André Borges Sampaio

Licenciatura

Resource Sharing and Search in Partially Decentralized Mobile Networks

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática

Orientador : João Carlos Antunes Leitão,
Professor Auxiliar Convidado,
NOVA University of Lisbon

Júri:

Presidente:	Maria Cecília Gomes
Arguente:	Joao Nuno Silva
Vogal:	João Carlos Antunes Leitão

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

SETEMBRO, 2016

Resource Sharing and Search in Partially Decentralized Mobile Networks

To my family, salsa and hortelãs

Acknowledgements

I would like to thank my adviser and mentor, Dr.João Leitão, for all the support, friendly words and hard work throughout this process. I would like to also give a special thanks to Dr. Nuno Preguiça, for all the help and suggestions. This work was partially supported by FCT/MCTES through the project Hyrax (CMUP-ERI/FIA/0048/2013) as well as through the NOVA LINCS strategic project (UID/CEC/04516/2013).

Finally, I would like to leave a big "thank you" to my family, friends, and special one that kept me inspired at all times.

Abstract

Media sharing between smart-devices, such as smartphones and tablets, is a common habit between users. This happens typically through third party services such as social networks. In order to achieve this, users usually rely on an Internet connection, either through infrastructure or by relying on 3G/4G technology.

Under the mobile paradigm, direct connections between any pair of mobile devices are not always possible to establish and infrastructured networks solutions are not always available, especially on highly populated events such as concerts or medium to large social gatherings.

The goal of this work is to explore file-sharing alternatives, that are independent from an Internet connection as much as possible, focusing on media sharing and search in the context of medium sized to large sized social gathering in mobile networks. We envision a case study application whose goal is to build and maintain a distributed image gallery using an hybrid (partially decentralized) edge computing network architecture. In which mobile devices communicate in a peer-to-peer fashion, except when such is not possible, or when resorting to an available infrastructure is a better option. The gallery can then be searched using sophisticated querying techniques such as facial recognition, that enables querying the network for photos that contain specific facial characteristics.

We propose and evaluate experimentally two alternative partially decentralized architectures to support such a case study that explore different degrees of direct interaction between devices.

Experimental evaluation shows that these alternative architectures have acceptable performance and cost, making them viable alternatives to support mobile applications in the edge.

Keywords: media sharing; smart-devices; mobile; distributed gallery; partially decentralized; peer-to-peer; facial recognition

Resumo

A partilha de multimédia entre dispositivos móveis, como *smartphones* e *tablets*, é um hábito comum dos utilizadores. Esta partilha é feita tipicamente através de serviços como as redes sociais. De modo a aceder a estes serviços, os utilizadores dependem de uma ligação à *Internet*, seja através de uma infraestrutura ou mediante a tecnologia 3G/4G.

No paradigma móvel, nem sempre é possível estabelecer ligações diretas entre qualquer par de dispositivos. Para além disso, nem sempre é possível aceder a serviços que utilizam redes centralizadas, pois estes por vezes encontram-se indisponíveis. Este último impedimento verifica-se especialmente em eventos altamente populados, por exemplo em concertos ou em grandes reuniões.

Este trabalho pretende explorar soluções alternativas para a partilha e pesquisa de dados multimédia, no contexto das redes móveis. Focando-se em particular em situações que envolvem, grupos de pessoas de média a grande dimensão. Pretende-se utilizar como caso de estudo, uma aplicação cujo propósito é de montar e manter uma galeria distribuída que opera sobre uma rede de arquitetura híbrida (parcialmente descentralizada) que faz uso de computação no extremo. Na qual os dispositivos móveis comunicam diretamente entre si (*peer-to-peer*), exceto quando não é possível estabelecer ligação ou nos casos em que recorrer a uma das infraestruturas disponíveis seja mais vantajoso. A galeria será pesquisável, servindo-se de técnicas sofisticadas de procura, como reconhecimento facial que permite pesquisar a rede por características faciais especificas.

Propomos e avaliámos experimentalmente duas arquiteturas parcialmente descentralizadas, de modo a suportar o caso de estudo como este que explora diferentes graus de interação direta entre dispositivos.

A avaliação experimental mostra que estas arquiteturas têm uma eficiência e custo aceitáveis, o que permite considerar considerá-las como alternativas viáveis o suporte de aplicação que recorrem a computação no extremo.

Palavras-chave: partilha; multimédia; dispositivos móveis; galeria distribuída; arquitetura híbrida; parcialmente descentralizada; computação no extremo; peer-to-peer; reconhecimento facial

Contents

1 Introduction **1**

1.1 Motivation . . . 2
1.2 Main Contributions . . . 2
1.3 Document Organization . . . 3

2 Related Work **5**

2.1 Mobile Computing . . . 5
2.1.1 Mobile Client-Server . . . 6
2.1.2 Mobile Ad-Hoc Networks . . . 7
2.1.3 Routing Protocols . . . 7
2.1.4 Overlay Protocols . . . 8
2.1.5 Discussion . . . 11
2.2 Peer-to-Peer Networks . . . 11
2.2.1 Overlay Networks . . . 12
2.2.2 Ad-hoc Querying . . . 14
2.2.3 Discussion . . . 15
2.3 Edge Computing . . . 15
2.3.1 Mobile Edge Computing . . . 16
2.3.2 Discussion . . . 17
2.4 Security . . . 17
2.4.1 Security in MANETS . . . 18
2.4.2 Attacks . . . 18
2.4.3 Mobile Edge Network Security . . . 19
2.4.4 Discussion . . . 20
2.5 Wireless Communication Technologies . . . 20
2.6 Discussion . . . 21
2.7 Summary . . . 21

3 Aether **23**
3.1 Assumptions . . . 23
3.2 Proposed Solution Overview . . . 24
3.2.1 Communication Model . . . 24
3.2.2 Aether Architecture . . . 25
3.3 Implementation . . . 26
3.3.1 Centralized Solution . . . 26
3.3.2 Aether . . . 27
3.4 Summary . . . 31

4 Aether+ **33**
4.1 Assumptions . . . 33
4.2 Proposed Solution Overview . . . 33
4.3 Architecture . . . 35
4.4 Implementation . . . 36
4.4.1 Peer-to-Peer Communication . . . 37
4.4.2 Fallback Communication . . . 45
4.5 Summary . . . 46

5 Case Study: Distributed Gallery **47**
5.1 Facial Detection and Recognition . . . 48
5.2 User Registration . . . 49
5.3 Querying . . . 49
5.4 File Indexing . . . 51
5.5 Gallery Interface, Functions and Options . . . 53
5.6 Summary . . . 54

6 Evaluation **57**
6.1 Prototype Details . . . 57
6.1.1 Implementation Specifications and Statistics . . . 58
6.2 Experimental Setup . . . 58
6.3 Image Retrieval Comparison . . . 59
6.3.1 Data Transfer . . . 59
6.3.2 Latency . . . 60
6.3.3 Battery Usage . . . 62
6.3.4 Fallback Mode . . . 62
6.4 Query Execution Evaluation . . . 64
6.4.1 Facial Recognition Performance . . . 64
6.4.2 Data Transfer . . . 67
6.4.3 Latency . . . 68
6.4.4 Battery Usage . . . 68
6.5 Summary . . . 69

7 Conclusions **71**
7.1 Future Work 72

List of Figures

2.1 Structure of routing protocols 9
2.2 Overlay Networks 12
2.3 Mobile Edge Computing Architecture 16

3.1 Baseline solution: Fully Centralized 24
3.2 Aether - Peer-to-peer with coordination infrastructure 25
3.3 Aether : components architecture 25
3.4 Aether fallback communication model 30

4.1 Aether+ - Peer-to-peer with fallback infrastructure 34
4.2 Aether+ architecture 35
4.3 Aether+ packet structure 38
4.4 Aether+ fallback communication model 46

5.1 Flandmark usage 48
5.2 Query with the semi-decentralized architecture 50
5.3 Fallback communication model adapted to gallery 51
5.4 Local index update on image reception 52
5.5 Interface Screenshots 53
5.6 Interface Screenshots 55

6.1 Comparison of the data volume exchanged between components 59
6.2 Comparison of the number of packets exchanged between client and server 61
6.3 Comparação de Pacotes IP e Latência 61
6.4 Comparison of the average battery usage 62
6.5 Received and sent bytes in fallback mode 63
6.6 Received and sent packets in fallback mode 64
6.7 Observed Latencies in fallback mode 65
6.8 Average battery usage in fallback mode 66
6.9 Accuracy comparison with different thresholds 66

6.10 Facial recognition process results 67
6.11 Total and recognition process latency 68
6.12 Average energy usage from the receiver and sender perspective 69

List of Tables

2.1 Technologies comparison 20

4.1 Routing table header 37

6.1 Threshold comparison 65
6.2 Received and sent bytes volume 67
6.3 Received and sent packets volume 68
6.4 Average CPU Load on the sender 69

List of Algorithms

1 Upload file - Server side 27
2 Execute query - Server side 27
3 File transfer with connectivity 28
4 Negotation and Data transfer 28
5 Fallback mode - Receiver 30
6 Connect Peer 39
7 Receiver - Routing Table Setup and Update 40
8 Bluetooth Scan 41
9 Connect to establish Bridge 42
10 Packet Routing 43
11 Send Broadcast Message 45
12 Untag user from image 54

1

Introduction

Smart-devices nowadays provide computation, storage, and networking resources on the move. These capabilities can be combined if several independent units communicate with each other forming a distributed network. This has recently been denominated as edge-cloud-computing.

