir caminho para novas aplicações através de crowd-sourcing dos recursos inexplorados, para formar redes hiperlocais conhecidas como edge-clouds. Enquanto que já existe trabalho desenvolvido para a rede subjacente, computação e armazenamento, segurança e privacidade permanecem em segundo plano. Nesta tese apresentamos Panoptic, um sistema distribuído descentralizado, com suporte para edge-clouds, que permite a procura de pessoas desaparecidas, semelhante ao já conhecido sistema Amber Alert, caracterizado pelo uso de privacidade e segurança. O limite de recursos presente nos dispositivos móveis, nomeadamente a capacidade da bateria, Panoptic oferece o descarregamento computacional que tenta minimizar o vazamento de dados enquanto que contribui para níveis de performance aceitáveis. Como acede a dados sensíveis, ou seja, fotos privadas, para procurar possíveis correspondências, este sistema tem a supervisão de uma autoridade certificada/verificada. De forma a manter a informação privada, todos os canais de comunicação são seguros e todas as caches são cifradas. O uso de uma edge-cloud é um fator chave porque permite uma resposta rápida, usando os dados locais, e evita o envio de dados privados para clouds públicas. Os nossos resultados mostram que é possível executar os algoritmos numa configuração edge-cloud, e que é benéfico usar esta arquitetura para diminuir a transferência de dados através de um sistema de cache mantendo a privacidade.

Acronyms

QoS Quality of service

GO Group owner

Contents

Abstract	III
Resumo	V
List of Tables	XI
List of Figures	XIV
1 Introduction	1
1.1 Context	1
1.2 Motivation	3
1.3 Problem Statement and Proposed Solution	3
1.4 Contributions	5
1.5 Thesis Structure	5
2 Related Work	7
2.1 Cloudlets	7
2.2 Edge-Clouds	9
2.3 Privacy-Based Systems	11
2.4 Face Detection and Recognition	12
2.5 Summary	18

3	Architecture	19
3.1	Workflow	21
3.2	Summary	23
4	Implementation	25
4.1	Backend	26
4.2	Frontend	27
4.3	Summary	32
5	Evaluation	33
5.1	Experimental Setup	33
5.2	Evaluation	35
5.3	Summary	41
6	Conclusion	43
6.1	Discussion	43
6.2	Future Work	44
A	Content of the .proto file	51
B	Display of the experiments configurations	55
C	Tables with the results from the experiments	61

List of Tables

2.1	The difference between the Cloudlet and the Cloud	8
5.1	Memory used by the application during execution time	37
5.2	Quantity of Megabytes[MB] received by the server (R) and quantity transmitted (T) back to the devices	40
C.1	Execution time taken to process each dataset on different configurations	61
C.2	Energy used per device for each dataset	62
C.3	CPU Load spent to process the workload	62

List of Figures

1.1	Distance to the different types of cloud measured from the mobile device.	2
2.1	Example of an edge-cloud architecture	9
2.2	The value of the integral image at point (x, y) is the sum of the pixel values to the left and above. Adapted from [26].	12
2.3	In the first row, the image shows the two features selected by the algorithm. The second row shows the features overlaid on a training face. Adapted from [26].	13
2.4	Scheme of Attentional Cascade. Adapted from [26].	13
2.5	Algorithm executing the search using the template visible at the top of the image.	14
2.6	The different stages during the execution of the Skin colour segmentation algorithm. 1) Reduce the image resolution. 2) Eliminate all non-skin pixels. 3) Segment all the regions with skin pixels. 4) Eliminate all non-face regions through geometrical analysis. Adapted from [29]. . . .	15
2.7	Example of eigenfaces from AT&T Laboratories Cambridge.	15
2.8	The subspace LDA face recognition system. Adapted from [32].	16
2.9	Gaussian Faces implemented as a binary classifier. Adapted from [33]. .	17
2.10	Gaussian Faces implemented as a feature extractor. The flipped version of the joint feature vector is used for robustness. Adapted from [33]. . .	17
3.1	Illustration of the architecture	19

3.2	Hyrax's Middleware Architecture	20
3.3	Illustration of the workflow	21
4.1	Layout of the implementation	25
4.2	Diagram of the work done in the server	27
4.3	Diagram depicting both of the services in a pipeline manner	31
4.4	Layout from the application	32
5.1	Depiction of the execution time used to process each dataset	36
5.2	Chart showing the power consumption values	38
5.3	Chart showing the CPU load in each configuration for every dataset	39
5.4	Chart depicting the amount of data transmitted to the devices	41
B.1	Configuration with 1 device	55
B.2	Configuration with 4 devices	56
B.3	Configuration with 4 devices set as an Edge-cloud	57
B.4	Configuration with 8 devices	58
B.5	Configuration with 8 devices set as an Edge-cloud	59

Chapter 1

Introduction

The constant evolution of mobile devices and their subsequent increase of processing power have led to the development of their applicability. Consequently, new technologies have emerged, such as Wi-Fi connecting the mobile device to the Internet through routers and access points, the GPS (global positioning system) and the edge-cloud system based on the presently known cloud paradigm, to name a few.

With the progression of mobile devices, edge-cloud computing has become a plausible solution as a complement to the Cloud [1]. Instead of sending information to a cloud, a local cloud is created with the devices that are physically close and a shared resource pool is created between them, in order to distribute the computational load.

1.1 Context

The Cloud system is a network of remote servers hosted on the Internet to store, manage, and process data. It was created with the purpose of enabling high-performance computing, normally used by military and research facilities, to consumer-oriented applications. A cloud system consists of hosting services, for example a storage system, over the Internet and removing the need for building some kind of infrastructure locally to host it. Using this kind of system allows for a service to be provided in any part of the Internet. In addition, this architecture usually has more power to process information than normal computers.

Most clouds are stored in data warehouses and can outperform a normal computer since they have access to more resources [2]. The Cloud system has some disadvantages

that can discourage its use [1]. One of the problems is its high maintenance cost and the distance between the Cloud and the mobile device, that can lead to high latencies (and associated variation) that are not suitable to the target application, e.g. augmented reality.

Before introducing edge-cloud computing, it is necessary to introduce a technology that can be used as complement of the Cloud service. The alternative is called Cloudlet and consists of using smaller servers (mini-pc) to provide the same service as the Cloud but with less processing power.

Cloudlets can be used as a cheaper complement to the Cloud service avoiding the need of increasing the cloud size. The Cloudlet is installed locally, closer to the mobile devices. This technology is deployed when a cloud service is needed but cannot afford a higher jitter, in other words, a higher variation in the latency on a package flow between two systems, which can result in network congestion and can degrade the quality of communications.

The main difference between the Cloud, the Cloudlet and the Edge-cloud is their proximity. Each architecture becomes closer to the mobile device. The Cloud is created through a set of servers combined together to share their resources. This architecture is located nearer of the backbone of the network.

The Cloudlet is installed locally using a server, closer to the edge of the network. Because it is installed nearer of the edge, this type of architecture will only share its resources to the devices that are physically nearby to the Cloudlet. And the Edge-cloud is implemented through the connection of several mobile devices creating a pool of shared resources. Another characteristic that differentiates the Edge-cloud from the other architectures is the absence of an infrastructure. These architectures can be implemented independently or combined with one another. A diagram illustrating the distance from the mobile device to the architectures can be seen in Figure 1.1

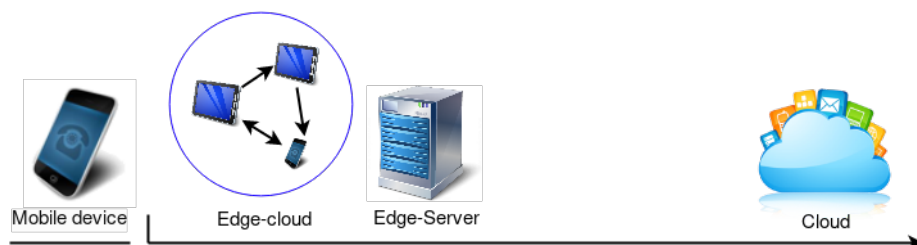


Figure 1.1: Distance to the different types of cloud measured from the mobile device.

1.2 Motivation

Edge-Cloud allows mobile devices to carry a heavier workload than what could be achieved by a single device. This form of computation can fill some gaps that cannot be covered by a cloud, such as providing computational capabilities in places where a steady connection to the Internet cannot be guaranteed. Furthermore, if the workload is heavier for a single device, but not justifiable enough to be transferred to the Cloud, it can be sent to the Edge-cloud due to its proximity to the device and its capacity to do heavier tasks.

Despite being favorable to the system for every connected device to share its resources and data, the user should be allowed to choose whether if he wants to share them or not, due to privacy and trust concerns.

Privacy has become a priority in the development of new software. The amount of critical data that goes through the network has increased over the years and with more services becoming available online the tendency will persist. Private information can be harmful in malicious hands and now people are more aware of the importance of online privacy. Furthermore, recent leaked information has shown to everyone the seriousness implied to this subject [3].

1.3 Problem Statement and Proposed Solution

The dependency on clouds by users has intensified, but the use of clouds has some disadvantages. If there are too many requests or if the network connection is not reliable this can affect directly the quality of service(QoS). One possible solution could be the enlargement of the Cloud and reinforcement of the connection, which can become costly.

A cheaper alternative is the deployment of several Cloudlets, servers that are closer to the edge of the network and can complement the Cloud by handling requests from the mobile devices. But this type of architecture has also the cost of installing servers in various locations.

Another solution can be the use of Edge-clouds. Several mobile devices that are physically close can create this type of cloud by connecting between them and sharing their resources. Through resource sharing, the devices can have more computational capacity and execute some tasks that are executed in the Cloud. Unlike the other

solutions, this one has no additional costs besides the acquisition of the mobile device, due to the fact that the technologies used are already embedded in most of the mobile devices.

Edge-clouds can be a viable solution for deployment of heavier algorithms, like face detection and recognition algorithms. Face detection is defined by the identification of the presence of faces on a given photo, returning its location and extent. Face recognition consists of the identification of a specific person, usually within a database of potential target subjects. Both have been used as tools for different types of tasks. They can be used by law enforcement to detect a possible suspect or missing person in the middle of a crowd. They can also be used for detecting a person's presence in a photo gallery, or for biometric verification [4].

Implementing heavier algorithms can be challenging for one mobile device due to the constraints of the processing power, limited storage and limited battery. A possible solution would be to upload the data to a cloud for execution. However, this is only viable if the connection is steady. A more trustworthy approach is the Edge-cloud system, since it can surpass the connection problem due to its proximity to the devices.

A problem in Edge-cloud computing, related with the communication of data, is the passage of sensitive information without any form of maintaining its privacy. Data privacy is the ability to control information about oneself. This can be translated as the capacity of protecting personal data through the use of policies, data regulation and technology [5]. The use of policies and data regulation includes agreements and laws created for data protection in case of misused information or information leakage [6]. Personal data can also be protected through technology by employing encryption and anonymity techniques. With the present technological advancement a general concern has been raised related to the impacts of information privacy [7].

This thesis proposes a system that seamlessly combines three fields, the Edge-cloud system, computer vision, and data privacy. Our aim is to prove that it is possible to run more exigent algorithms, without the downside of depleting the mobile device, and at the same time having a private layer to avoid information leakage.

Our test case is intended to help the authorities in the search for missing people, following the principles present in the Amber Alert system. It is designed in light of the new GDPR (General Data Protection Regulation) [6], where we pursue data minimization, user empowerment through informed consent and accountability/traceability.

From a technical view, our system accesses the galleries of the connected devices

and, using the computer vision algorithms, tries to identify the missing person. This form of approach can be invasive and to prevent any breach of privacy the service uses encryption techniques while minimizing the amount of sensitive data that is transferred.

1.4 Contributions

In this thesis we have made the following contributions:

- **Modular Architecture** - we have proposed a modular architecture with privacy and security by design, that tries to offer a transparent approach to the use of edge-clouds when dealing with sensitive data. We have taken previous work by [8] and re-designed its middleware to feature security and privacy.
- **Practical Test Case** - an implementation of an actual application case, the Amber Alert System, where sensitive data is accessed and potentially migrated between nodes.
- **Implementation** - a full implementation has been done in an effort to provide empirical evidence on the feasibility of our concepts, including backend support and an Android mobile application.
- **Evaluation** - we offer a complete evaluation on the impact of our system, namely, the energy consumption on the devices and the traffic on the wireless infrastructure.

1.5 Thesis Structure

This thesis is divided into five chapters. Each one explains a different part of the progress that was made during the execution of this thesis.

In chapter 2, we present the current state-of-art on each different area as well as other similar applications and frameworks that are available.

In chapter 3, we describe the architecture behind the service that was developed and give a brief sighting of the work-flow.

In chapter 4, a more detailed work-flow is described and the decisions took during development are explained.

In chapter 5, the results of the experience are shown and evaluated.

In chapter 6, a conclusion of the thesis is presented with a discussion of the work produced and future work.

Chapter 2

Related Work

In this chapter, we present the most relevant state-of-art for this dissertation. We intend to focus on the combination of the edge-cloud system, face detection and recognition, and privacy system. Unfortunately, there is no state-of-art that fully combines these fields in order to provide an insight on the application of computer vision algorithms to an edge-cloud, with an embedded privacy system.

Some researches present a hybrid system that uses the edge-cloud system combined with other types of cloud. The algorithms and technologies, to avoid depletion of the mobile device, are deployed on a higher level, for example a server or a cloud. To the extent of our knowledge, this is the first thesis to consider the possibility of combining the three fields to provide a viable alternative/complement to the Cloud.

Section 2.1 describes the cloudlet technology differentiating it from the cloud and presents previous work done in this field. Section 2.2 exposes the research done on Edge-clouds. In section 2.3 the current state-of-art of privacy is shown. Section 2.4 summarizes key face detection and face recognition algorithms, succinctly explaining them.

2.1 Cloudlets

The Cloudlet is a small-scale cloud located near the edge of the Internet. It can be considered as the middle tier in a four-tier hierarchy: the Cloud, the Cloudlet, the Edge-cloud and the mobile device. The purpose for implementing this system is to “bring the cloud closer” to the user. This is achieved by switching the connection from

the Cloud to the Cloudlet, which is nearer to the mobile device.

The system is installed locally through a small server and allows the augmentation of every mobile device connected to it by leasing some of its resources. Physical proximity is essential, because it allows a lower latency and a higher bandwidth, permitting a much faster end-to-end response.

The Cloudlet only has a soft state, meaning that it has cache copies of data or code that are available elsewhere. This is a significant advantage because there is no critical danger to the data if the Cloudlet is destroyed or lost [9]. Table 2.1, shows the differences between the Cloudlet and the Cloud.

	Cloudlet	Cloud
State	Only soft state	Hard and soft state
Management	Self-managed, little to no professional attention	Professionally administrated 24 × 7 operator
Environment	"Datacenter in a box" at business premises	Machine room with power conditioning and cooling
Ownership	Decentralized ownership by local business	Centralized ownership by Amazon, Yahoo!, etc
Network	LAN latency/bandwidth	Internet latency/bandwidth
Sharing	Few users at a time	100s-1000s of users at a time

Table 2.1: The difference between the Cloudlet and the Cloud

Presently, there is a research that developed a similar model to our system, Cachier [10], an edge caching system that offloads work from the mobile device to a cloud and uses a cache system in an edge server. The cache stores analyzed images to respond to repeated requests. The use of an edge server lowers the latency of a mobile device when sending a request for an image analysis, because similar requests are saved in the edge cache. However, no security nor privacy is addressed by this work.

Another research on Cloudlets, FemtoClouds, combines the use of a cloudlet with mobile devices, providing a dynamic and self-configuring mobile cloud system that allows the scaling of the cloudlet computation capacity, by coordinating multiple mobile devices. The system takes leverage on nearby unutilized mobile devices creating a computational cluster, providing a compute service.

Initially, a mobile device sends its information (computation availability of the mobile device, computation available to share, utilization history) and sharing policy to the cloudlet, in order to join the computational cluster. Based on the new device's computational availability and battery level, the cloudlet can choose to refuse the inclusion of the new device into the cluster. A mobile device can offload an intensive task to the cloudlet, which then schedules to the cluster, by calculating the required computational time in the available mobile devices [11, 12].

Edge Accelerated Web (EAB)[13] Browsing is a prototype developed for mobile edge computing to fasten web execution. In EAB, a cloudlet is deployed between the mobile device and the server. When a mobile browser sends a request for an URL page, the response is intercepted by the cloudlet, which then excludes some of the contents.

The tasks done by the cloudlet consists in fetching and evaluating web contents, layout of the contents component, and task rendering. This prototype can outperform the normal web browsing [12].

2.2 Edge-Clouds

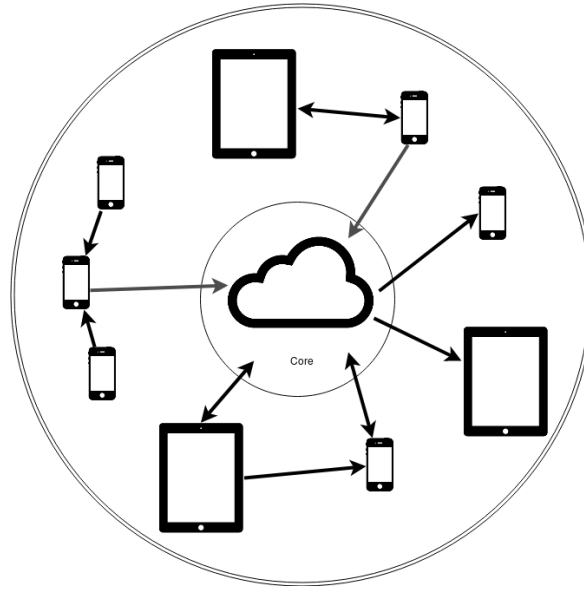


Figure 2.1: Example of an edge-cloud architecture

Edge-Clouds, also known as Mobile-edge Computing(Figure 2.1), are a recent technology. With the development of mobile devices, it has become reasonable to develop and implement a local cloud system to aid the ever-growing mobile traffic.

This architecture can be described as a group of connected mobile devices that share their resources, creating a cloud constituted only by mobile devices. The system can run locally and access local resources, while being isolated from the rest of the network. Because of its closeness to the edge, it has a lower latency, improving the user's experience and minimizing the congestion in other parts of the network [14].

Currently, the research on Edge-clouds is growing and some frameworks have been developed to implement mobile edge-clouds, for example, the Hyrax [15], the first system to propose the idea of distributed, cloud-like computing on mobile, edge devices. Hyrax has been used before to implement distributed storage systems [16] by utilizing peer-to-peer techniques to create a pool of shared resources. Another system that provides a reliable storage system in mobile ad-hoc networks is Pan [17], though it does not allow work distribution between the nodes.

The research done by [18], proposes Replisom, an edge-cloud architecture to reduce cloud responsiveness when multiple Internet of things (IoT) devices replicate memory objects to the edge-cloud through LTE environment. This architecture augments the evolved NodeB with cloud computing resources at the edge that provide virtual machine, storage and network resources for specific applications [12].

The edge-cloud pulls the replica to the respective virtual machine, when multiple IoT devices try to update the memory objects, each device sends the new updated memory object to the neighbor devices using device-to-device communication, instead of uploading it to the edge-cloud. The receiving device compresses the memory replicas into a single replica, that is later used to respond to the periodical requests sent by the edge-cloud. The edge-cloud recovers the memory object using compressed sampling construction algorithms[12].

Another form of edge-cloud computation is used in the framework proposed by Gao. Computation offloading among the peers of mobile devices [19] is a probabilistic framework that offloads mobile computation among the peers of mobile devices within the tactical edge. In war-zone areas, applications such as processing in-situ sensory data about the nearby environment takes great amounts of computation.

This framework offloads part of the application to the nearby nodes to reduce computational time and energy consumption. The decision made by the node to offload work, depends on the computational power and energy level of its neighbor, and the probability of a future contact between them, which is done by applying the properties of inter-contact time (ICT).

If the new node ensures that the task is completed within the contact period then the task is offloaded. The framework can efficiently distribute the workload between the nodes, but does not consider if a node suddenly leaves the network [12].

2.3 Privacy-Based Systems

Mobile devices play a very active role in the daily life. Because of this, there is a continuous improvement of the mobile device with the purpose of enhancing the user's experience. Since the device's embedded systems and sensors can be used without the user's knowledge, nowadays a mobile device can be a powerful surveillance tool. From these data it is possible to know details of the user's daily life that otherwise should be private [20].

The necessity of protecting personal privacy is increasing, and developers when designing new systems, that handles critical data, must design their architectures with privacy as a mandatory goal [21] at the risk of losing users trust [22].

In order to achieve privacy, a system is required to properly implement a policy design, which includes user access and control. From a technical standpoint, data protection is a fundamental building block. To do so, there are two main models, namely pro-active and reactive.

The proactive model tries to implement encryption techniques to encrypt the data, while the reactive tries to find flaws in the system after the attack occurred and react accordingly. Both approaches add overhead, albeit with different levels of resources needed [23] for each approach.

In mobile ad-hoc networks an approach advised to solve the security problem is using the principle of distributed trust. No single node is trustworthy in a network, but threshold cryptography can distribute trust to an aggregation of nodes. This method divides a private key into n shares. Through the combination of identity-based techniques with threshold cryptography it is possible to achieve flexible and efficient key distribution [24].

Another proposition is the use of SCAN, a framework developed to add a layer of security to the mobile ad-hoc networks. This layer allows the node to monitor the routing and packet forwarding behavior of its neighbors, and can detect if there are any malicious nodes in its surroundings [25].

2.4 Face Detection and Recognition

In the field of face detection, there are several approaches that can help with the detection of faces in a given photo or film. Depending on the situation, some algorithms can surpass others and achieve the best result.

Face detection

The following list presents some algorithms used for face detection.

1. ViolaJones, also known as Haar-Cascades algorithm, evaluates images based on its features. The use of these features allows the system to execute faster than using a pixel-based system. Another option to speed up the algorithm is to use an intermediate representation instead of the original image (the integral image, Figure 2.2) [26]. This image at location (x, y) contains the sum of the pixels above and to the left of (x, y) from the original image:

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y')$$

where $ii(x, y)$ belongs to the integral image and $i(x', y')$ to the original image. This tool enables the use of rectangular features during the execution of the algorithm.

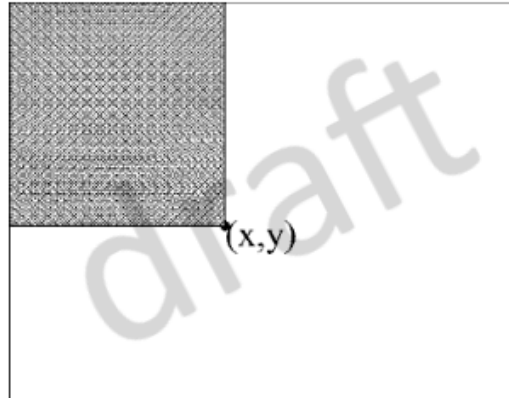


Figure 2.2: The value of the integral image at point (x, y) is the sum of the pixel values to the left and above. Adapted from [26].

The use of rectangular features over the integral image is a very classic approach in this algorithm. There are other alternatives, such as steerable filters, which can be more suitable for analysis of boundaries, for image compression or for

texture analysis. In this case, their use can be more efficient because it supports a more effective learning. This characteristic alone can overcome the disadvantages. Rectangular features are created with different aspect ratios, starting at the base scale, diminishing by a given factor, until it detects the face.

Supporting this algorithm, a variant of AdaBoost [27], shown in Figure 2.3 for feature selection and classifier training was included, eliminating the need to create all the rectangle features. AdaBoost uses several weak classifiers to create a strong classifier.



Figure 2.3: In the first row, the image shows the two features selected by the algorithm. The second row shows the features overlaid on a training face. Adapted from [26].

When running with AdaBoost, the first selected feature is the region of the eyes, because the area is darker than the region of the cheeks and nose, and the second feature relies on the fact that the eyes are darker than the bridge of the nose. This proves that the technique is effective.

Another technique added to the algorithm, increasing its effectiveness, was the use of Attentional Cascade after AdaBoost.

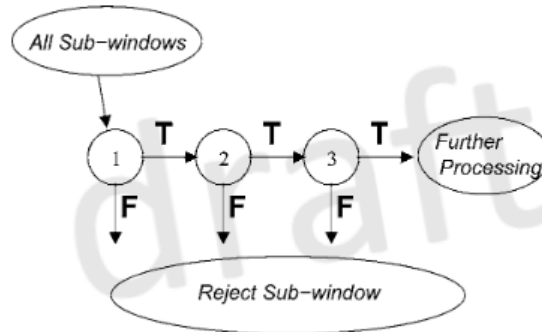


Figure 2.4: Scheme of Attentional Cascade. Adapted from [26].

The cascade consists of several evaluations done to the classifier that was created by AdaBoost. If the classifier fails one of them, it is then immediately discarded, as shown in figure 2.4. This process enables a more accurate classifier to detect the face.

The use of these techniques allows for a more efficient and faster algorithm.

2. Template matching algorithm uses a sub-image as a template and applies it over a given image. After its insertion in the search area, at each position, the correlation is calculated, quantifying the similarity between the two images. The use of a threshold over the correlation allows the disposal of false positives [28]. This algorithm is better used when there are other parts of skin present. An example of the algorithm analyzing an image can be seen in Figure 2.5.



Figure 2.5: Algorithm executing the search using the template visible at the top of the image.

3. Skin colour segmentation algorithm(Figure 2.6), as the name itself suggests, uses skin colour to detect faces. Because skin colour has a very strict range of pixel value, it is possible to distinguish it from the rest of the image. With the use of heuristics, each pixel can be evaluated as skin or non-skin. After the evaluation, the image is segmented, eliminating the non-face regions through geometrical analysis [29].

Face Recognition

The problem of face recognition has been researched for a long time because of its broad application and the challenges that it provides. The face recognition algorithm can still be influenced by environmental characteristics, such as pose, lighting, face



Figure 2.6: The different stages during the execution of the Skin colour segmentation algorithm. 1) Reduce the image resolution. 2) Eliminate all non-skin pixels. 3) Segment all the regions with skin pixels. 4) Eliminate all non-face regions through geometrical analysis. Adapted from [29].

expression. The list that follows presents some of the current algorithms used for face recognition.