Edge-computing on smartphones takes advantage of the aggregation of available resources from each individual device creating a mobile-edge-cloud that creates an opportunity not only to support file-sharing among users but also to perform distributed computations among the devices, minimizing the dependency of other resources or infrastructures such solutions have the potential for, boosting availability in situations where there is network congestion in data services or when those services are not available. For example, when there is a large number of users in the same location trying to upload photos or videos to share those contents with other users (potentially close by) over centralized social networks at the same time. Media sharing between mobile devices, whether it is a smartphone or a tablet, is a recurrent habit between smart-device users, especially in social gatherings [Ahe+07].

Recent approaches have explored this type of strategy and have studied the potential of edge clouds in media sharing as a novel way of publishing and retrieving multimedia content [Mar09]. This type of approach is aimed at poor or no cellular connectivity scenarios such as battlefield settings or highly populated events while also providing a way of sharing crowd-sourced data and computational power, for instance to perform extraction of features and execute distributed search algorithms over the multimedia content being shared.

In order to substantiate this line of research, previous approaches have modified an existing cloud computing system, Apache Hadoop, to run on Android mobile devices.

The results obtained in their investigation allowed to conclude that the usage of this architecture results in more consistent latencies than uploading and serving files over a remote server [Mar09]. In this thesis we also follow this line of research, albeit focusing on a different type of applications and on a different set of requirements. In particular we focus on, search and retrieval of resources through mobile applications with limited connectivity to infrastructural network and components.

1.1 Motivation

Client-server based architectures that resort to centralized infrastructures present several challenges such as the need to contact to a remote host through an Internet connection, or dealing with service unavailability due to hardware or connection related problems. In contrast, infrastructure-less network architectures such as the one mentioned earlier, address some of these challenges while also leading to new ones such as the inherent complexity of performing distributed searching on a constantly changing network topology [Lun00]. An hybrid partially-decentralized network topology might have the capacity of mitigating issues present in both centralized-based and decentralized architectures by maximizing availability, reducing latency, and being more robust to data services overload or full unavailability of centralized components.

Adopting this hybrid approach, relying both on smart-devices and remote infrastructure elements, can lead to the extraction of benefits of a fully decentralized (i.e, peer-to-peer) network while still enabling one to fallback to a centralized infrastructure in scenarios when peer-to-peer connectivity is not available or not convenient when compared to a solution that resorts to a centralized remote host. Furthermore, the centralized infrastructure can also be leveraged to improve the operation of decentralized mechanisms, such as routing among the peripheral devices, or user authentication, which are hard to tackle in fully decentralized architectures [FLR13].

These characteristics can provide mobile applications a with the fundamental mechanism for collecting and retrieving crowd-sourced data while also offering the potential to reduce user frustration through an improved availability that translates into a more fluid, and thus more satisfying experience.

1.2 Main Contributions

The capabilities of a partially-decentralized network design can empower applications with access to distributed data without the need of an Internet connection, especially useful on file-sharing scenarios such as photos and videos, taking that into consideration, we envisioned a concrete case-study of a crowd-sourced distributed photo gallery that would feature image searching by facial recognition. Such an application would allow users to query the complete set of pictures taken by other users that are attending the same event or gathering. For example, searching for all pictures where you appear while

attending a 100 person birthday party or getting photos from a specific perspective in a football match.

In this work, we present the design, implementation and experimental validation of the case study described above operating on top of three distinct architectures with varied degrees of decentralization as to explore the capabilities of a partially-decentralized mobile architecture that enables the operation of such an application, providing access to distributed crowd-sourced data resources through the design and combination of a set of specific components. In particular, we explore the following aspects:

Partially-decentralized Overlay Network: an hybrid logical network topology designed to logically connect a set of mobile devices that takes into account hardware challenges, such as battery drainage, and the physical location of participants that allows devices to communicate without Internet access.

Distributed Photo Gallery: a crowd-sourced photo gallery populated by media retrieved from nearby devices, which extracts relevant features from this media.

Facial Recognition Querying: a feature that enables searching over a distributed photo gallery using facial recognition techniques that can leverage computation on the edge-mobile-cloud.

The three different architectures that we explored are the following:

Fully Centralized architecture: where all operations are executed through a centralized component. This serves as a control scenario which we employ to study the benefits and trade-offs of the following two architectures:

Aether: is an hybrid protocol, that resorts to direct data transmission between devices, while incorporating a centralized component that orchestrates the device-to-device communication.

Aether+ is also an hybrid protocol, based on Aether, that is fully decentralized but uses the central component as fallback, when there is no connectivity between devices.

1.3 Document Organization

The remainder of the thesis is organized as follows:

Chapter 2 introduces fundamental concepts which are relevant for the context of the contributions presented in the thesis;

Chapter 3 presents and describes Aether, which is the proposed solution that addresses the challenges being tackled on the context of the thesis. This chapter depicts the design, architecture and implementation of this solution;

Chapter 4 presents and describes an improved version of Aether, named Aether+, including design, architecture and implementation;

Chapter 5 presents the evaluation of our system and the results of comparison to the centralized baseline prototypes, also developed in this work;

Chapter 6 concludes this thesis by summarizing the thesis and discussing pointers for future work.

2

Related Work

This dissertation addresses challenges related with the design of decentralized and partially decentralized distributed architectures with emphasis on the support for media sharing, and distributed querying. Hence, several aspects related with these topics have to be addressed. In the following we present a survey of relevant previous works that address challenges related and complementary to the goals of this work.

In Section 2.1 mobile computing both infrastructure-based and infrastructure-less architectures, client-server and ad-hoc respectively, are described and compared.

In Section 2.2 relevant peer-to-peer systems aspects are discussed and compared, focusing mainly on differences on these architectures accordingly with their degree of decentralization also surveying some alternative designs for overlay networks.

In Section 2.3 edge computing, especially the mobile-edge paradigm, is described and discussed.

In Section 2.4 security on mobile computing is discussed, specifically on wireless ad-hoc and edge computing networks.

In Section 2.5 relevant communication technologies are analysed and compared based on previous studies.

2.1 Mobile Computing

Mobile computing distinguishes from the static variant mainly due to the mobility of nodes and the resource constraints associated with mobile resources such as limited battery life and wireless bandwidth and connectivity.

Even tough mobile devices present some resource restrictions, clients still expect the same level of service from applications as with their stationary counterparts. In order

to achieve satisfying service performance and availability, infrastructure-based classical models like the client-server model need to be adapted and extended to fit the specific requirements of this environment [DD08].

2.1.1 Mobile Client-Server

Accordingly to the definition, presented in [JHE99] for client-server information system, a server is defined as any machine that holds a complete copy of one or more databases (i.e, application state). A client is defined as an entity that communicates with the servers in order to interact or operate over the existing data.

The mobile environment, in certain tasks, blends these two roles to compensate for the resource limitations of mobile devices. By performing some client operations on resource-rich servers or by mimicking the functions of a server in a client to cope with unstable connectivity.

The amount of functions that are relocated among these entities to the other role classifies client-server architectures as:

Thin Client: This architecture moves a great part of the application logic and functionality from clients to servers. This type of architecture is suitable for scenarios where the client devices hardware properties do not meet the application requirements.

Full Client: These architectures emulate server functions on client devices, therefore allowing offline usage and minimizing the implications of connectivity uncertainty over the behavior of applications.

The ability to operate in a disconnected environment is used as fallback in intermittent, low bandwidth, high latency, or high expense network cases in which network characteristics have degraded beyond the acceptable usability standards.

Flexible Client-Server: These architectures dynamically redirect and perform application logic on clients and servers. In order to boost performance and availability, it temporarily dims the distinction between mobile devices and stationary hosts [JHE99].

The different approaches described above are used in accordance with the requirements of device hardware and expected connectivity scenarios. For instance, mobile thin client computing is an enabler for the execution of hardware demanding applications. Only requiring clients to display graphics and outputting results through virtual network computing (VNC) technology. A concrete example of one approach to achieve this is:

THINC is a virtual display architecture for thin-client computing that provides high fidelity display and interactive performance in both LAN and WAN environments. Instead of providing a real driver for a particular display hardware, THINC introduces a simple virtual display driver that intercepts drawing commands at the device layer, and sends them over to a client device to display [BKN05].

Another common alternative is to rely on web based applications. Whose logic is fully executed on the server side, and over which the client interacts through a web interface. A concrete and well known example of this is:

Office Online is a collection of office productivity tools, previously available offline, such as a word processor (Word), spreadsheet (Excel), slide show presentation (Powerpoint). That can be used through the Internet via a web browser [Mic16].

Mobile full client applications such as the, photo editing tool Photoshop [**Adobe2016**], which exists for both mobile and desktop platforms, rely on the capabilities of the client, CPU and local storage. By avoiding to resort to a central server and operating locally, they provide a larger set of features and a more fluid user experience (albeit being essentially local applications with few to no distributed aspects).

Mobile flexible client applications use the best of both worlds, by processing locally and using server resources as required. Desktop video games like Diablo 3 [Bli16] implement this type of architecture as they need both thin and full client characteristics to support graphics, and store and provide game content. There are also approaches that use this model over the classic client-server on other fields such as e-commerce [Mah12], and object recognition and tracking with augmented reality [Gam10] in mobile platforms.

2.1.2 Mobile Ad-Hoc Networks

A Mobile Ad-hoc Network (MANET) is a set of wireless mobile hosts dynamically forming a network without the use of an existing network or centralized coordination infrastructure [Tse+02; Su+09]. Due to this, mobile ad-hoc networks are required to address network aspects, one of the most relevant is routing. We now briefly describe some of the routing approaches commonly used in this context.

2.1.3 Routing Protocols

Distributed coordination requires that nodes follow some type of protocol to communicate efficiently. Several protocols have been proposed such as DSDV (Destination Sequenced Distance Vector) and DSR(Dynamic Source Routing), among others [SWS12]. In general, routing protocols can be classified as follows:

Proactive protocols are table driven, in other words, each node maintain a local routing table. This table contains the address of nodes he may communicate with and respective number of hops required to arrive to the destination, as well as (at least) the next hop for reaching that destination. Each entry is tagged with a sequence number created by the destination node. In order to maintain stability from time to time, each node broadcasts and updates its routing table (using informaton received by other nodes).