1. EigenFaces algorithm: it uses a representation for each image, seeing each one as a vector. Using this representation, it can create a high-dimensional vector space, combining all the face images in one set. After this, the algorithm uses, over the vector space, a probability distribution, creating a covariance matrix. The *eigenvectors* derive from that matrix, with each image contributing a set of pixels to the *eigenvector*, creating an image that resembles a ghostly face (the *eigenface*, shown in Figure 2.7).

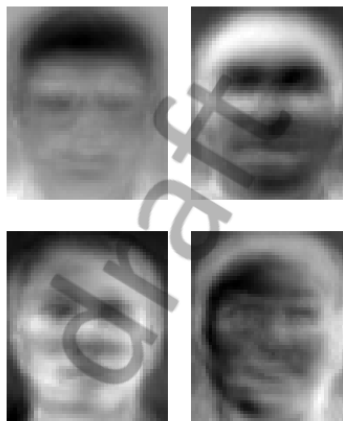


Figure 2.7: Example of eigenfaces from AT&T Laboratories Cambridge.

The execution flow in this algorithm is very simple. After acquiring the initial set of face images, it calculates the *eigenfaces* [30]. This procedure creates the *facespace*, which is a combination of images that have the highest eigenvalues. Then, it calculates the distribution for each known individual by projecting it onto the *facespace*. This concludes the training phase.

In the execution phase, the algorithm receives a face image and determines its weight by projecting it onto each of the eigenfaces. Subsequently, it confirms if it is a face by checking if it is sufficiently close to the *facespace*. If it passes the authentication, it checks its weight pattern, classifying it as a known or unknown person [30].

Optionally, the algorithm can be optimized, after the classification of the input image, by updating the *eigenfaces* and/or weighting patterns, and by calculating the weight pattern and incorporating it into the known faces, if the unknown image appears more often.

2. FisherFaces algorithm: originally, this algorithm was used to solve taxonomic problems through multiple measurements. Given two or more populations, it is possible to distinguish certain traces that characterize each population. With this approach, the algorithm can be implemented for face recognition because each face has certain unique characteristics, which allows them to be told apart from one another [31].

The algorithm has two steps of execution. The first step uses the Principal Component Analysis (PCA), a technique which is also used in EigenFaces, with the purpose of projecting the face image into a face subspace. After that, the PCA projection vectors are used in the Linear Discriminant Analysis to form a linear classifier in the subspace [32]. A diagram with the system is shown in figure 2.8.

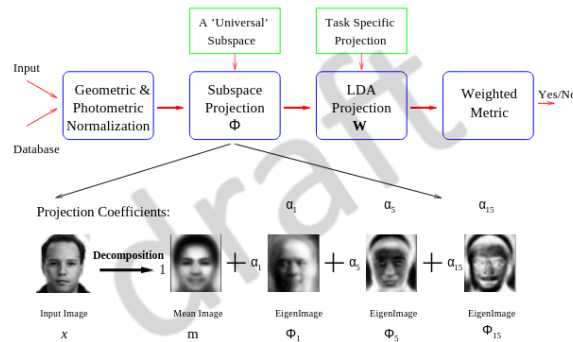


Figure 2.8: The subspace LDA face recognition system. Adapted from [32].

3. Gaussian Faces algorithm: this algorithm can be implemented in two ways, as a binary classifier(Figure 2.9) or as a feature extractor(Figure 2.10), combined with local binary patterns. Each face is first normalized to a 150×120 size and is later divided into overlapped patches of 25×25 pixels. Each patch is mapped, by a given descriptor, to a vector. A multi-scale local binary pattern feature is generated in each patch. As a binary classifier, it receives a pair of face images and calculates its similarity vector, predicting if it is the same person or not.

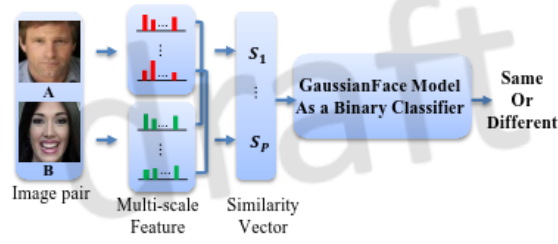


Figure 2.9: Gaussian Faces implemented as a binary classifier. Adapted from [33].

When the algorithm is used as a feature extractor, after learning the hyper-parameters from the training data, it estimates the latent representations. After that, it automatically groups the latent data points into different clusters. This can be seen as a codebook generated from the model. Like the classifier version, it receives a pair of face images and, besides the multi-scale feature, it also determines the joint feature vector.

When an unseen pair of faces is added, the algorithm checks the joint feature for each pair of patches and estimates its latent representation. Finally, it concatenates the new high-dimensional features from each pair of patches to form a new high-dimensional feature for the pair of face images. This can be compared to the features created from the training data [33].

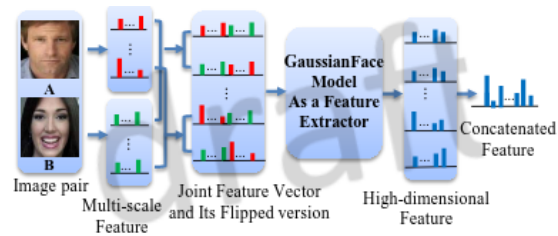


Figure 2.10: Gaussian Faces implemented as a feature extractor. The flipped version of the joint feature vector is used for robustness. Adapted from [33].

2.5 Summary

In this chapter the state of art was presented. As it was possible to verify there are many different approaches to solve some problems that might present during the development of the system. Some decisions were made based on this study: for face detection and recognition we chose ViolaJones and EigenFaces, respectively, because of the above-mentioned advantages and low energy requirement. The Hyrax system was selected for connecting the mobile devices and create a local cloud system because it provides nearby connectability and a storage system.

Chapter 3

Architecture

In this chapter the architectural design is presented. For each element that partakes in the architecture it is given a brief explanation of the element and a description of its function and/or contribution to the system. In Figure 3.1 it is possible to see the schematics of the architecture.

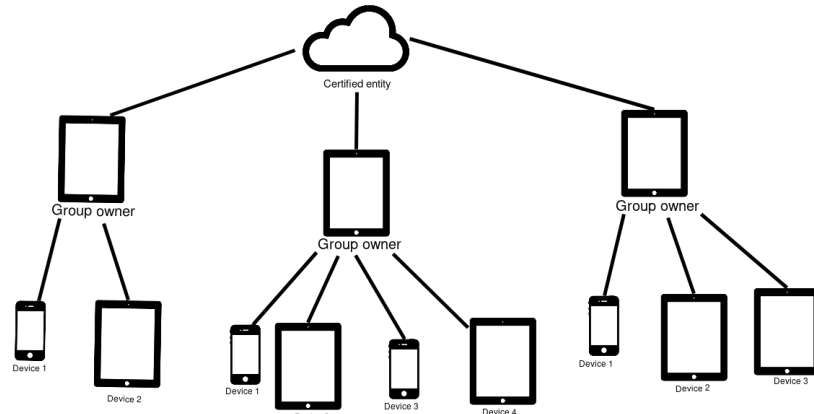


Figure 3.1: Illustration of the architecture

Furthermore, in section 3.1 a succinct description of the workflow is given, to help understand the connection between the different elements. A diagram of the workflow can be viewed in Figure 3.3.

The architecture is composed by a server, edge-clouds and mobile devices, with the application installed in each mobile device. The server, maintained by a trusted entity (e.g. police), is used to verify every search request that comes from the edge-clouds and also work as a storage system, to keep the possible matches that came from the devices.

Edge-cloud can be seen as a group of mobile devices that are physically nearby and connected between them. While there are several methods to form these clouds, a common way is to use WiFi-Direct, using hotspots, to interconnect groups of devices. The node that is responsible for maintaining this group is denominated as group owner(GO). For a mobile device to become a group owner, first it searches its surroundings for other devices, looking for an existing GO. If no device was found then the mobile device becomes a GO. These form a tree like structure around the infrastructural Wi-Fi infrastructure (normally, access points - APs).

Another function of the GO is to keep a cache for incoming responses from the server, to avoid redundancy of requests to the server. Each device establishes a connection to the GO through WiFi-Direct. This technology creates a connection between the devices that are close to each other, without the need of an access point or a router. To prevent overcrowding each group, the GO only allows three connections at the same time, if there are more devices to connect, another group is created.

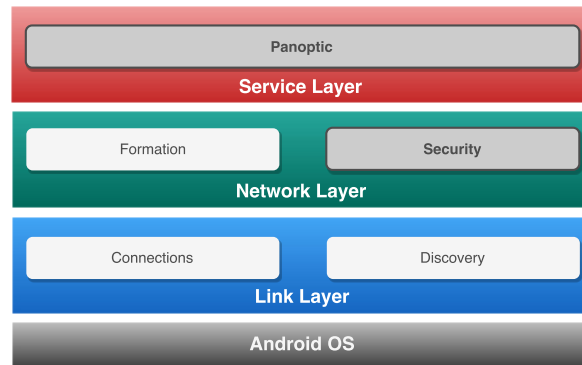


Figure 3.2: Hyrax's Middleware Architecture

The application developed for this system was build for Android. We chose this operating system because, at the moment, it has the largest market share on mobile devices [34] and its openness allows for added flexibility. We extended previous work developed in Hyrax [8], namely, we made use of the lower level network layers present in their middleware, as shown in Figure 3.2.

We had to redesign the architecture to feature security and privacy by design, as such, we introduced a novel "security" component for the network layer. Additional, a completely new service, dubbed "Panoptic" was incorporated in the service layer. The remaining middleware components, namely, the Link Layer and the formation portion of the Network Layer were used without modifications, as they already offered the needed functionality for our system.

Since the data touched by the application is sensitive, as it accesses the gallery of the mobile device, we had to introduce secure channels into the architecture, in an effort to avoid problems such as communication snooping. In our current design, the data channels between the devices and trusted server use HTTPS. While we could use the same approach in device-to-device communication, that would imply the deployment of certificates to all devices/users. This is feasible but undesirable, given the added management overhead needed to provide a trusted authentication infrastructure, that would also increase the setup time for an unexperienced user.

Because of this, we decided to use TCP for device-to-device communications and perform the symmetric encryption over that data pushed along these channels. Furthermore, the data injected by the user is signed, so it is not possible to forge requests on behalf of the server.

3.1 Workflow

The workflow of the system, shown in Figure 3.3, goes as follows, the user requests the server to search for a missing person by sending the photo of the missing person's face. The entity responsible for the server analyzes the request. This is done in order to prevent misuse of the system. For example, a stalker could use the system to prey a potential victim. If the request is approved the photo is added to the train dataset of the recognition algorithm.

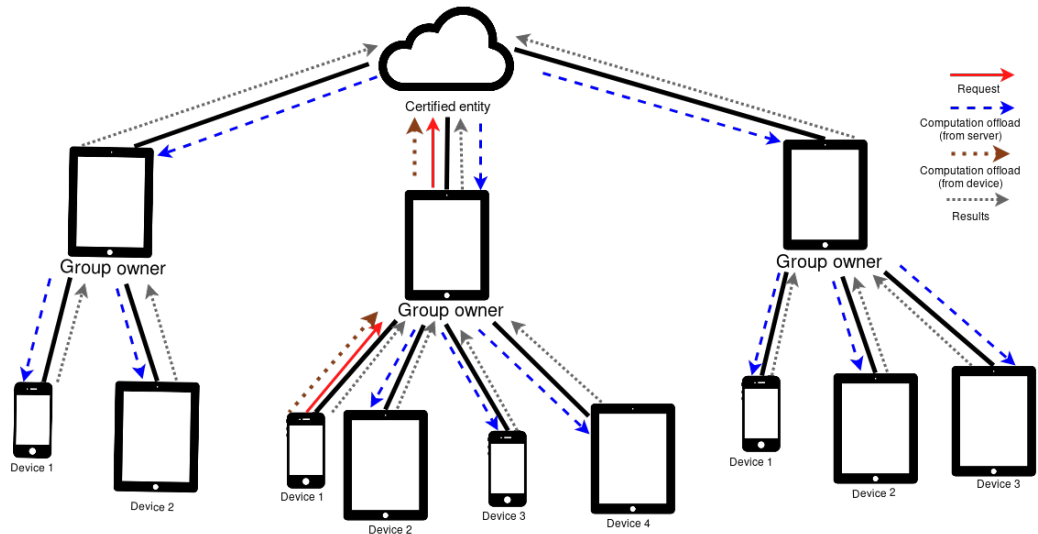


Figure 3.3: Illustration of the workflow

A train dataset can be described as a set of elements used to teach an algorithm, in the case of face recognition, a set of identified faces are used to create a model so it can later be used, by the algorithm, to identify new faces.

Before broadcasting the train dataset, the server associates a new encryption key in the search session. By doing so the server can maintain multiple searches at the same time without risking an overlap of photos from different searches, and thus archiving isolation and forward secrecy. Every photo that comes from the server is encrypted with a key and signed by the server to maintain the privacy and avoid photo leakage.

The server then broadcasts the train to every device present in the edge-cloud. In each device, the missing person will be searched on the device's gallery, including the requesting device. If there is a match, the matching photo is sent back to the server for further analysis. The device that made the request is the only device that is able to see the results from the search, besides the server.

In the mobile devices, if their photo gallery size is greater than a specific threshold, by default 2 photos for the requesting device and 5 photos for other devices, the exceeding photos are sent to the server to be processed by other available nodes. This helps minimizing the potential battery drain of a particular device. The photos received in the server are sent to any device that is ready and waiting to process more images.

To protect the trusted server from being flooded by repeated requests from the devices, the system keeps a cache with the responses from the server. The cache is kept in a device to whom the other devices are connected. Furthermore, in order to prevent the cache from overgrowing with stored responses and avoid depletion of the device's resources, the system checks for old responses and removes them and/or removes responses when the maximum size is reached, to store newer responses.

The work distribution service is executed in background and the user has no control over this. The user knows that the service is running because the application shows a load bar when the device is processing images.

In order to be more transparent, the application has additional features to give some control to the user. One of the features is denying resource sharing by not processing photos from other sources. Another feature is denying photo sharing, i.e., the user can choose not to send any photo from his gallery to the server. By picking this option the analysis of the photos are made in his own device.

3.2 Summary

With this chapter the architecture design was presented as well as the workflow, demonstrating the connectivity between the different elements. This description serves as an overview before describing in depth the work that was developed in the next chapter.

Chapter 4

Implementation

The present chapter describes the implementation of the system, with a more detailed work-flow. The chapter is divided into two sections. The first section describes the system's backend, created to verify requests and store images. The second section characterizes the frontend, explaining the implementation of the GO server, the background services and the application. A diagram depicting the implementation is shown 4.1.

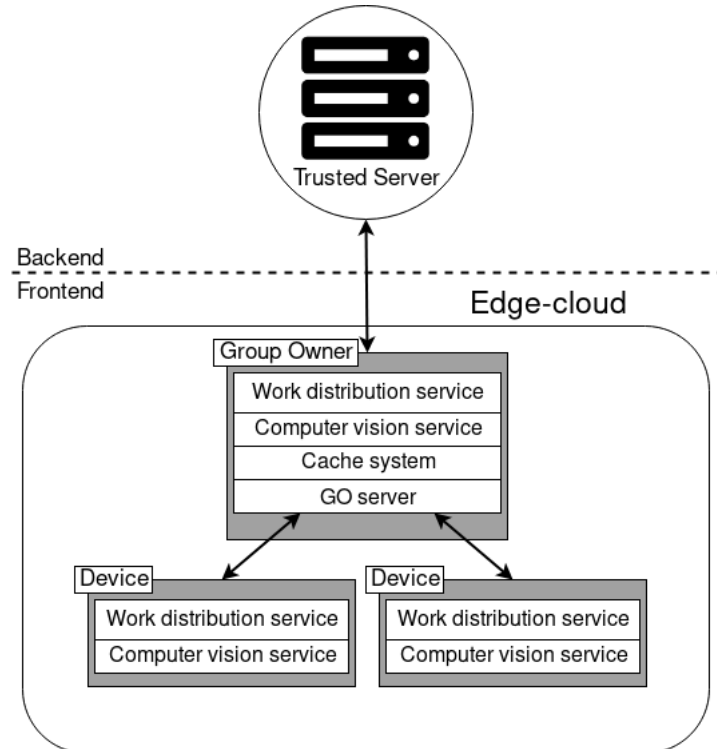


Figure 4.1: Layout of the implementation

4.1 Backend

The backend supports the central authority previously mentioned in our architecture. As a certified authority, it verifies all the requests for missing people and store possible matches. The server was developed using Jetty [35] for maintaining a matching development ambient in Java between the backend and the frontend, and to take advantage of the versatility of this technology. In order to provide confidentiality it only accepts HTTPS connections, thus each device must present the certificate to be allowed to open a communicating channel. A private key assesses the presented certificate.

Given that the server can handle multiple concurrent clients, we chose to make all operations synchronous. To avoid concurrency issues we also used specially designed data structures that could manage concurrency, for example the use of the `ConcurrentHashMap`. Critical zones of the server were safeguarded by using mutex to avoid race conditions when accessing image files.

As the data sent to the devices is sensitive and could pass unprotected channels, it was required to insert a second level of protection. To implement it, each message was encrypted using an AES key and signed by the server, using the signature algorithm `SHA1withRSA`. The signature allows a device to verify the authenticity of the message.

Another implementation was the creation of a periodical task to continuously run in the server, with the objective of verifying the timestamp of all downloaded images that are saved in a list. A copy of each downloaded image is kept in a list with a timestamp of the moment it was downloaded. When the timestamp surpasses the defined limit, by default 5 minutes, the task removes it from the list.

The work of the server is shown in Figure 4.2 and goes as follows, it receives and assesses the search request in form of a face image, if the request is valid, an AES key is generated and associated to that specific search session. A model with the person's missing face is then created for the eigenfaces algorithm. After the model is created the server allows for the available devices to begin the search for the missing person.

Each device, firstly requests the session key and then requests the model to be downloaded. After the download finishes, the search in the device's photo gallery can start. Thereafter, the server starts receiving images from the devices for work distribution. When the server distributes work among the devices, the images downloaded by the devices are kept in a list with a timestamp to indicate that they are not available for other devices to analyze them. If a device takes more then

the normal amount of time, by default 5 minutes, to analyze the images allocated to it, the periodic task removes the image from the list, so it can be downloaded again and analyzed by other device. When the device finishes the search, it sends to the server the matches founded to store them away for further analysis. A diagram showing the work of the server is presented in Figure 4.2.

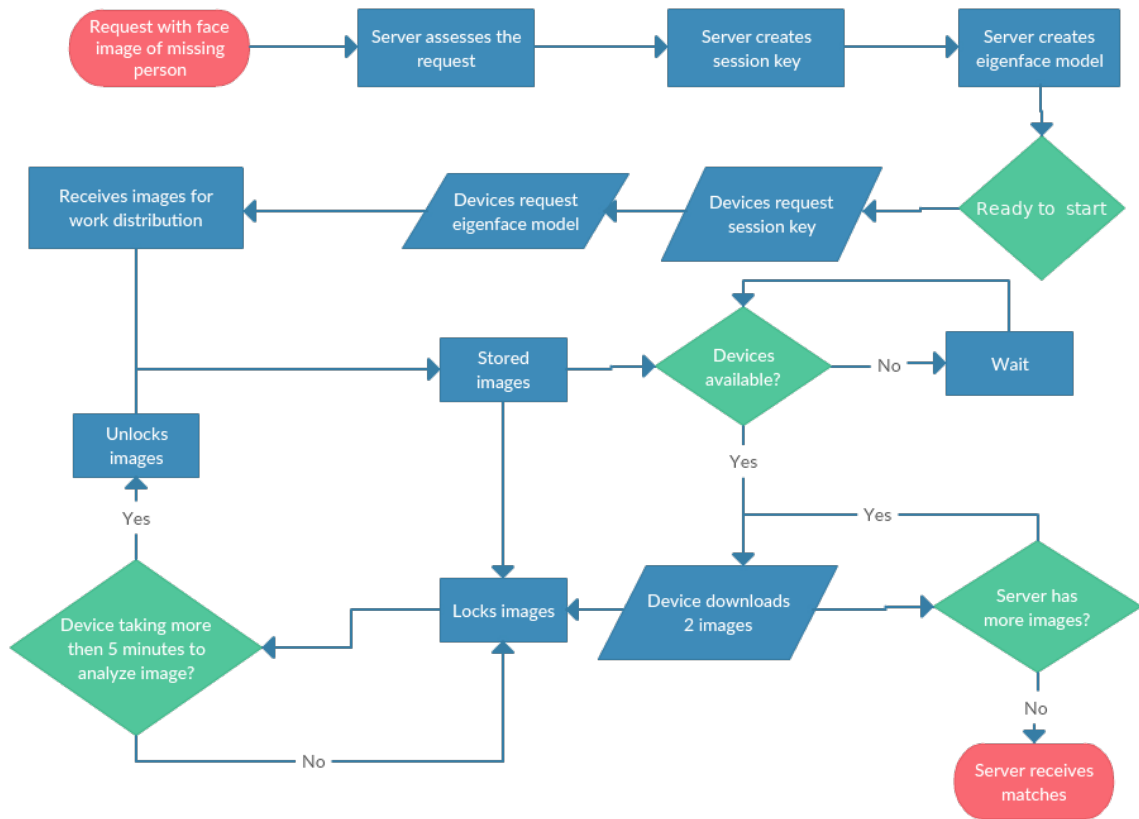


Figure 4.2: Diagram of the work done in the server

4.2 Frontend

The frontend of the system is formed by the GO server, the background services and the application. The basis of the frontend was implemented by using the link layer and the network layer from the work of [8], as it supports device-to-device connection, through WiFi Direct and the Internet connection, through WiFi. The network layer also implements an algorithm that automatically creates a group between the nearest devices.

A logic layer was developed to control the network layer and to use it to implement

methods for communication, server requests and server responses. The connection between server and device is made using the HTTPS protocol through GET/POST methods from the REST API. The GET method allows to retrieve resource representation/information from the trusted server, and the POST method permits to create new resources in the trusted server.

The original work from [8], which this system is based, used HTTP to connect the device and the server, however, because we are accessing private data, the protocol was changed to HTTPS. The use of HTTPS protocol enables the encryption of data that passes through the communicating channel. This protocol requires the use of a certificate trusted by the server to be implemented, which is provided in the system.

The POST/GET methods include generic methods to communicate with the server. One of the methods can list the available sessions and the images to be analyzed, another method can upload a request, an image or an image match, other method can download images to be analyzed or the eigenfaces model and lastly, there is a method to remove images that were analyzed by the device.

Besides the HTTP communication, the communication between devices was originally established through TCP/IP sockets. This transport does not provide any encryption whatsoever to the data that passes through it. While we tried to use gRPC to perform secure RPC, we ultimately decided against it as it had implementation issues in Android. The original implementation of TCP/IP sockets was kept and, as an alternative, to provide a secured transport, the messages are encrypted and signed by the server.

For data transfer, it was used the protobuf encoding to send it through the channels. This type of encoding is a language-neutral, extensible mechanism for marshalling structured data, which is defined in a Interface description language (IDL) like description enclosed in .proto files. These files are used to generate source code for the intended language allowing the instantiation of the desired object. An example of the structure used for this system can be seen in Listing 4.1. In the example shows the generic message to upload an image file to the server.

This message is constituted by a `type` field, a `location` field, a `size` field and a `src` field. The `type` field indicates the type, defined by the enumeration `Type`, of the image that is being uploaded. `FACE` indicates that it is a search request to the server, `IMAGE` indicates that it is an image for workload distribution and `MATCH` indicates that it is an image where the missing person was founded. The `location` field, used only on the `IMAGE` and `MATCH` type, keeps the location where the image was taken.

The size field is used to save the size of the image. And lastly, the src field saves the image's bytes.

```
//generic file object representation
message Image {
    enum Type{
        FACE = 0;
        IMAGE = 1;
        MATCH = 2;
    }
    //image location
    string location = 1;
    //image size
    int32 length = 2;
    //Image type
    Type type = 3;
    //size must be under 1 MB
    bytes src = 4;
}
```

Listing 4.1: Generic image file representation

A security layer was also implemented with methods to encrypt and decrypt messages using the session key that is retrieved from the trusted server, because of the second encryption done to the messages, by the server. This was done in order for the message to be secured while passing through unsecured channels, as aforementioned. For that reason another message was created to encapsulate the encrypted message. An example is shown in Figure 4.2.

The encrypted message is defined by six fields. The label field indicates the search session that the message belongs to, the fileName indicates the name of the image, the timestamp used for the cache in the GO Server which marks the time when the image was downloaded, the signature field where the signature of the message is saved, the cypheredMessage where the encrypted message is kept and the androidId indicating the Id of the mobile device that requested the image.

```
message EncryptedMessage{
    ...
    // file management
    string label = 6;
    string fileName = 7;
    //timestamp used for the cache
    int64 timestamp = 8;
```

```
bytes signature = 9;  
bytes cipheredMessage = 10;  
string androidId = 11;  
}
```

Listing 4.2: Encrypted message encoding. The original message is converted into encrypted bytes and saved in the field `cipheredMessage`

Another addition to the system was the incorporation of computer vision algorithms. We used the implementation from the work of [36] for face detection and exported the eigenfaces algorithm to Android for face recognition. However, new data structures were created to be able to accommodate the algorithm on the Android system.

GO server

The logic layer, besides the generic methods used for performing requests to the trusted server, also implements the GO Server, i.e., an Android Server used when a device becomes a GO, with the purpose of connecting nearby devices, receiving requests from them and forwarding some of the requests to the trusted server. In case the GO loses the connection to the external network, all devices connected to it are disconnected and the procedure to create a GO starts again.

The GO server also features a cache system, that was implemented to be deployed at the same time, with the function of avoiding request flooding the trusted server, by caching in encrypted responses from it. Any repeated request that occurs is responded by the GO server, which can be the model used by the recognition algorithm or photos that need to be analyzed. Although, the latter will only occur if a device loses connection or if it takes too long to analyze the designated photos.

Services

Two Android background services are also featured in the frontend, shown in Figure 4.3. The first, creates a pipeline using the computer vision algorithms, that is only executed when the application needs to analyze images. While the second service is ran periodically, in the background, to verify the server for new searches and execute the work provided by the server.