Reactive protocols focus on reducing overhead by determining routes on demand. The route discovery process, floods the network with RREQ (Route Request) packets, to map the path between source and destination whenever a node needs to communicate with a specific destination.

Hybrid protocols combine both proactive and reactive features in an adaptive way. Applying what type of protocol fits best on each situation.

For completeness, we now provide some examples of proactive, reactive and hybrid protocols:

OLSR is a proactive protocol, that can be seen as an optimization of the pure link-state algorithm[1], in a way that satisfies the requirements of the mobile paradigm. The key feature in this protocol, is that only some nodes forward the broadcast messages during the flooding. Furthermore these nodes, also known as multipoint relays (MRPs), are also the only ones that contain the link state information. This technique substantially reduces message overhead and number of control messages, when compared to the traditional flooding mechanism [TP03].

BATMAN is an optimization of OLSR, on its weak performance on large networks. In BATMAN, nodes do not maintain the full route to the destination, instead each node along the route only maintains the information about the next hop (i.e. the neighboring node), through which the best route can be found [JNA08].

DSR is a reactive protocol, that caches the complete hop-by-hop route between sender and receiver, when it floods the network. The packets carry the source route in the packet header, in order to dynamically discover unknown paths between nodes [SWS12].

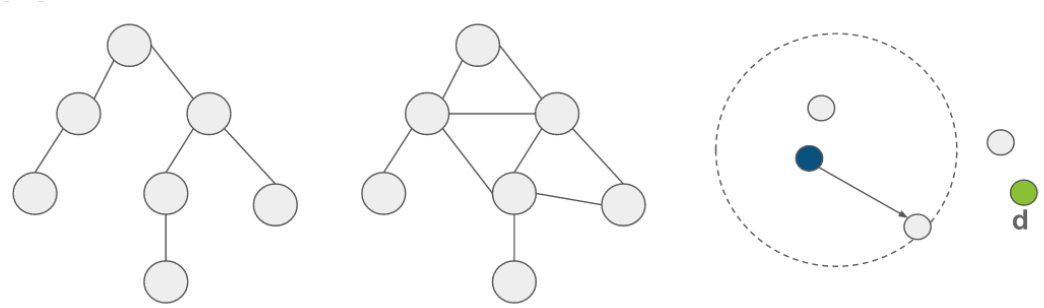
PADIS is an hybrid protocol used for data dissemination in MANETS, based on the geographic position of the data replicas, uses neighbouring devices resources to disseminate the data among other users [Mir07].

2.1.4 Overlay Protocols

The logical topology that support the interactions between nodes in Peer-to-peer (P2P) networks is usually abstracted with some form of overlay network. Overlays allow nodes to communicate to each other at the application level using logical overlay links that hide the physical network structure. This type of virtual network is used to support many different distributed protocols (as it is usually called on overlay networks deployed in an ad-hoc environment) and operations, such as indexing and route discovery [Aky07].

The lack of a coordination point, requires that nodes that make up the MANET, communicate to several other users rather than to a single centralized entity. So, the communication between nodes usually follows a multicast pattern, where one node sends a

[1] In a pure link-state protocol, all the links with neighbor nodes are declared and are flooded across the entire network by each individual node [Jac+01]



(a) Tree-based (b) Mesh-based (c) Stateless

Figure 2.1: Structure of routing protocols

piece of information to one or more nodes. However, sometimes overlay protocols provide unicast (one sender, one receiver) support or resort to both communication types.

Several overlay multicast routing protocols have been proposed, each one optimized for efficiency and robustness. These can be categorized, by their dissemination approach, as:

Tree-based overlays adapt tree structures, as in Figure 2.1a, according to specific (typically application) requirements. The most basic scheme, arranges all the routes to form a tree infrastructure with the source node as root, thus creating one and only one path between every pair of sender and receiver. This scheme is very efficient due to resource optimization but introduces overhead induced by tree reconfiguration on node relocation or failure. In a one-to-many on-demand media system such as oSTREAM [CLN04], the approach is to establish a minimum spanning tree and use media buffering at the host to aid in the distribution of asynchronous service requests for the same streaming media. However, this strategy is not useful for many-to-many situations. In a many-to-many systems such as Yoid [Fra00], there is a single shared tree from all members from the designated group. Trees are managed by a concept of parent/child relationship between members. Each member divides the set of all other members into two groups called parent-side and child-side members. The groups are defined such that parent side members are all members reachable via the parent and all others are child-side members.

Mesh-based uses multiple redundant routes, as presented in Figure 2.1b, that translates into a more robust routing system but wastes resources by forwarding unnecessary duplicate data across the multiple existing paths. This type of construction motivates, the use of gossip [HHL06] to route messages (or to build routing tables).

Stateless schemes avoid the overhead of maintaining a tree or a mesh infrastructure, such as the Differential Destination Multicast (DDM) protocol [Cor01] (represented in Figure 2.1c). In the DDM protocol, every data packet includes a header with all

the addresses of the receivers, in order to map the path to the packets final destination.

Overlay multicasting is not made on the network layer. Instead, a virtual overlay network is built on top of the physical network. Links in the overlay are unicast tunnels over the physical layer. These characteristics simplify the multicast operation and supply a static network topology overview even though the underlying physical topology is dynamic in nature [Su+09].

Routing protocols are not the only concern surrounding node communication, as they also depend on the connectivity of the network.

Some MANETs are fully connected[2], so the connection between two nodes is direct. However, in some cases not every node is within the reach of every other node to always enable, it may require more than a single-hop of communication. When this occurs, it common to categorize the network as being multi-hop, where source host packets are relayed through several intermediate hosts to enable packets to reach their destination.

In order to communicate or discover routes between two endpoints, nodes usually rely on some form of broadcasting[3].

The most simple broadcast approach is flooding, which is a dissemination mechanism that, every incoming packet is sent out on every outgoing line except the one it arrived on [Tan96]. Even though it generates vast numbers of duplicate packets, in many cases flooding is more efficient than maintaining a structure with the status of the data dissemination due to the high mobility of the nodes.

Broadcasting by flooding also presents some drawbacks, such as:

Redundant message overhead is caused by a large number of redundant messages being forwarded, particularly in a system with a highly connected topology. When multiple messages with the same message ID are sent to a peer by its multiple neighbors, all, except for the first message, are considered as redundant messages. These redundant messages are pure overhead: they increase the network transfer and peer processing burden without enlarging the propagation scope [JGZ03].

Contention can be caused by the retransmission of messages by nodes that are in close proximity. These transmissions are all from nearby hosts, and therefore may contend with each other. This can lead to a full disruption of the system due to an emergent phenomena known as broadcast storm [Tse+02].

Collision of two packets that forces the hosts to resend them, and thus reduces the network efficiency, making it more complex to have nodes execute a distributed protocol among them [Tse+02].

[2]Each of the nodes is directly connected to each other. In graph theory it is known as a complete graph or clique

[3]Transmitting a message to all members of a system using a single transmission from the perspective of the sender

The results presented in [Tse+02] provide an analysis on the above flooding deficiencies, and motivates the study of further improvements to overlay protocols [Su+09] and broadcasting algorithms [MLR06].

2.1.5 Discussion

Mobile Computing introduces some challenges associated with the mobility of the nodes and the inherent resource limitations, such as the battery capacity. However, clients still expect the same quality of service as in its static variant. In order to adapt to these new restraints and maintain the same quality, the classic client-server was revamped to be adjusted according to the needs of each system while satisfying the limitations of the nodes. Even though, the client-server was enough for most systems, this model did not take advantage of client-to-client communication, and thus appeared the Mobile Ad-hoc networks, that introduced a new set of challenges, consequence of the architecture decentralization. One of the most important challenges is node communication, since there is no centralized component. The solution to this problematic, involves packet routing protocols that ensure that nodes can send and receive messages correctly. Those protocols are responsible for finding a path, that allows the message to be sent, by maintaining a view of the networks topology (proactive) or just finding the correct path on the go (reactive). Either way, both methods must also concern implicit protocol problems that may disrupt the system, usually associated with message broadcasting. Another challenge is the highly dynamic topology, induced by the nodes mobility, which is addressed by creating an abstraction over the physical layer and thus easing the communication by maintaining a more static view of the network. This logical layer can be one or a combination of several different ways to structure the network, considering that each one has its trade-offs.

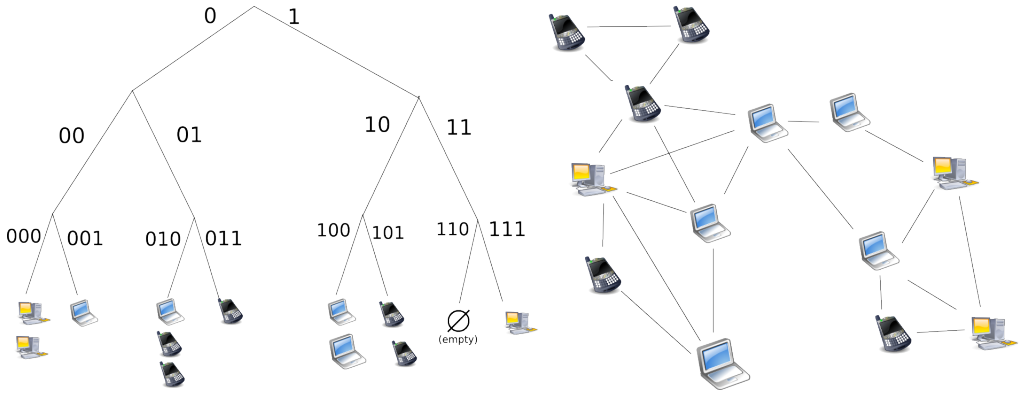
In this work, we have into account the challenges present in mobile computing and employ the necessary protocols to solve them.