The computer vision service starts by executing the ViolaJones algorithm over a list of group photos that it receives as input, inserted 2 by 2 to keep the resource usage to a minimum, identifying the faces present in each photo. After the algorithm finishes, a cropper creates temporary image files, each with a person's face originated from the group photos, with information from the original photo.

After the cropper completes its task, a comparison model with the missing person's face is directly loaded into the eigenfaces algorithm. Subsequently, the images created by the cropper are then enqueued for analysis by the algorithm. In the end the algorithm returns a list with an identification for each image inputted.

The work distribution service implements the logic layer request methods. Regularly it inquiries the server, by retrieving the list of available searches, which is compared with its local history. When a new search occurs it requests for its session key, storing it in its memory to be utilized for data decryption. After, it requests for the eigenfaces model with the missing person's face to be used by the computer vision algorithm. The service then verifies the gallery's size to upload the excess to the server for work distribution among the other devices. When it finishes uploading the images it executes the computer vision service.

After the computer vision service finishes, the service verifies the server once more for images that still need to be analyzed, by requesting the list of images. If there are any present, the service downloads five or the remaining of them, and runs again the computer vision service. Otherwise, it uploads the images where a match occurred and removes from the server the images that were analyzed. A flowchart demonstrating this is shown in Figure 4.3.

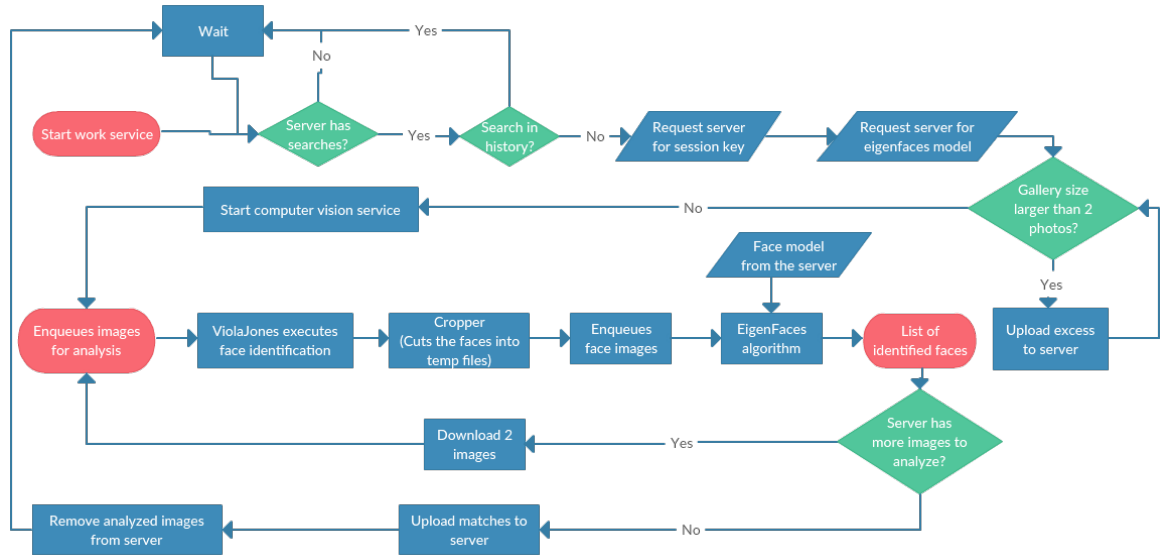


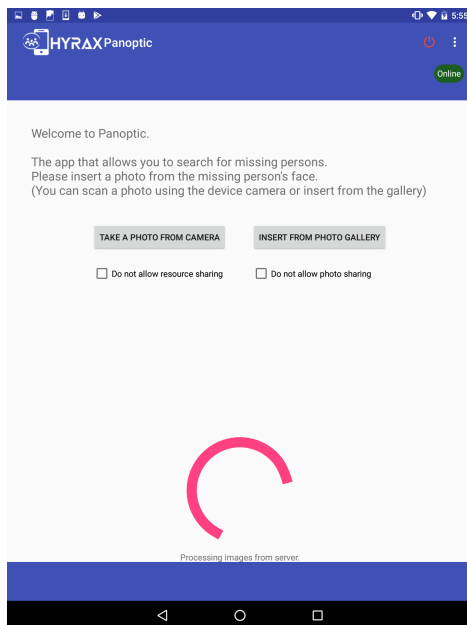
Figure 4.3: Diagram depicting both of the services in a pipeline manner

Application

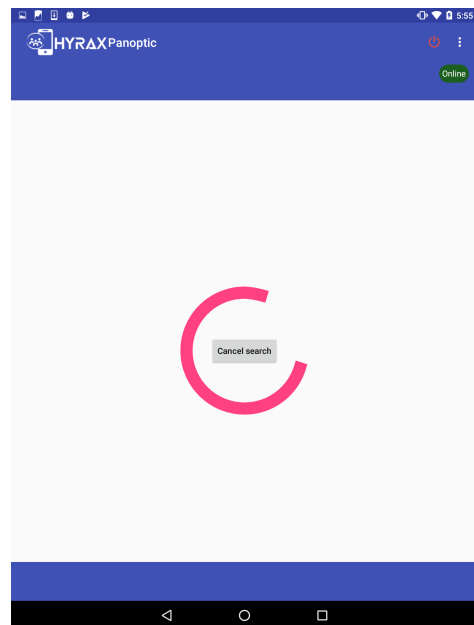
The user interface (UI) is implemented across two activities, the main activity and the result activity. The main activity is constituted by two fragments, the main

fragment and the loading fragment. The main fragment allows the user to issue a search request, through loading the face photo of the missing person from the gallery or by scanning the face from another photo using the device's camera. The fragment has also the options to not allow resource and/or photo sharing, and it also shows if the work distribution service is doing work on behalf of the server. An example can be seen in Figure 4.4a.

After the user loads the photo, the application uploads it to the server for approval of the search. The UI switches the main fragment to the loading fragment, that has the option to cancel the search locally, shown in Figure 4.4b. When the search finishes, the results are shown in the result activity.



(a) Application doing work from the server



(b) Application executing a search

Figure 4.4: Layout from the application

4.3 Summary

In this chapter it was possible to assess the implementation process of the system. In the end it was possible to verify that there were other technologies that could be used, but due to the circumstances and limitations of the architecture, we had to bear with our decisions.

Chapter 5

Evaluation

The current chapter presents the experiment executed to attest the resource consumption of the system. In the first section, the experimental setup is described, as well as the different system configurations used to test the application. In section 5.2 the outcome for each analyzed resource is shown with a short discussion of the results.

5.1 Experimental Setup

To verify our system, we have tested the resource consumption of the application on a mobile device and overall performance from an infrastructural standpoint (traffic used). To carry out the experiment, eight Google Nexus tablets running Android 6.0 with 2 GB of RAM and a dual-core 2.3 GHz CPU with a 6700 mAh battery were used, with an ASUS RT-AC3200 router, that supported 802.11n ac, and a JETTY server installed on a server running Manjaro Linux 17.0.4 with 8 GB of RAM and a quad-core 2.60 GHz CPU. An isolated network was created by connecting the server and the tablets to the router, to mitigate the interference of external communication.

To evaluate the behavior of the system in different situations, five different configurations were envisioned. The first setup, shown in Appendice B in Figure B.1, was a tablet connected to the server to confirm if a single device could carry out the image analysis without any work distribution, and to verify if the footprint left by the system was too demanding on the mobile device.

The second setup, depicted in Appendice B in Figure B.2 consisted in four tablets connected to the server. This configuration allowed us to observe the amount of resources

used in a distributed environment. The values extracted from this configuration were also used to compare with the values of the first configuration. Additionally it gave us a perspective on the quantity of data transferred and received.

The third installment, displayed in Appendice B in Figure B.3 consisted in four tablets, three tablets connected to a fourth, and the fourth tablet, working as a GO, connected to the server, providing a stable connection to the web and a cache service. This configuration was intended to corroborate the assumption that a centralized connection, with a cache service embedded, would relieve the connection of repeated requests.

The fourth and fifth configurations, shown in Appendice B in Figures B.4 and B.5 respectively, were equal to the second and the third respectively, except in the quantity of devices. These configurations were set with eight devices with the purpose of proving that with more devices available, the less demanding the work becomes for each device. The constitution of groups in this architecture was limited to three connections per group to avoid an overcrowd of connections in the GO. Attending to this rule a second group was created in the fifth configuration.

To test the application, three datasets with different sizes were used. Each dataset was divided among the mobile devices that participated in each configuration, with the small dataset having 15 photos, the medium 29 photos and the large dataset 58 photos. For each configuration, we executed ten experiments per dataset while using the Trepn Profiler [37] to observe the resources used.

During every experimental run, we sampled several resources, namely, the CPU utilization, the RAM usage, the energy consumption and the data received and transferred to the server. Another resource that was taken into account was the execution time, due to the QoS and considering the test case of the missing child.

In addition to these configurations, we also deployed the face detection and recognition service to a public cloud, the Google Cloud platform [38], in order to retrieve the execution time. The certified entity receives the photos from all the devices and offloads them to the cloud for analysis. After the work is done the results are sent back to the certified entity.

Using the cloud service for computation helped us understand the differences of using an edge-cloud system, in terms of execution time, besides of creating a baseline for comparison.

Dataset

The dataset used to test the application had very strict rules. The normal procedure to test two different algorithms is by using two distinct datasets to test each algorithm in separate. The datasets that are used to test face detection algorithms usually are constituted by photos of groups of people, while the dataset for face recognition has several photos of a person's face, to train the algorithm and some photos from the same person to test it afterwards. However, this would not be useful to test the application.

The preferred dataset to assess the application needed to have photos from groups of people, and at the same time have faces from some of the people of those groups to train the recognition algorithm. With these definitions we conducted a survey on the Face Recognition Homepage [39]. The datasets presented in the Homepage were constituted mainly by faces of person in grayscale images. Each dataset was divided into two parts, the train part where every face was identified and the test part which had no identification.

Other datasets included faces with different face expressions and some of them had also different light exposure. In the end a total of 44 datasets were analyzed, with none of them having the desired characteristics aforementioned. After finishing the survey we found out that none of the available datasets would meet our requirements. To overcome this situation we decided to create a new dataset.

5.2 Evaluation

In this section we present the values extracted from the various experiments that were conducted. The results shown in each table represents the median of the ten experiments ran on each dataset, with the corresponding 95% confidence interval. Overall we can state, by observing the results, that the application performance excels with more devices in the edge-cloud, and that the creation of an edge-cloud benefits the infrastructure, by lowering the amount of repeated requests using the cache system.

In Figure 5.1 we demonstrate the execution time of the application, with a more detailed view in Table C.1. This measure is important, because the QoS is one of the top concerns when developing new software, and in our case study it is critical to take the least time possible. This can make the difference between the disappearance of a person or being lost for brief moments.

As seen in Figure 5.1, there is a significant difference between running the application

with one device than using several at the same time. It is also possible to notice a difference in time between the edge-cloud configurations and the configurations directly connected to the server.

Regarding the size of the configurations, this affects directly the execution time, which can be noticed in the medium dataset and in the large dataset. While in the small dataset, it is possible to verify that all configurations, with the exception of the 1 device configuration, spent the same time to analyze the dataset. This occurred as a result of the dataset being constituted with 15 photos and being evenly distributed among the devices, both in the 4 devices as in the 8 devices configurations, thus not requiring the work distribution.

Despite the size of the datasets doubled at each step, this was not reflected in the execution time. The medium dataset took almost the same time as the small dataset, on account of the amount of faces detected through the detection algorithm. The output of the face detector dictates the time spent by the recognition algorithm, in other words, more faces detected more time spent to identify each of them.

Lastly, while the public cloud takes less time to do the computation, as expected, it is possible to assess a lower execution time with the incremental introduction of nodes on the edge-cloud, specially when comparing the time taken to analyze the large dataset by the 8 devices configuration.

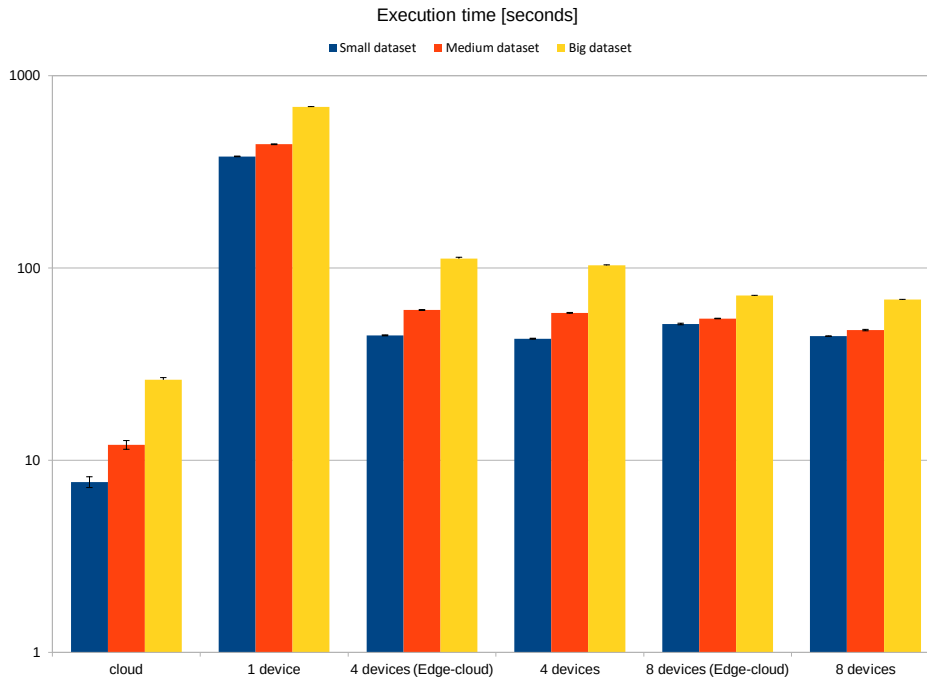


Figure 5.1: Depiction of the execution time used to process each dataset

Table 5.1 presents the memory usage during the experiments. It is possible to verify that using the edge-cloud setup usually needs more memory than the directly connected, because the group owner caches in all the responses that are received from the server. This trait translates into more memory used by that device. Despite this, the difference between the configurations can be disregarded because the message size does not surpass 1 MB.

Memory usage [KB]			
	Small dataset	Medium dataset	Large dataset
1 device	1813746.50 ± 1608.99	1776540.50 ± 2499.23	1779056.07 ± 6376.21
4 devices (Edge-cloud)	1809581.30 ± 520.12	1816682.06 ± 1017.01	1803380.50 ± 2008.33
4 devices	1802020.03 ± 660.61	1688778.50 ± 8513.24	1798262.32 ± 5471.08
8 devices (Edge-cloud)	1806153.09 ± 3037.17	1790949.32 ± 1441.14	1778155.22 ± 8247.05
8 devices	1801698.50 ± 1238.77	1772711.50 ± 2520.74	1792232.65 ± 1630.48

Table 5.1: Memory used by the application during execution time

One of the limitations of the mobile device is the capacity of its battery, thus minimizing energy usage is paramount. A energy consumption comparison is shown in Figure 5.2, with a more detailed overview in Table C.2. We were concerned that running computer vision algorithms on a mobile device would deplete the battery in a way that would render our approach impractically for current generation of mobile devices.

For example, we experienced problems with battery depletion while testing with only one mobile device. The device had to be fully charged before changing to a bigger dataset, otherwise it would completely drain its battery. However, by distributing work among the other devices, we managed to lower amount of energy required to execute the search by a single device (at the cost of more energy being used by the other devices). In Figure 5.2 we can view the energy consumption during different configurations and different datasets. At first glance, we can view that the energy spent by 1 device is fair, compared to the other configurations. However, this value must be linked with the execution time shown in Figure 5.1 before being evaluated.

Through comparison between the times shown in Figure 5.1 and the energy consumption in Figure 5.2, we can assert that the configuration using 1 device, will deplete most of its battery to complete the search. In more extreme cases it will run out of energy before finishing the search. Also, we can observe that duplicating the amount

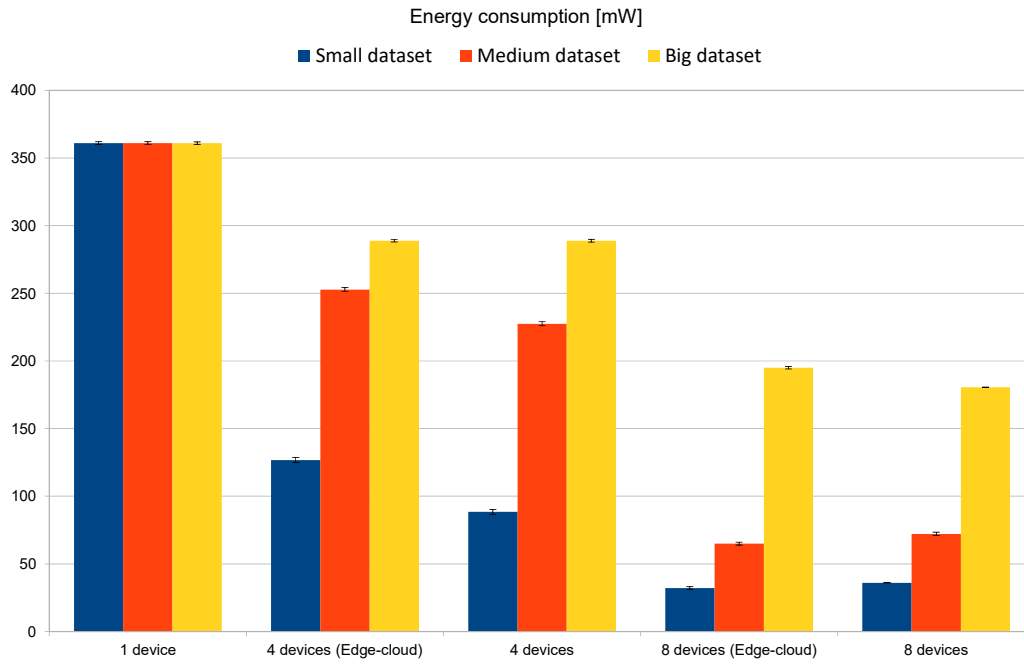


Figure 5.2: Chart showing the power consumption values

of devices to distribute the workload, lowers the amount of energy spent, almost by a factor of 2, again for a single device.

Another remark, is that in an edge-cloud configuration the energy used is generally superior to the direct connected configuration. The edge-cloud configurations spend more energy than the directly connected, because the group owner needs more energy to enable the connection between the devices and to maintain the connection with the external network.

Another resource analyzed, the CPU load, shows a great progress through workload distribution, this can be visualized by analyzing the Figure 5.3 and the Table C.3. The single device is capable of executing the workload but requires a great amount of the available CPU capacity. This can be sensed by the user if he tries to do other tasks in the device at the same time.

By analyzing the results taken from the experiments we can confirm that by increasing the amount of available devices, the load carried by the CPU will diminish. This is supported by the values shown in Table C.3, through analysis of the values between the configurations with 4 devices and the configurations with 8 devices.

Taking a closer look on the retrieved values, it is possible to view that the variation between the different datasets is small. This phenomenon occurs because the detection

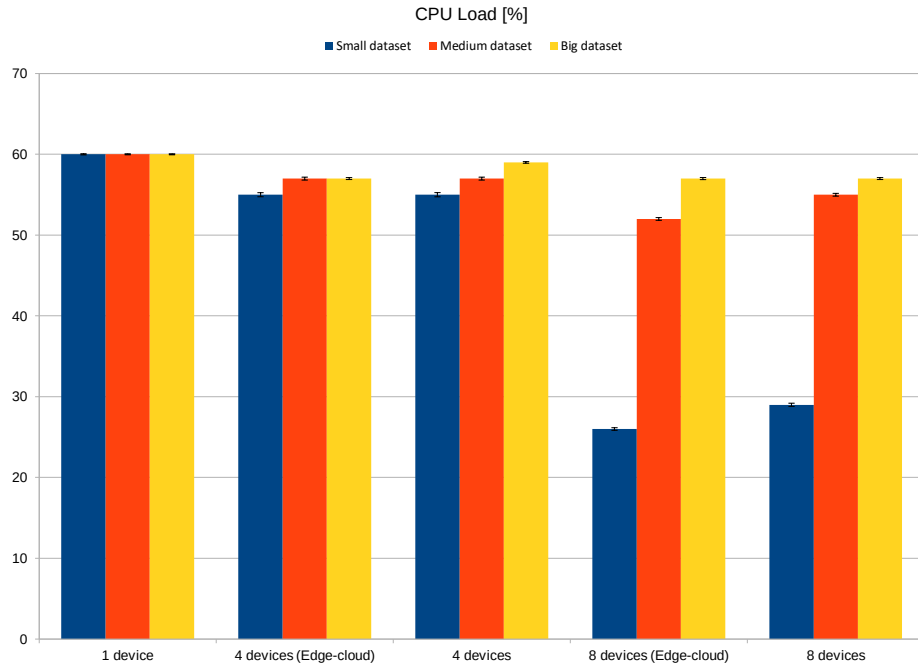


Figure 5.3: Chart showing the CPU load in each configuration for every dataset

algorithm was defined to accept two photos for analysis at a time, which does not differentiate in the CPU load used. The variation that is seen between the datasets, happens because of the recognition algorithm.

Despite inserting two photos at a time, it is not possible to control the amount of faces detected. This has a direct consequence on the CPU load used by the recognition algorithm. It is possible to visualize this in both of the 8 devices configuration, from the medium dataset to the large dataset. In those datasets there was a more accentuated increase of over 10%. This is less noticeable in the other configurations, because in those experiments, it took much more time to analyze the photos, allowing the CPU of the devices to stabilize.

The procedure of inserting 2 photos at a time, safeguards the devices by keeping to a minimum the use of available resources. Lastly, we can verify that the CPU load in the 4 devices was lowered almost by half in the 8 devices configuration, which can contribute to a lower power consumption from the device.

Table 5.2 examines the quantity of bytes received and transmitted from the server. With this verification we wanted to authenticate the difference between using a group owner with a cache system versus a direct connection to the server. We believed that caching in repeated responses would benefit the connection because it would remove duplicated requests, this can also be visualized in Figure 5.4 where it is shown a chart

of the transmitted data to the devices. For example, in a stadium where there is a large crowd trying to access the network. This could be a cheaper solution instead of increasing the local infrastructure.

Network data transmission [MB]			
	Small dataset	Medium dataset	Large dataset
1 device	R: 3.52 ± 0.01 /T: 5.99 ± 0.04	R: 6.44 ± 0.09 /T: 9.08 ± 0.01	R: 14.35 ± 0.01 /T: 17.18 ± 0.03
4 devices (Edge-cloud)	R: 1.15 ± 0.03 /T: 3.12 ± 0.94	R: 3.18 ± 0.12 /T: 4.96 ± 0.17	R: 14.42 ± 0.21 /T: 13.66 ± 0.16
4 devices	R: 0.74 ± 0.01 /T: 6.23 ± 0.03	R: 3.26 ± 0.01 /T: 8.54 ± 0.06	R: 13.92 ± 0.21 /T: 16.95 ± 0.16
8 devices (Edge-cloud)	R: 0.98 ± 0.03 /T: 7.85 ± 0.84	R: 1.59 ± 0.03 /T: 8.18 ± 0.40	R: 9.61 ± 0.10 /T: 15.04 ± 1.50
8 devices	R: 1.04 ± 0.04 /T: 11.32 ± 0.13	R: 1.68 ± 0.02 /T: 11.72 ± 0.08	R: 9.62 ± 0.07 /T: 17.50 ± 0.17

Table 5.2: Quantity of Megabytes[MB] received by the server (R) and quantity transmitted (T) back to the devices

In the results shown in Figure 5.4 we can visualize that advantage. Both of the edge-cloud configurations surpass the direct configuration to the server. The server transmits less data to the devices due to the group owner’s cache.

A further analysis to the table, shows that the received data by the server is lower in all configurations compared to the 1 device configuration. This is a direct result of increasing the number of devices and distribute evenly the experimental datasets between them. In other words, less photos are sent to the server for work distribution.

Another remark from the network data transmission, is the variation from the transmitted data, between the directly connected and the edge-clouds configurations. As shown in table, it is a small variation, that never surpasses 3 MB, because the cache system was used only to respond to requests for the eigenfaces model.

In every experiment with an edge-cloud configuration, the only request that was repeated was the model request, because it was common for all devices. The lack of other type of repeated requests can be explained, considering the fact that all image requests that passed through the GO were new, and therefore were not in cache.

For the system to be used to respond to repeated image requests, the experiments would require to simulate node failure, for other devices to request the same images that were assigned to that node. But since our objective, to prove the cache system, was achieved by the model request, we deemed unnecessary to simulate node failure.

Lastly, we noticed the variance between the medium and the large dataset was considerable. Such occurrence can be justified by the size of the different datasets. The size of the medium dataset(29 photos) was not substantial enough and only a few

photos were sent to the server, for distribution. Every device was capable of executing the majority of their gallery. Unlike the large dataset (59 photos) where all the devices had to dispatch most of their photos to the server, thus increasing the data traffic of the network.