2.2 Peer-to-Peer Networks

In P2P systems, nodes (i.e individual processes that are part of the system) are typically equipotent participants. Each peer acts as both a server and client, providing and consuming resources in a mostly decentralized fashion.

P2P architectures can be categorized by the degree of dependency to a centralized infrastructure:

Fully decentralized networks are completely distributed and do not rely on a central component to offer their services. On this type of network, the exit of any participant should not have any meaningful impact on the provided services as the overall operation of the system.



(a) Structured (b) Unstructured

Figure 2.2: Overlay Networks

Partially-decentralized networks have peers communicate to each other directly but use a pivotal infrastructure to provide or simplify the design of some of the network services [Sch01].

2.2.1 Overlay Networks

Overlay networks are classified accordingly to the method used to select and manage the links that are established between nodes (and consequently used to index available resources). These logical topologies can be of two types:

Structured overlay networks, Figure 2.2a, are organized according some pre-defined specific topology, in order to leverage the known information about the topology. Typically to improve the performance and cost of search and retrieve of resources.

The most common example of structured overlay are Distributed Hashtables (DHT), in which the ID of each node and object may consist of several dimensions (i.e. several IDs for each object). The hash keys are generated from descriptions of the content such as metadata or keywords.

In this type of topology, nodes are connected based on their identifier. Being that, they are linked to the nodes whose identifier are closest to their own, effectively forming an ordered ring (the identifier space is circular by definition).

On DHT networks, problems arise when there is high churn[4] rates that lead to high volume of traffic due to DHT maintenance. This factor leads to severe scalability and fault tolerance issues [Lei12; LR14] .

Unstructured overlay networks, Figure 2.2b, do not present any type of particular topology as a result of peers connecting to each other randomly.

[4] A phenomena where several nodes join and leave the network at a somewhat high rate [Sch01]

The fact that these networks are not imposed with a pre-defined structure, makes them easy to build and optimize locally. Being unstructured makes all nodes equally privileged, producing additional robustness to the network to churn.

Unstructured overlay networks limitations are related to their support to perform query routing over the network, because there is no correlation between the local position of a peer in the network and the data maintained locally at that node. Peers usually resort to mechanisms based on message flooding over the network in order to find the resources (e.g particular data pieces). Unfortunately, the lack of an association between the peers and the shared content does not assure that required data is found, except if every node in the network processes the query. [Lv+02; LR14]

P2P networks gained popularity from Napster [Pat01], that inspired unstructured and structured systems like:

Gnutella is a decentralized protocol for performing distributed search, that uses an unstructured overlay model [Kir03]. In Gnutella, nodes called *servants* perform tasks usually associated both clients and servers. They provide client-side interfaces through which users can issue queries, view search results, accept queries from other servants, check for matches in their local data set, and respond with corresponding results. These nodes are also responsible for spreading the information that maintains the network integrity. In order to join the system, nodes initially connect to one of the known hosts, that are almost always available. Then it opens connections to one or more nodes already in the network, to be able to communicate with and through them [Rip01]. A revamped Gnutella protocol as been proposed [Sin01], in which nodes are categorized hierarchically(known as Gnutella version 2). Nodes can be leaf-nodes or ultrapeers. Leaf-nodes only maintain one connection open, and that is to a Ultrapeer. Ultrapeers act as proxies to leaf-nodes connected to them. Reducing the number of nodes involved in message handling and routing, this design improves scabality and diminishes traffic among nodes, however nodes are no longer equal, meaning that the failure of a ultrapeer will have more impact on the system than the failure of a leaf node.

Freenet is a decentralized protocol for distributed data store, that also uses an unstructured model, to maintain and distribute information anonymously among network users. Freenet main focus is security and privacy [Fre15].

Pastry is a decentralized distributed object location and routing substrate for wide-area peer-to-peer applications, that resorts to a structured P2P model [RD01].

Chord is a distributed lookup protocol that enables peer-to-peer systems to efficiently locate nodes that store a particular data item. It only offers one primitive: given a key, return the nodes responsible for the key. Keys are distributed over the nodes

using consistent hashing[5] and replicated over succeeding nodes. Nodes typically store their successor nodes, forming an ordered ring (considering nodes identifiers), making it easy to reason about the overlay structure. For fault-tolerance a list of successor nodes is kept and for efficient lookups a finger table is used to jump over nodes in the graph [Sto+01].

2.2.2 Ad-hoc Querying

The infrastructureless nature of ad-hoc protocols, translates into a lack of resource indexation. So, when a host needs to find information, he does not know who are the other hosts that hold it. In order to obtain maximum network coverage, a protocol may resort to message broadcasting to find such hosts.

Many ad-hoc querying protocols have been proposed. The ones that do not assume nodes to be equipped with some sort of location hardware, make use of flooding, typically with a few optimizations. Despite many improvements that have been proposed to flooding protocols such as DSR [Joh+03] and ARA [GSB02], many messages are propagated unnecessarily (leading to resource consumption).

Gossip, or epidemic, protocols introduce a probabilistic variable that decides if a message should be forward or not, thus reducing the amount of transmitted messages.

Gossip protocols establish that when a node wants to broadcast a message, it selects f nodes at random and sends the message to them (f is a typical parameter named fanout). Upon receiving the message for the first time, each node proceeds to repeat this procedure [LPR07]. This is feasible as a result of the assumption that any node in the network can send a message to any other node, either because there is a direct link to that node or a route to that node is known. However, in the context of ad-hoc networks, this assumption is not realistic, due to the unique aspects of ad-hoc networks such as the mobility of nodes and the lack of a routing backbone. Nevertheless, this can be circumvented by taking advantage of a radio communications characteristic, the one hop broadcast message delivery. In wireless communications, messages are usually received by all the nodes that are in the close vicinity of the sender (one hop away). By managing this physical-layer broadcast feature, for instance by controlling the probability with which this broadcast is sent, gossip protocols can overcome this issue [HHL06].

This dissemination protocol (which can be used to build a somewhat inefficient routing mechanism), presents a bimodal behavior based on one branch of mathematics named *Percolation Theory*. Being bimodal means that in sufficiently large networks: in some executions the gossip process fails quickly and almost none of the nodes gets the message. However, in the remaining executions, a substantial fraction (close to 100%) of the network gets the message with high probability. The number of executions in which most of the nodes get the message, depends on the gossip probability and the network topology [HHL06].

[5] a hashing technique that adapts very well to resizing of the hash table. Typically k/n elements need to be reshuffled across buckets. k is the number of keys and n is the number of slots in a hash table.

Gossip protocols present themselves as highly scalable and resilient when implementing reliable broadcast. Studies [HHL06] have obtained results of up to 35% less messages in large networks, when compared with optimized flooding protocols. Even though it exhibits excessive message overhead in order to ensure reliability with high probability.

Some solutions for wired environments have explored gossip behavior to ensure robustness without consistency paying the price associated with the natural redundancy of gossip protocols.

Plumtree, or push-lazy-push multicast tree [LDT07], blends the tree-based broadcast primitives with gossip. By joining the reliability of a epidemic protocol and the small message complexity of tree-based protocols, and therefore covering the major faults of each other. Unfortunately, this protocol was not designed to wireless environments, and adopting it to cable ad-hoc networks is still an open challenge.

2.2.3 Discussion

Choosing between fully decentralized and semi-decentralized as well as between structured and unstructured architectures is a matter of comparing the trade-offs of each one of these design choices and select those that fit the particular requirements of the system being designed.

Fully decentralized and unstructured overlay networks are easier to build and maintain, while also being robust to faults but unable to provide no particular benefit when querying. Unlike structured networks, where search is efficient at the cost of sacrificing some robustness, and scalability.

Partially-decentralized architectures have the potential to have performance than fully decentralized architectures, due to the fact that certain operations, such as creating and maintaining an index to support search operations, can take advantage of the centralized infrastructures. Not only this approach can be efficient but it also benefits from the decentralized unstructured aggregation of nodes. However, using a central infrastructure introduces a dependency that goes against the purpose of achieving high scalability by avoiding potential bottlenecks in the architecture [YCM12].

In this work we explore a partially-decentralized architecture, where we only resort to the centralized component when necessary, as to minimize the dependency of a potential contention point. This should offer the possibility to address some complex queries in a efficient way while still benefiting from the common benefits of decentralized networks.

2.3 Edge Computing

Edge computing is an extension to the Cloud Computing paradigm. Cloud Computing grants clients quick deployments, cost efficiency in terms of maintenance and upgrade, and easy access to data. However, latency-sensitive applications short delay requirements may not be satisfied as a result of servers being at considerable distance.

This novel architecture draws away computing and storage services from centralized infrastructures, to the logical extremes of a network. The edges provide location awareness and wide-spread geographical distribution. Consequently, by covering broad areas, nodes can be in the vicinity of others which translates into low latency to communicate and interact with such nodes.

Edge and Cloud can coexist and be used in conjunction, consider for an example the case where, edge nodes may collect data from sensors, and process it before sending it to the Cloud [Bon+12].

2.3.1 Mobile Edge Computing

Mobile Edge Computing (MEC), or fifth generation (5G) cellular wireless networks, results from the enriching of mobile network carrier radio base stations provided services to include cloud operations [Bec+14]. The 5G networks objective is to provide higher data rates, enhanced end-user quality-of-experience (QoE), reduced end-to-end latency, and lower energy consumption in response to the increasingly demanding mobile Internet requirements. Figure 2.3 portraits a simplified representation of the MEC network topology.