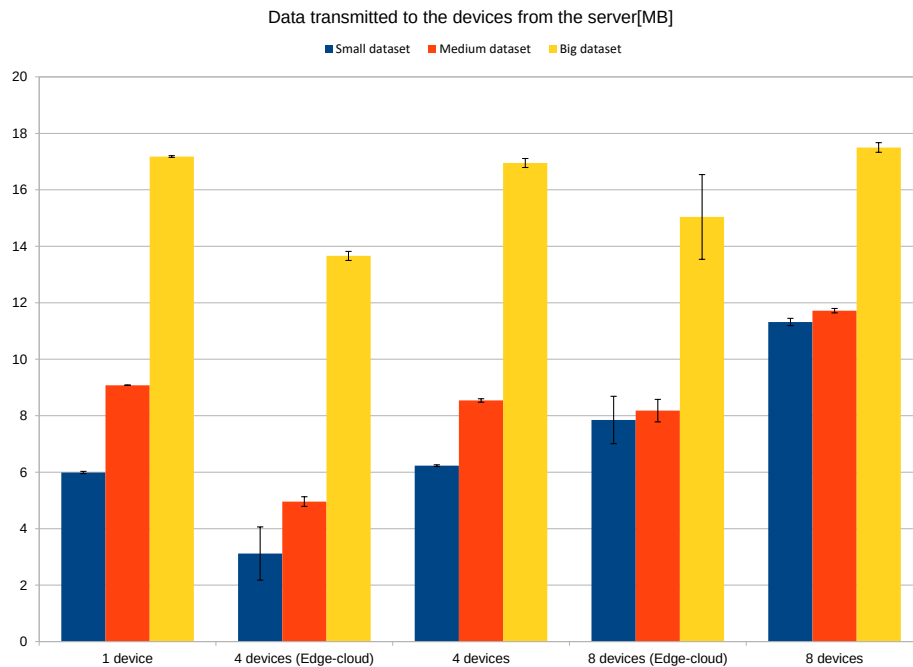


Figure 5.4: Chart depicting the amount of data transmitted to the devices

5.3 Summary

In this chapter it was possible to visualize the resources used by the system. As shown in the results there are advantages in using an edge-cloud configuration. The system can complement the current infrastructure in crowded places, by diminishing the amount of data transferred. Another advantage is releasing the CPU load, by distributing the work between the other devices, therefore lowering the amount of energy spent by the device.

Chapter 6

Conclusion

6.1 Discussion

In this thesis we presented the system of edge-cloud combined with computer vision algorithms and security techniques. We showed that despite these algorithms withdraw a substantial amount of resources it is possible to avoid energy depletion on a single device with proper distribution . By also designing this architecture with a security layer, for enhancing privacy, we shown that it is possible to protect data despite the associated overhead.

With this approach it was possible to answer some questions of investigation, for example, if mobile devices are capable of running heavier algorithms, in this case computer vision algorithms, in a distributed environment and the quantity of energy needed to run these algorithms and if this decreases with the escalation of devices.

With the results extracted from the experiments, we showed that it is possible to equip mobile devices with resource demanding algorithms in a privacy-preserving manner. It is also possible to justify that deployment of edge-clouds is a viable complement to the current network architecture. By caching in most of the requests helps the local infrastructure to reduce the data flow that passes through.

The proliferation of mobile devices in urban regions combined with the dawning of smart cities [40, 41] indicates that there is a chance for edge-cloud systems and applications to thrive. This will lead to a lesser investment on new infrastructures to accommodate the extra amount of data.

6.2 Future Work

For future work, considering the results we had and the above-mentioned case study, we plan in testing this architecture with an Intel SGX to verify the difference between distributing work to the connected devices and offloading to a trusted public cloud. By deploying the same computer vision algorithms in the trusted server instead of running the image analysis in the devices, we intend to verify the advantages and disadvantages of each architecture.

Another possibility that we are also considering is the best form to combine these architectures, in order to achieve a hybrid setup with the best of both worlds. This possibility must also be tested against the other architectures to verify if it is a possible solution and to adjust it accordingly.

Bibliography

- [1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.
- [2] M.-G. Avram, “Advantages and challenges of adopting cloud computing from an enterprise perspective,” *Procedia Technology*, vol. 12, pp. 529–534, 2014.
- [3] A. Acquisti, “The economics of personal data and the economics of privacy,” 2010.
- [4] E. Acosta, L. Torres, A. Albiol, and E. Delp, “An automatic face detection and recognition system for video indexing applications,” in *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 4, May 2002, pp. 3644–3647.
- [5] J. M. Fromholz, “The european union data privacy directive,” *Berk. Tech. LJ*, vol. 15, p. 461, 2000.
- [6] “GDPR new regulation,” http://ec.europa.eu/justice/data-protection/reform/files/regulation_oj_en.pdf.
- [7] F. Bélanger and R. E. Crossler, “Privacy in the digital age: a review of information privacy research in information systems,” *MIS quarterly*, vol. 35, no. 4, pp. 1017–1042, 2011.
- [8] J. Rodrigues, J. Silva, R. Martins, L. Lopes, U. Drolia, P. Narasimhan, and F. Silva, “Benchmarking wireless protocols for feasibility in supporting crowdsourced mobile computing,” in *Distributed Applications and Interoperable Systems*. Springer, 2016, pp. 96–108.
- [9] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, “The case for vm-based cloudlets in mobile computing,” *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

- [10] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, “Cachier: Edge-caching for recognition applications,” *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, Jun 2017.
- [11] K. Habak, M. Ammar, K. A. Harras, and E. Zegura, “Femto clouds: Leveraging mobile devices to provide cloud service at the edge,” in *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*. IEEE, 2015, pp. 9–16.
- [12] A. Ahmed and E. Ahmed, “A survey on mobile edge computing,” in *Intelligent Systems and Control (ISCO), 2016 10th International Conference on*. IEEE, 2016, pp. 1–8.
- [13] N. Takahashi, H. Tanaka, and R. Kawamura, “Analysis of process assignment in multi-tier mobile cloud computing and application to edge accelerated web browsing,” in *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2015 3rd IEEE International Conference on*. IEEE, 2015, pp. 233–234.
- [14] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, *Mobile Edge Computing A key technology towards 5G*. ETSI, 2015.
- [15] U. Drolia, R. Martins, J. Tan, A. Chheda, M. Sanghavi, R. Gandhi, and P. Narasimhan, “The case for mobile edge-clouds,” in *2013 IEEE 10th International Conference on Ubiquitous Intelligence and Computing and 2013 IEEE 10th International Conference on Autonomic and Trusted Computing*, Dec 2013, pp. 209–215.
- [16] J. A. Silva, H. Paulino, and J. Lourenço, “Crowdsourcing mobile devices to provide storage in edge-clouds,” *16th International Conference on Distributed Computing and Networking (ICDCN 2015)*, January 2015.
- [17] J. Luo, J.-P. Hubaux, and P. T. Eugster, “Pan: Providing reliable storage in mobile ad hoc networks with probabilistic quorum systems,” in *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*. ACM, 2003, pp. 1–12.
- [18] S. Abdelwahab, B. Hamdaoui, M. Guizani, and T. Znati, “Replisom: Disciplined tiny memory replication for massive iot devices in lte edge cloud,” *IEEE Internet of Things Journal*, vol. 3, no. 3, pp. 327–338, 2016.
- [19] W. Gao, “Opportunistic peer-to-peer mobile cloud computing at the tactical edge,” in *Military Communications Conference (MILCOM), 2014 IEEE*. IEEE, 2014, pp. 1614–1620.

- [20] K. Shilton, “Four billion little brothers?: Privacy, mobile phones, and ubiquitous data collection,” *Communications of the ACM*, vol. 52, no. 11, pp. 48–53, November 2009.
- [21] M. Langheinrich, *A Privacy Awareness System for Ubiquitous Computing Environments*, October 2002, pp. 237–245.
- [22] J. L. Boyles, A. Smith, and M. Madden, “Privacy and data management on mobile devices,” *Pew Internet & American Life Project*, vol. 4, 2012.
- [23] H. Yang, H. Luo, F. Ye, S. Lu, and L. Zhang, “Security in mobile ad hoc networks: challenges and solutions,” *IEEE wireless communications*, vol. 11, no. 1, pp. 38–47, 2004.
- [24] H. Deng, W. Li, and D. P. Agrawal, “Routing security in wireless ad hoc networks,” *IEEE Communications magazine*, vol. 40, no. 10, pp. 70–75, 2002.
- [25] H. Yang, J. Shu, X. Meng, and S. Lu, “Scan: self-organized network-layer security in mobile ad hoc networks,” *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 2, pp. 261–273, 2006.
- [26] P. Viola and M. J. Jones, “Robust real-time face detection. (International),” *Journal of Computer Vision*, no. 57, p. 137, 2004.
- [27] Y. Freund and R. E. Schapire, “A decision-theoretic generalization of on-line learning and an application to boosting,” *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, Aug. 1997.
- [28] T. Wu and A. Toet, “Speed-up Template Matching through Integral Imagebased Weak Classifiers. (International),” *Journal of Pattern Recognition Research*, no. 1, pp. 1–12, 2014.
- [29] J. Kovac, P. Peer, and F. Solina, “Human skin colour clustering for face detection,” in *EUROCON 2003. Computer as a Tool. The IEEE Region 8*. IEEE, 10 2003.
- [30] M. Turk and A. Pentland, “Eigenfaces for recognition,” *J. Cognitive Neuroscience*, vol. 3, no. 1, pp. 71–86, January 1991.
- [31] R. A. Fisher, “The use of multiple measurements in taxonomic problems,” *Annals of Eugenics*, vol. 7, no. 2, pp. 179–188, 1936.
- [32] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld, “Face recognition: A literature survey,” vol. 35, no. 4, pp. 399–458, Dec. 2003.

- [33] C. Lu and X. Tang, “Surpassing human-level face verification performance on lfw with gaussianface,” *arXiv preprint arXiv:1404.3840*, 2014.
- [34] M. Butler, “Android: Changing the mobile landscape,” *IEEE Pervasive Computing*, vol. 10, no. 1, pp. 4–7, 2011.
- [35] “Jetty homepage,” <https://www.eclipse.org/jetty/>.
- [36] N. Pérez, S. Faria, and M. Coimbra, “Understanding the energy saving potential of smart scale selection in the viola and jones facial detection algorithm,” in *Proceedings of the 10th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC 2017) - Volume 2: BIOIMAGING, Porto, Portugal, February 21-23, 2017.*, 2017, pp. 122–127.
- [37] “Trepn profiler,” <https://developer.qualcomm.com/software/trepn-power-profiler>.
- [38] “Google Cloud platform,” <https://cloud.google.com/>.
- [39] “Facial recognition homepage,” <http://www.face-rec.org/>, 2005.
- [40] M. Batty, “Big data, smart cities and city planning,” *Dialogues in Human Geography*, vol. 3, no. 3, pp. 274–279, 2013.
- [41] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, “Sensing as a service model for smart cities supported by internet of things,” *Transactions on Emerging Telecommunications Technologies*, vol. 25, no. 1, pp. 81–93, 2014.

Appendices

Appendix A

Content of the .proto file

The present appendix shows the content of the .proto file used to represent the structured data for serialization of the communication messages.

```
syntax = "proto3";

package communication;

option java_outer_classname = "MessageProto";

message EncryptedMessage{

    enum Type {
        ERROR = 0;
        KEY = 1;
        PING = 2;
        FACE_LIST = 3;
        REAL_LIST = 4;
        IMAGE_LIST = 5;
        REMOVE_IMAGE = 6;
        DOWNLOAD = 7;
        UPLOAD = 8;
    }
    Type type = 1;

    // Server information
    Error error = 2;

    Ping ping = 3;
```

```
    bytes key = 4;

    repeated string list = 5;

    // file management

    string label = 6;

    string fileName = 7;

    //timestamp used for the cache
    int64 timestamp = 8;

    bytes signature = 9;

    bytes cipheredMessage = 10;

    string androidId = 11;
}

//ping message meaning to know if a server is alive
message Ping {
    int64 client_ts = 1; //source timestamp (mandatory)
}

//generic file object representation
message Image {
    enum Type{
        FACE = 0;
        IMAGE = 1;
        MATCH = 2;
    }

    //image location
    string location = 1;

    //image size
    int32 length = 2;

    //Image type
```

```
Type type = 3;

//size must be under 1 MB googles' orders
bytes src = 4;
}

//error response
message Error {
    string error_message = 1; // error message
}
```


Appendix B

Display of the experiments configurations

This appendix displays the different configurations created to run each experiment.

Configuration: 1 device

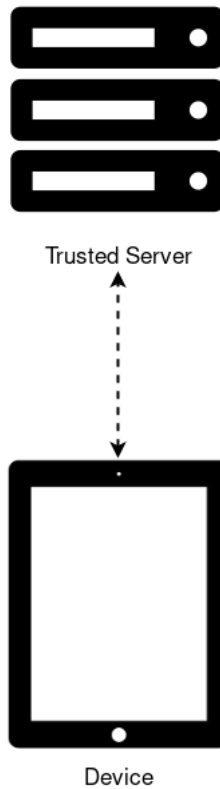


Figure B.1: Configuration with 1 device

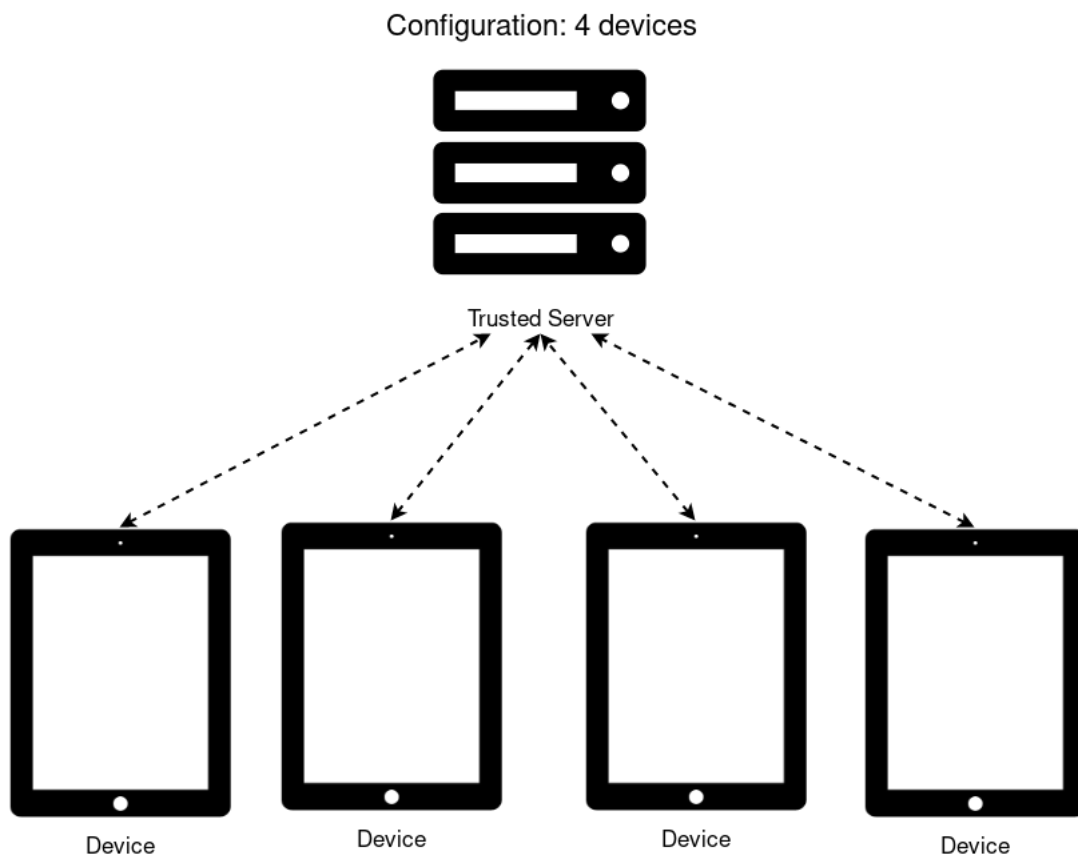


Figure B.2: Configuration with 4 devices

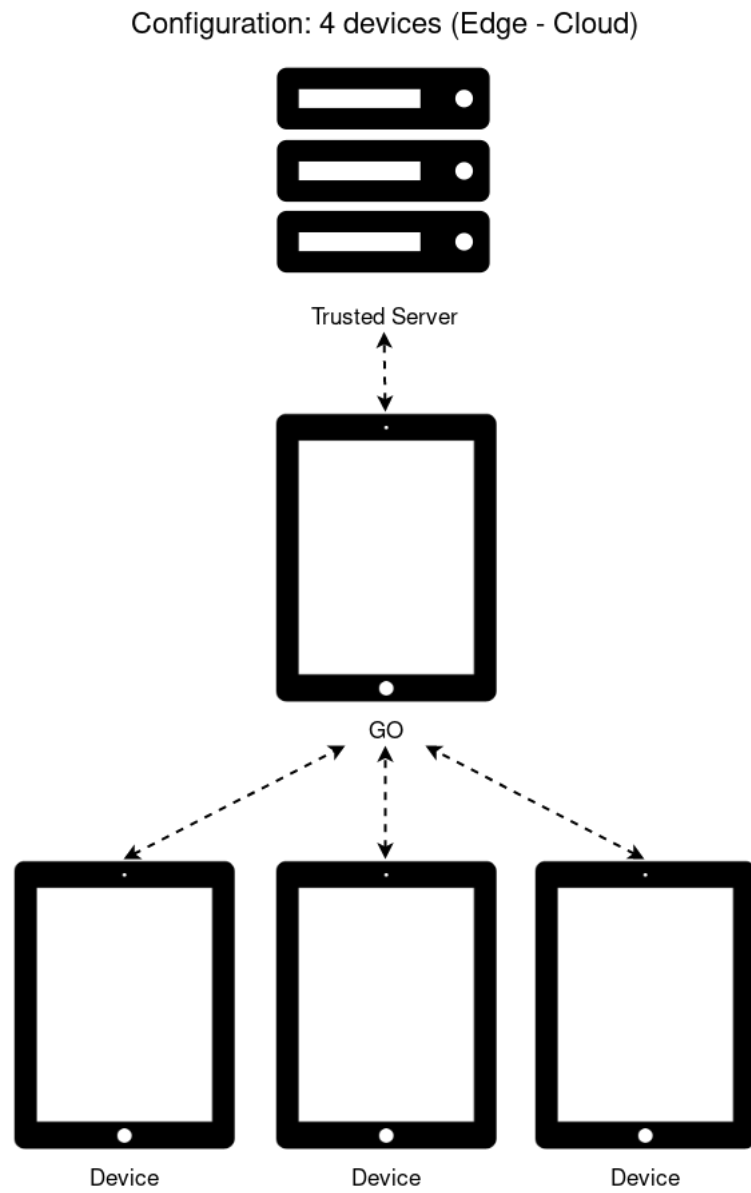


Figure B.3: Configuration with 4 devices set as an Edge-cloud

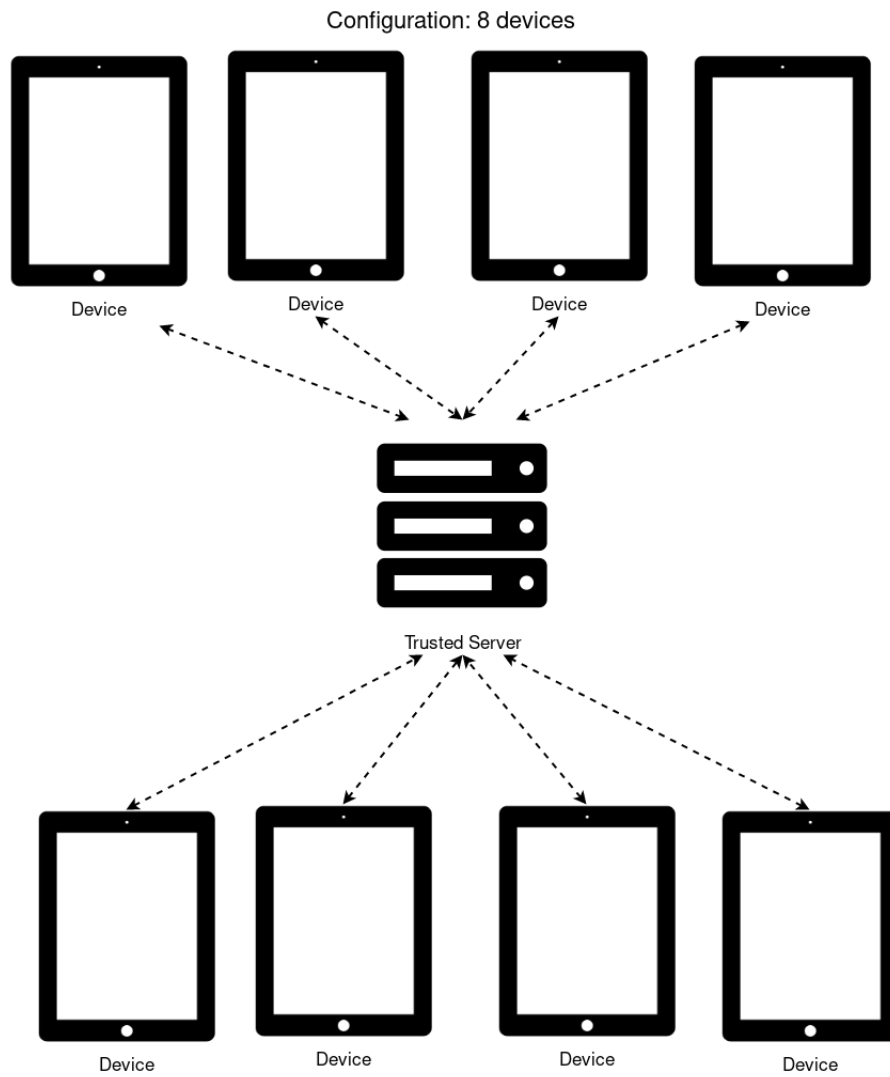


Figure B.4: Configuration with 8 devices

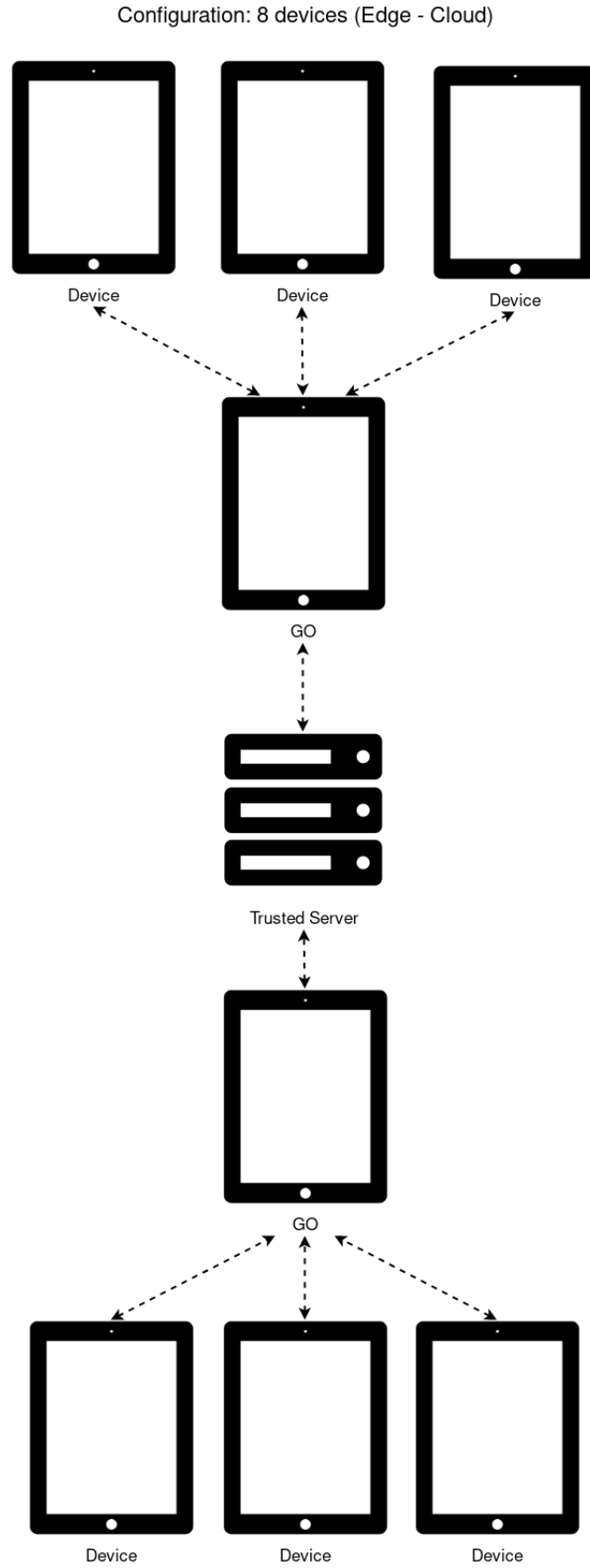


Figure B.5: Configuration with 8 devices set as an Edge-cloud

Appendix C

Tables with the results from the experiments

This appendix gives a more detailed view regarding the results over the execution time, the energy consumption and the CPU load spent by the application.

Table C.1 demonstrates the values regarding the execution time for each configuration. Table C.2 provides a more detailed perspective about the energy spent by the system. Table C.3 shows a detailed overview of the results obtained from the CPU load used in each configuration.

Execution Time [seconds]			
	Small dataset	Medium dataset	Large dataset
cloud	7.70 ± 0.50	12.02 ± 0.63	26.17 ± 0.72
1 device	380.28 ± 1.54	439.54 ± 1.72	688.14 ± 2.55
4 devices (Edge-cloud)	44.56 ± 0.23	60.40 ± 0.28	111.80 ± 1.86
4 devices	42.85 ± 0.23	58.39 ± 0.28	103.31 ± 0.39
8 devices (Edge-cloud)	51.09 ± 0.49	54.56 ± 0.19	71.80 ± 0.22
8 devices	44.14 ± 0.16	47.50 ± 0.41	68.41 ± 0.22

Table C.1: Execution time taken to process each dataset on different configurations

Battery energy[mW]			
	Small dataset	Medium dataset	Large dataset
1 device	361.08 ± 0.94	361.08 ± 0.91	361.08 ± 0.90
4 devices (Edge-cloud)	126.77 ± 1.82	252.76 ± 1.46	288.86 ± 0.97
4 devices	88.45 ± 1.84	227.48 ± 1.49	288.86 ± 0.98
8 devices (Edge-cloud)	32.13 ± 0.97	64.99 ± 1.09	194.98 ± 0.95
8 devices	36.11 ± 0.20	72.21 ± 1.18	180.54 ± 0.12

Table C.2: Energy used per device for each dataset

CPU Load [%]			
	Small dataset	Medium dataset	Large dataset
1 device	60.15 ± 0.07	60.13 ± 0.06	60.24 ± 0.06
4 devices (Edge-cloud)	55.42 ± 0.25	57.08 ± 0.18	60.02 ± 0.12
4 devices	55.03 ± 0.26	57.01 ± 0.18	59.05 ± 0.10
8 devices (Edge-cloud)	26.16 ± 0.16	52.23 ± 0.17	57.12 ± 0.13
8 devices	29.23 ± 0.20	55.05 ± 0.17	57.11 ± 0.12

Table C.3: CPU Load spent to process the workload