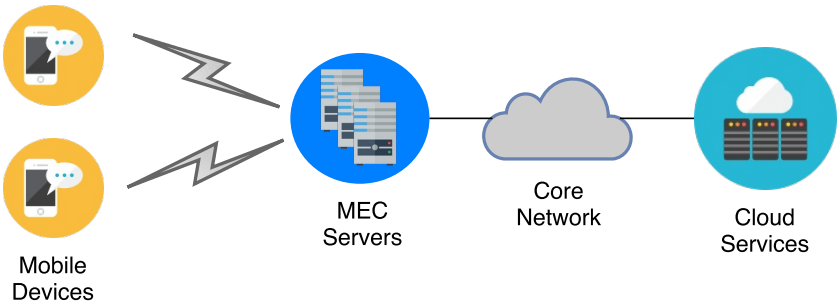


Figure 2.3: Mobile Edge Computing Architecture

In figure 2.3 there are three distinguishable entities:

Mobile devices such as smartphones and tablets, operated by regular mobile device users, in which network carrier subscribers can access cloud services within the range of the *Radio Access Network(RAN)*[6].

MEC servers owned and managed by network carriers, have the responsibility of managing traditional network traffic plus hosting and maintaining mobile edge applications such as *Cloud-RAN*. Cloud-RAN is a cloud service composed by radio access networks that resorts to the proximity of end users to provide better latency and throughput [Gud+13].

[6] A collection of base stations, each making independent control plane decisions on the radio layer with some loose distributed coordination via mechanisms such as SON (self organizing networks) [Gud+13]

Cloud Services such as providing access to particular software applications over the network (Application Service Providing) or act as a proxy server to deliver content to users with high availability and performance (Content Distribution Network).

In MEC the goal of resource migration is reducing latency and bandwidth consumption. Being that in the mobile paradigm, by performing certain tasks closer to the client in latency sensitive applications such as real-time, communication or media streaming, network congestion is reduced and applications get better quality of service (QoS) [Bec+14].

Latency reduction is achieved by the proximity of the cloud services, as resources are made available by other clients, and thus being more likely to be geographically closer.

Bandwidth reduction is related to the local processing of large data before sending to the cloud [AA16].

MEC compares to Mobile Cloud Computing (MCC) due to the fact that both provide cloud services. However, by resorting to opposing architecture models [YLL15]. This trend clearly shows that there is an interest to further support the operation of mobile application on the edge.

2.3.2 Discussion

The Edge Computing is an ever more recurring architecture, and a viable alternative to Cloud Computing, considering that nowadays devices can provide increasingly better resources with lower latencies, associated with their geographic position. The next generation telecommunication network, 5G, leverages this architecture to provide new and better services, such as low latency cloud services and applications.

The work developed on this thesis, uses concepts from edge computing similarly to the the way, they are used in the design of the 5G network.

2.4 Security

Security is an important element of any distributed system, especially when these systems communicate through wireless networks. A secure distributed system must ensure a set of essential properties:

Availability: ensures that resources are accessible to authorized parties at expected times. This should happen despite malicious attacks to the network.

Authentication: ensures that the communication between two nodes is legitimate, in other words, each participant is genuine and not an impersonator.

Confidentiality: ensures that data is protected, in way that only the desired recipients can access the information.

Integrity: ensures that messages between two nodes are not modified by a third, not authorized, party.

Non-Repudiation: each node must provide convincing evidence of the other's participation in a protocol session. If one peer falsely denies participating in a session, then the other peer can present his evidence to a judge, who can safely conclude that the other peer did participate. Crucially, the judge does not have to monitor the network traffic, but can make his judgement on the basis of the evidence alone [ZG96]

2.4.1 Security in MANETS

In the context of wireless networks, and in particular for MANETs, there are specific characteristics that significantly increase the challenges associated with enforcing these security properties.

Decentralization: Being infrastructureless requires nodes to have contributory collaborative roles in the network rather than ones of dependence. For instance, any security solution should rely on cooperative scheme instead of centralized one.

Wireless links: Non physical connections are more vulnerable than wired. Considering that attackers can come from every direction and target any node, wireless connections are more susceptible and harder to protect, in particular to personification attacks.

Multi-hop: In ad-hoc networks hosts also act as application level routers, and packets must pass through several, probably untrustworthy, nodes in their path between source and destination. As a result of this, malicious nodes may attempt to eavesdrop messages or modify their contents without being detected.

Resource limitation: Mobile nodes have limited battery supply and that may be used as a way to disconnect participants from the network. Not only battery, but also computing and storage limitations, restrict the use of complex security solutions such as cryptography.

Node mobility: The fact that several nodes exhibit, potentially fast paced, mobility patterns, complicates node tracking on a large network, and thus makes it more difficult to trace a malicious node [DB04].

2.4.2 Attacks

Attacks are composed by actions that intentionally aim at harming the network. These can be categorized by:

Origin

External attacks include attacks perpetrated by a node that does not belong to the logical network or is not allowed to access it.

Internal attacks include attacks launched by a malicious node that is actively part of the system [DB04].

Types

Passive attacks involve continuously collect of information, in order to use it for mounting of a future attack. The attacker does not disrupt the routing operation, it only eavesdrops packets, for picking up sensitive information or to deduce the network topology for routing information.

Active attacks refer to all remaining attacks that require active interaction with the system. However, these require the attacker to inject arbitrary packets into the network, and therefore increases the chances of detection [Lun00].

Attacks are not the only threats to ad-hoc networks, as internal nodes can (accidentally namely, due to hardware faults) deviate from the intended behavior and damage the the overall connection or performance of the system. For instance, when a node decides to not forward packets on behalf of other participants, in order to spare battery.

The physical network is not the only concern, being that overlay networks also have their vulnerabilities and may be subject to attacks themselves.

In particular, attacks to routing protocols can be classified as:

Modification attacks are those, in which malicious nodes cause redirection of the network traffic and DoS attacks by altering control message fields or by forwarding routing messages with modified values.

Spoofing attacks also known as impersonation attacks, occur when a node uses an identity that is not its own in the network, for instance by altering its MAC or IP address in outgoing packets.

Fabrication attacks correspond to the generation of false routing messages [DB04].

Sybil attacks happen when a peer participates in the network as multiple identities. By doing so, the attacker can gain large influence in the system, that can be used to disrupt communication in systems that rely on quorum based protocols or stealing information [Mon09].

2.4.3 Mobile Edge Network Security

The 5G network challenges arise from introducing IT applications into the telecom world. This integration imposes that these also obey the security requirements of the radio network.

Regarding physical security, there are also some challenges, due to the poor security conditions of radio base stations when compared to large data centers. Furthermore, issues can emerge when using third-party software applications, so these must come from trusted source, authenticated and authorized [Pat+14].

	3G/4G	WiFi	WiFi-Direct	Bluetooth
Transfer Rate	100Mbps	250Mbps	250Mbps	1Mbps
Energetic Cost	Very High	High	High	Low
Range	Unlimited	up to 100m	up to 100m	10m

Table 2.1: Technologies comparison

2.4.4 Discussion

In this work, we do not directly explore the challenges imposed by security of partially decentralized mobile edge architectures. We note however that the fact that these architectures shall include a centralized component, which is more reasonable to trust, might make some of the primary challenges easier to tackle.

2.5 Wireless Communication Technologies

The most common wireless communication technologies can be divided in two types, infrastructure based, such as WiFi and 3G/4G, and peer-to-peer based, such as Bluetooth and WiFi-Direct. All these technologies have their benefits and trade-offs, regarding transfer rate, energetic efficiency, range and monetary cost. Just one these technologies, may incur in monetary costs to the user which is 3G/4G, which is a telecommunications data service. In order to understand the differences between each of the available communication mechanisms, Table 2.1 provides a comparison of each one.

All the presented technologies, provide a high transfer rate, except for Bluetooth, however this one consumes less energy than the rest. Even though devices nowadays provide increasingly more and better resources in terms of storage and memory, they are still limited energetically. Considering that other technologies provide a high transfer rates and Bluetooth provides low energy connectivity, a combination of two technologies would benefit from the Bluetooths low energy consumption while still providing high transfer rates. A study [KBK] explored this premise and calculated the ratio between energetic cost and transfer rate when combining each one of these technologies with each other and concluded that the pair Bluetooth/WiFi-Direct provides the best ratio. One important characteristic of WiFi-Direct is that it organizes devices in groups and these groups have a restricted number of memebers (up to 8 per group) with no support of inter-group communication. Even though there is no backing from the original protocol, this challenge has been addressed in previous works using both WiFi interfaces, regular and direct [Teó+15]. This a solution uses two relay components that can be a centralized component or a hotspot device, in order to establish a communication path between two groups. Some previous works, also leveraged the combination of communication technologies such as:

CoolSpots is a system in which a switch between WiFi and Bluetooth interfaces is performed, based on policies that were studied in order to increase battery lifetime [Per+06].

Cell2Notify is a system in which they use 3G/4G interface to wake up the WiFi interface when VoIP call comes to achieve a better service and reduce energy consumption [WZM10].

2.6 Discussion

In this work, we follow the same direction as [KBK] and opt to combine Bluetooth and WiFi-Direct, to take advantage of the low energy consumption and the high transfer rates, due to the fact that in the context of this thesis, the energy factor is a primary concern.

2.7 Summary

In this chapter, we present and explain the subjects and fields covered in this thesis. Furthermore, we present some previous works that help understand the context of the work developed in this thesis.

Next we introduce the first proposed solution, Aether. This solution incorporates direct communication between devices in the classic client-server architecture, while resorting to the centralized component as a coordination entity.

3

Aether

This chapter describes and discusses, the first proposed solution. The premise of this solution, is that it is possible to use little Internet access to support mobile edge computing applications such as a distributed gallery. By using an hybrid ad-hoc protocol, Aether, that focuses on peer-to-peer communication for file transfer while still using a centralized infrastructure to coordinate devices, and handle some aspects of the system, achieves good functionality with good performance

3.1 Assumptions

In Aether, it is assumed that the application with which the user interacts, runs on a mobile device. It is assumed that the mobile devices provide a set of different technologies that allow them to communicate between themselves (*WiFi-Direct* and/or *Bluetooth*) and to access the Internet (through a *WiFi* infrastructure and/or 3G/4G). Furthermore, it is also assumed that the devices can simultaneously access Internet and communicate to other devices directly, using the same or a different technology from the ones available. Additionally, it is considered that the users explicitly select the contents that they wish to share with other users, and that those contents are indexable through a set of features. Finally, it is assumed that the great majority devices is concentrated on an area small enough to allow direct communication between all devices.

3.2 Proposed Solution Overview

This proposed solution will be explained using a bottom-up approach, in other words, departing from a solution close to the classical client-server architecture with no peer-to-peer interaction and move towards a solution that focuses on a more decentralized architecture while only resorting to a centralized infrastructure to process files and manage a central index. The intermediary solution will be used as a comparative baseline in the evaluation of the final solution.

3.2.1 Communication Model

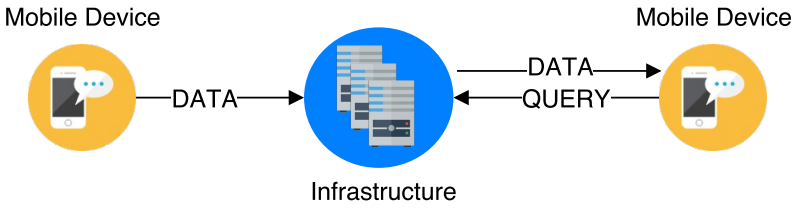


Figure 3.1: Baseline solution: Fully Centralized

The diagram of the centralized solution (Figure 3.1) represents a client-server architecture using mobile devices. In this solution mobile devices do not communicate directly, and consequently rely on the centralized infrastructure that stores, processes and serves data between different clients. This fully centralized solution has a bottleneck and single point of failure in the centralized component, which is the main motivation for the incorporation of peer-to-peer communication.

Aether architecture, depicted in Figure 3.2, departs from the centralized architecture to allow mobile devices to communicate between themselves whenever possible, in a peer-to-peer fashion resorting to a centralized infrastructure for coordination and resource indexing/querying. The centralized component maintains an association between device and resources it possesses. So that, when a peer searches for a resource, it only has to contact the centralized component, in order to obtain the address of the the device(s) that can provide it. After obtaining the address(es), a client requests and transfers the data directly from those devices. However, the owner of the resource may not be directly reachable, in which case the centralized infrastructure serves as a middle-man in the communication between such clients.

This approach, reduces latency and mitigates the bottleneck effect but it does not directly address, the single point of failure challenge.

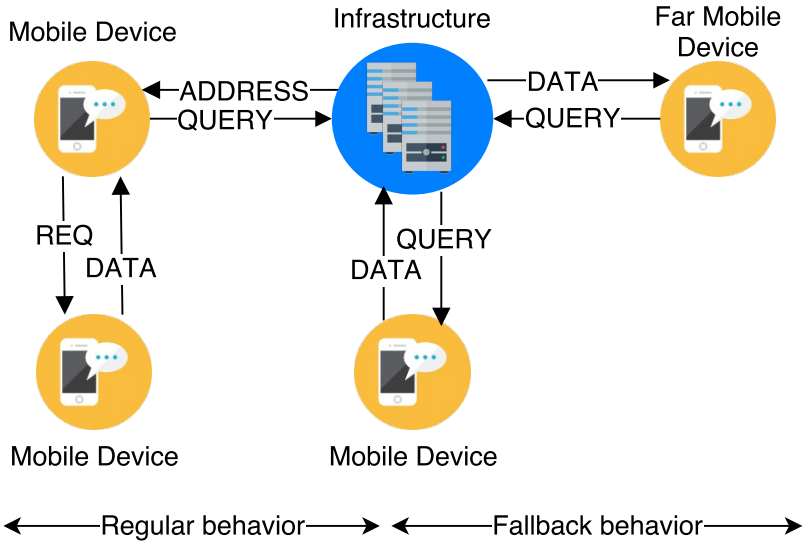


Figure 3.2: Aether - Peer-to-peer with coordination infrastructure

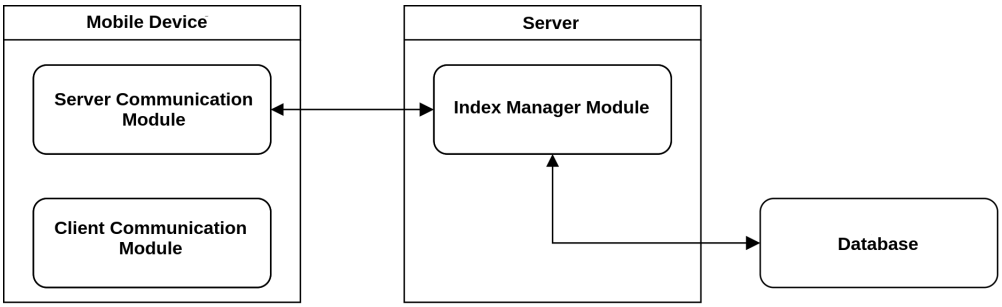


Figure 3.3: Aether : components architecture

3.2.2 Aether Architecture

Figure 3.3 outlines, in a simple way, the architecture and the interaction between the different components of the system in the Aether solution (Figure 3.2) that introduces direct communication to the classical centralized client-server architecture. The centralized architecture (Figure 3.1) is omitted due to its trivial nature and because the only difference is that it does not include the peer-to-peer communication module.

The client (mobile device), contains two communication modules. One module to communicate with the centralized component, which allows the client to query and update the index of the available resources in the system. This index, offers the client with a way of obtaining the identity of the devices, that offer relevant files according to the queries executed by the user. The peer-to-peer communication module, provides the necessary mechanisms to communicate and to exchange data directly with other clients (i.e, mobile devices).

On the server-side, there are also two components. One of them represents the resource index manager, that materializes all the index maintenance logic. This module is

responsible for processing of the clients query requests and process the updates to the index adding new entries to it (due to the production of new shared resources, or because an existing resource becomes available in a new device), as well as removing index contents that lose their relevance (i.e, because the contents were removed from the last device that had them). This component is also responsible for guaranteeing the index persistence.

The server also features a durable storage component (typically represented by a database) used to ensure the index persistence and to speed up its access.

Now that we have the functions of each primary component of the system, we can describe the interaction model between the system actors, which can be summarized as follows:

1. A client that is interested in certain files, requests to the server the list of identifiers from the devices that own the desired items (for instance through the use of a query);

2. After receiving the list of devices that own relevant resources (and the resource identifiers), the client iterates each address and tries to establish a connection to that device, in order to be able to request and receive one or more resources (i.e,files);

3. Upon obtaining the files, the client communicates with the server to update the index, consisting of the association of his identifier to each file entry that corresponds to the newly received files.

For this process to work properly, the peer-to-peer communication module maintains a service permanently active that waits for connection requests from other clients.

3.3 Implementation

In this section, the implementation of both solutions (the baseline centralized solution and Aether) is described and explained. To this end, we also expose the protocols employed and general implementation details of each component. Additionally, the fallback mode implementation is also discussed.

3.3.1 Centralized Solution

This solution is the simplest, with the highest degree of centralization, since it uses the classic client-server architecture, where a client sends the files that generates to the server. This last one, processes those files and extracts relevant information, stores the data in the file system and, among other information, stores the file path to each received item and indexes that information accordingly, as presented in algorithm 1.

By doing this, the system creates an index that establishes a relationship between the relevant metadata present in a file and its location in the servers local file system. Notice

Algorithm 1 Upload file - Server side

procedure UPLOADFILE($file$,$file\_info$)
  $path \leftarrow$ **storeInFileSystem(**$file$**)**
  $metadata \leftarrow$ **extractMetadata(**$file$**)**
  $file\_info.path \leftarrow path$
  $index.put(metadata, file\_info)$
end procedure

that this metadata is application specific and highly depends on the types of queries that are relevant for an application. For instance, to perform queries over image collections, the metadata will consist of a set of image features.

Building this index allows to efficiently locate (and later retrieve) files, in which the client is interested. For that purpose, the user submits a request to the server with the desired search criteria, (i.e, a query) according to the extracted metadata. In turn, the server will process the query internally against its index as shown in algorithm 2.

Algorithm 2 Execute query - Server side

1: **procedure** EXECUTEQUERY($query$)
2: $file\_ids \leftarrow$ **queryIndex(**$query$**)**$.ids$
3: **return** $file\_ids$
4: **end procedure**

After retrieving the identifier of each file that satisfies that query, the server sends that list to the client. Upon receiving the list, the client iterates through each id and downloads individually each image using a GET request to $/files/\{id\}$, where id represents the identifier of a particular file.

3.3.2 Aether

In contrast to the fully centralized solution described in 3.3.1, the server in Aether only stores and processes the files. It is not responsible for their distribution using them only to create and maintain the index. Instead the file transfer is achieved by direct communication between devices, whenever possible, following two different protocols, according the capacity of both clients to establish a direct connection between them. When both devices are able to establish a connection, (i.e, devices are within range) the files transfer is conducted by employing the protocol described in Algorithm 3.

According to Algorithm 3, when a client wishes to obtain one or more files, he communicates to the server and requests the list of devices that contain the desired data. After receiving this list, the client iterates each device, in order to initiate the file transfer. However, before establishing a connection, he verifies if the files offered by that client have not already been downloaded from another device. This verification is done to prevent unnecessary connections and data transfers, when the files are already present in the clients device. Finally, a Bluetooth connection is established and the connection details

Algorithm 3 File transfer with connectivity

```
1:  procedure FILETRANSFERP2P(server_address,query)
2:      list ← requestDevicesList(query)
3:      for all device ∈ list do
4:          if device ≠ mine then
5:              files_list ← checkExistingFiles(device.files)
6:              if files_list ≠ empty then
7:                  connection ← connect(device)
8:                  transfer(connection, files_list)
9:              end if
10:         end if
11:     end for
12: end procedure
```

negotiation and file transfer begins. This last setp is summarized in Algorithm 4.

Algorithm 4 Negotation and Data transfer

```
1:  data: server_address, device_id
2:  procedure NEGOTIATEANDTRANSFER(connection,to_receive)
3:      support ← check if the other client supports WiFi-Direct
4:      if support then
5:          connection ← startWDConnection
6:      end if
7:      if is file sender then
8:          list ← connection.read
9:          for all file ∈ list do
10:             connection.send(file)
11:         end for
12:     else
13:         connection.write(to_receive)
14:         while transfer is incomplete do
15:             file ← connection.read
16:             updateIndex(file.id, device_id)
17:         end while
18:     end if
19: end procedure
```

The negotiation and transfer process, starts by verifying if both devices support WiFi-Direct. If that is not the case, the protocol resorts to a fallback mode and maintains the Bluetooth connection open, in order to transfer the files by this technology. When both devices support WiFi-Direct, a new connection is established and the file transfer begins, with the request of the desired files and ends with the receipt of all the solicited data. In this protocol, WiFi-Direct connections have priority due to the fact, that they possess high transfer rates, when compared to Bluetooth, and therefore preferable to handle the file transfer.

Since normally the data transfer is made through direct communication between devices without resorting to the server as an intermediary, except when there is no connectivity, the index was adapted to relate the retrieved information of each file with the identifier of the devices in which it is available. The identifier of each device is composed by a pair formed by the *MAC address* from the *Bluetooth* and *WiFi* interfaces. So that when a file is received, the client makes a $/adddevice$ POST request to server, indicating that occurrence, with his and the files identifier as parameters. This request, updates the index with a new entry relating that device to that file. The same process is used when a device no longer possesses a certain file, but instead of updating the index with a new entry, it removes an existing one.

The index key used to identify a device eases the process of establishing *Bluetooth* connections, because knowing the devices *MAC Address* allows to contact the device directly, without being required to enable discovery mode and searching for the desired device (this mode is denoted as a highly battery consuming task). Unfortunately in the *WiFi-Direct* particular case, this is not possible, however the *WiFi* interfaces *MAC address* is enough to distinguish the device that we intend to communicate with, although it is necessary to search and filter all devices interfaces in range until the correct device is found to establish a communication channel.

However, when there is no connectivity, the fallback mode is selected and the server works as an intermediary in the file transfer. The implementation of this mode uses two separate server-side components that work together, one is the Google Cloud Messaging service (GCM) and the other is a REST web service that temporarily stores the data and offers a link to access it as long as it is available. The GCM service is a free service that enables developers to send messages between servers and client applications. Furthermore, it handles all aspects of queueing of messages and delivery to and from the target client app [Goo16a]. To use this service, clients need to obtain an Instance ID from the GCM server. This identification is later used as an address to that device, when sending and receiving messages. In the context of this work, and more specifically for the case of Aether, this service is used to notify a system user that another client requested a certain file from him. This last one, after being notified, uploads the file to the server the web service and retrieves a link. The GCM service was not used to transfer the files due to its limit of 1KB per message. When the link is acquired, it is sent (through the GCM) to the user that executed the query, as depicted in the example presented in figure 3.4.

In the interaction depicted in Figure 3.4 "Mobile Device 2" issues a query to the centralized component, that in turn returns the list of devices that satisfy that query. With the list of devices, "Mobile device 2" is now able to inquiry "Mobile Device 1" with the list of desired files. After receiving the list, "Mobile Device 1", confirms that it owns the desired data. After confirming, this data is then sent to the data web service that returns a set of temporary access links to the device that executed the query ("Mobile Device 2"). Finally, the device iterates the links and downloads each file from the server. A detailed implementation of the fallback mode is presented in algorithm 5.

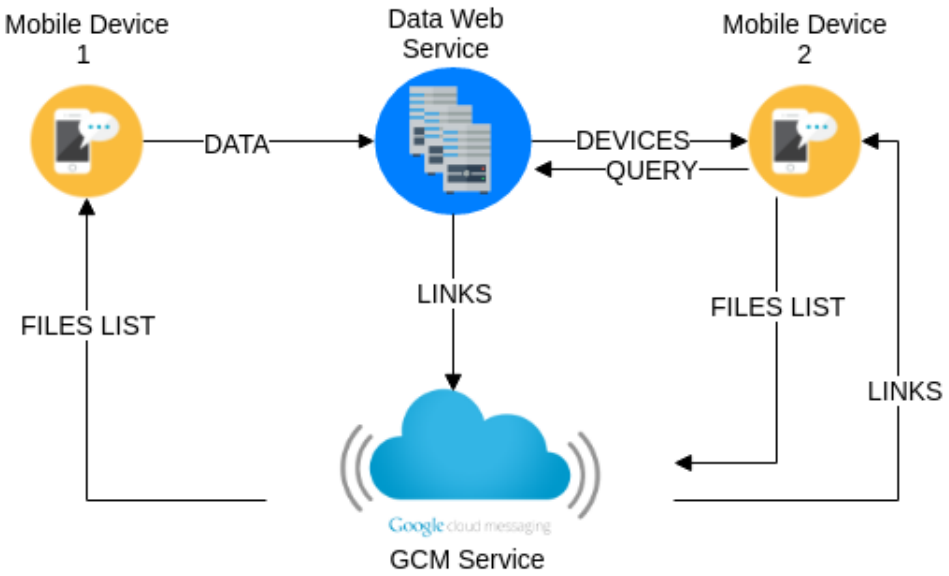


Figure 3.4: Aether fallback communication model

Algorithm 5 Fallback mode - Receiver

```
1:  data: URL : GCM_URL
2:  procedure FALLBACKRECEIVER
3:      token ← register to GCM
4:      store(token)
5:      sendToServer(token)
6:      while true do
7:          listen to GCM
8:          if message is received then
9:              if message.TYPE == QUERY then
10:                 file ← findFile(message.data)
11:                 link ← uploadToServer(file)
12:             else if message.TYPE == LINK then
13:                 downloadImage(message.data)
14:             else if message.TYPE == FILE_COUNT then
15:                 display to user the count of files to download
16:             end if
17:         end if
18:     end while
19: end procedure
```

In fallback mode (algorithm 5), the first step, which is done once per session, is registering to the GCM service. This registration does not involve user interaction and generates a token, that is stored locally and in the server, which will be used to identify and authenticate, every time a client wishes to send a message. Once this step is complete, the client keeps on listening for messages coming from the service. However, this listening is not simple polling (pinging the server to query for new messages), it uses the GCM push notification model, which is event-driven and only creates a new connection only when there are new messages. When a message is received, it can be one of three types, and they are the following:

Query messages correspond to a query sent by another client, triggering a mechanism that finds the requested file, uploads that file to the server, in order to obtain a link that enables its download. Lastly, the link is sent to the client that executed the query.

Link messages contain a link, that points to one of the files, requested by the client, so that he is able to download it.

Count messages, not only inform the user of how many files are expected to be received, but also to allow the system to determine when the reception of all images is complete.

Another possible, and simpler fallback solution, would be using the centralized component to not only store the index but also store the files, and be accessed to retrieve the files when a device is not within reach. However, in this work we followed a different direction, because we wanted to explore the impact of reducing the responsibility of the centralized component as much as possible, including in terms of storage.

3.4 Summary

In this chapter we present and describe the implementation of an alternative architecture to the classic client-server, Aether. This architecture incorporates peer-to-peer communication for the file transfer while still maintaining a centralized component for file indexing. Furthermore, also includes a fallback mode, used when there is no connectivity between devices. The explanation of the Aether architecture, includes a baseline centralized model, in order to further understand the reasoning process behind Aether, and its implementation.

In Aether, the centralized component, processes the files and maintains an index that establishes a relationship between a file metadata and the devices that contain it. Devices can then query the centralized infrastructure and obtain the list of devices, that own the desired files. This list is then used, to contact each of the devices and transfer the data.

The used direct communication technologies are a combination of Bluetooth and WiFi-Direct, with different purposes. The first technology, keeps on listening for communication requests, due to the fact, that Bluetooth has low energy consumption and low transfer rates, in contrast to WiFi-Direct that has a high energy consumption but high transfer rates which is used to transfer files.

Aether also includes a fallback mode, that allows devices that are not within reach of others, to still interact with the network, by using the central component as an intermediary in data transfer. However, this is not done by simply polling the server for new updates, Aether resorts to GCM (Google Cloud Messaging) service, that is part of the Android OS, which notifies users when there is an intent of communication.

This architecture could be improved by reducing even more the responsibilities currently hold by the centralized component and moving them to the edges of the network, thus increasing the degree of decentralization. We do so in the next chapter, when we introduce an alternative architecture that we dub Aether+.

4

Aether+

This chapter describes and explains an alternative solution, named Aether+, that improves Aether, by lowering even further the dependency to a centralized component, while still using this last one as fallback. Through the decentralization of the original Aether solution, the system can function without an Internet connection and even if there is no connectivity between devices and if Internet connectivity is available, users can continue interacting with the system through the centralized component.

4.1 Assumptions

The Aether+ shares the same assumptions as the original Aether solution, that was previously described in Chapter 3.

4.2 Proposed Solution Overview

This solution, introduces some changes to the original Aether design, more specifically, to the communication model and architecture. These changes, are required to cope with further decentralization and with the proposed fallback protocol.

The Aether+ (figure 4.1) organizes mobile devices into mesh-based groups, taking into consideration on their physical proximity. The members of each group also establish bridging connections to other groups, in order to allow inter-group communication. When there is no possibility to establish any bridge connection between groups the centralized component is leverage to enable communication between different groups. Therefore, when performing queries, devices attempt as much as possible to restrict

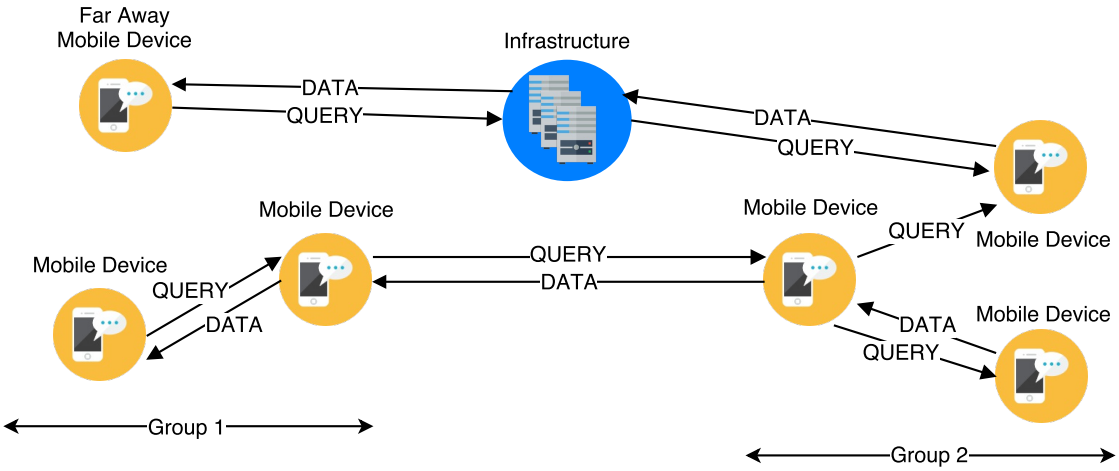


Figure 4.1: Aether+ - Peer-to-peer with fallback infrastructure

themselves to contacting nodes in their own groups, as a way to minimize communication overhead, and potentially minimize the load on the centralized infrastructure. When this is not possible, for instance, when the searched resource is not locally available at any element of the queries device group, connections are used as a gateway, through which queries to other groups are made. The gateway member receives the query from a fellow group member, and retransmits that query to the foreign group with which he as a connection, so that the other group can query its member for the desired resource/resources.

The centralized component in this solution, is no longer a requirement for the system to operate in all scenarios. Since it is only used as fallback serving as an intermediary in data transfer and/or querying, when there is low or no connectivity between devices or groups of devices.

As illustrated on Figure 4.1, the leftmost device in group 1 queries another member of its own group, so that he can also query group 2 through the bridge connection. In turn the group 2 leader, floods the group network with the query. A group leader is a device that possesses a connection to another group.

The resource transfer follows the same protocol, when the desired data is found, if the destination is a member outside the group, the resource owner sends the data to a member of its own group that has a bridge connection, that consecutively transmits the data to the intended group, that in turn delivers to the group member that executed the query (as depicted in Figure 4.1, where the rightmost peer has to send data to the leftmost peer by resorting to elements of both groups). The group communication is not restricted only to adjacent groups, since one group can communicate with another, using an intermediary group to mediate this contact.

The fallback case is also depicted in figure 4.1 where the device in top left corner, named "Far Away Mobile Device", represents a device that is not contingent to the area covered by the other devices and therefore cannot establish a direct connection. In order to enable, this device to interact with the remaining devices that form the system, the

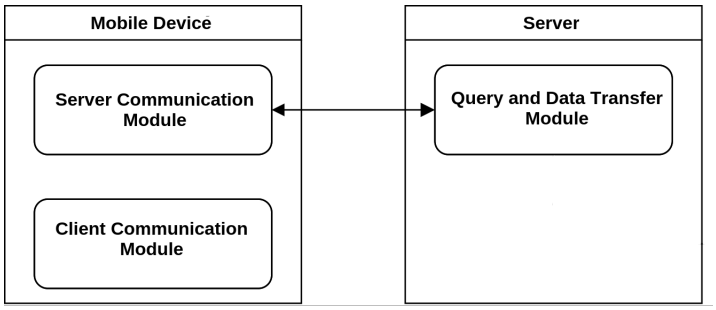


Figure 4.2: Aether+ architecture

centralized component will work as an intermediary in the communication between the "Far away Mobile Device" and the existing groups. A simple interaction example is depicted in figure 4.1 where the distant device queries the centralized component, which in turn queries every group member including a group 2 device, that fortunately owns the desired data and sends it through the same inverse communication path.

4.3 Architecture

The architecture used in Aether+, is based on the Aether concept, but removes and changes the responsibility of some of the original components, to be able to adapt to the new degree of decentralization.

Figure 4.2 represents the Aether+ architecture. In contrast to the original Aether architecture (Chapter 3), the server-side components have changed. The database component is no longer required, since the files are no longer persisted in a centralized component but rather stored locally in the client devices, as well as the index manager component that is of no use, considering that is no longer the responsibility of the server to maintain an index. Instead, there is a component that is responsible for receiving and routing, query requests and data to their destinations. This component serves as an intermediary in client-to-client communication, when it is not possible to establish a direct communication channel between devices. The client modules have also changed their behavior, in order to adapt to the fact that communication is now primarily done in a peer-to-peer fashion and to the modifications done in the server-side. The server communication module that used to interact with the index, now transmits and receives queries and data from and to other clients through the centralized component, when necessary. The peer-to-peer communication module suffered some changes being now responsible for the maintenance of a routing table containing information about the topology of the network that follows a group based organization, and therefore allows communication between devices for both querying and transferring data.

The interaction model for this solution, is more complex than the one presented in the original solution, which is a known characteristic of decentralized architectures. In this model, when there is connectivity between clients, system actors operate as follows:

1. A client that is interested in certain files, broadcasts a query to all the members of his own group.

   (a) If a fellow group member(s) owns those files, he sends them directly to the requesting client and the process terminates;

   (b) If all or some files, are not found within his own group, the exectuion procedes to step 2.

2. The client transmits the query to a same group member that has a bridging connection to another group, in order to propagate the query to other groups. Until the files are found and sent to the requesting client or until a parameter associated with query messages named *time-to-live(TTL)* expires [1].

When a client cannot connect with any of the existing groups (i.e. the client is not physically close enough to establish communication channel to other clients), the protocol resorts to its fallback mode, in which the interaction with the system uses the centralized infrastructure as follows:

1. A client is interested in certain files, so he sends a query request to the centralized component, that in turn propagates the query to every available (different) group member;

2. After a group member that owns the desired files is found, he transmits the data to the centralized component, so that this last one can route it to the client that issued the query.

4.4 Implementation

This section discusses how Aether+ was built and describes the implementation of the components that constitute the architecture described above.

This solution, differs from Aether in the way that the centralized component is not a requirement for the system to operate, but an optional element that maximizes network coverage when devices cannot establish direct communication channel. Usually because they are not in the physical proximity. Since all the communication is mostly achieved in a peer-to-peer fashion, the resource index is now distributed across all devices.

In the Aether+ solution, there are two modes of communication, that may work simultaneously or separately, which are the following:

Peer-to-Peer Communication mode works by forming a peer-to-peer network, organized in groups arranged by physical proximity;

[1] Query messages are generated with an initial TTL, which is decreased every time the message is forwarded. If it reaches a value of zero, the TTL is said to expire.

Fallback Communication resorts to a centralized component, in order to establish a communication path between two or more devices that are not within reach of each other through any of the available peer-to-peer technologies, serving as an intermediary when querying and transferring data;

Simultaneous Communication is the combine of both communication modes, allowing peer-to-peer and fallback users to communicate, and adjust the selected communication mode to the existing environment.

4.4.1 Peer-to-Peer Communication

The peer-to-peer network main communication technology is WiFi-Direct, that by its API allows to easily form a simple group based peer-to-peer network. The networks topology follows a 1:N model where N clients (with N equal or below eight) are connected to one group owner, forcing the group owner to work as an intermediary in client-to-client communication, and thus draining more battery and consuming more resources from his device than from the rest of the group members. In order to solve this problem, the partially-decentralized architecture, migrates the 1:N model to a N:N model (mesh-based network topology) where every group member can connect directly to every other member of its own group. This migration, is materialized by the implementation of a routing table, that establishes a relation between a clients MAC address (virtually unique) and his network information (name, Group Owner MAC, Group ID, IP, Bridge details), as presented in table 4.1. Note that the name corresponds to the device name and timestamp to the last update temporal reference.

WiFi MAC	GO Owner MAC	GID	Name	BT MAC	IP	Bridge	Timestamp

Table 4.1: Routing table header

This routing table provides a local view of the network topology, since it stores information about the neighbours of each client, in other words, it only contains the network details of fellow group members and members of groups with which the clients group has created a bridge. A special packet structure was also conceived, in order to propagate the necessary information maintained in the routing table.

Figure 4.3, illustrates the structure of the packet used in this solution. Starting from the top, the packet contains the source and destination addresses for both WiFi and Bluetooth (notice that this packet structure is employed to support communication using both WiFi-Direct and Bluetooth). In addition to the MAC addresses, the packet also contains the sources IP. Lastly, the packet contains three smaller sections, each one representing one of the following:

ID represents a packet identifier that may be temporarily non-unique, which is used to verify if a packet has already been received or not;

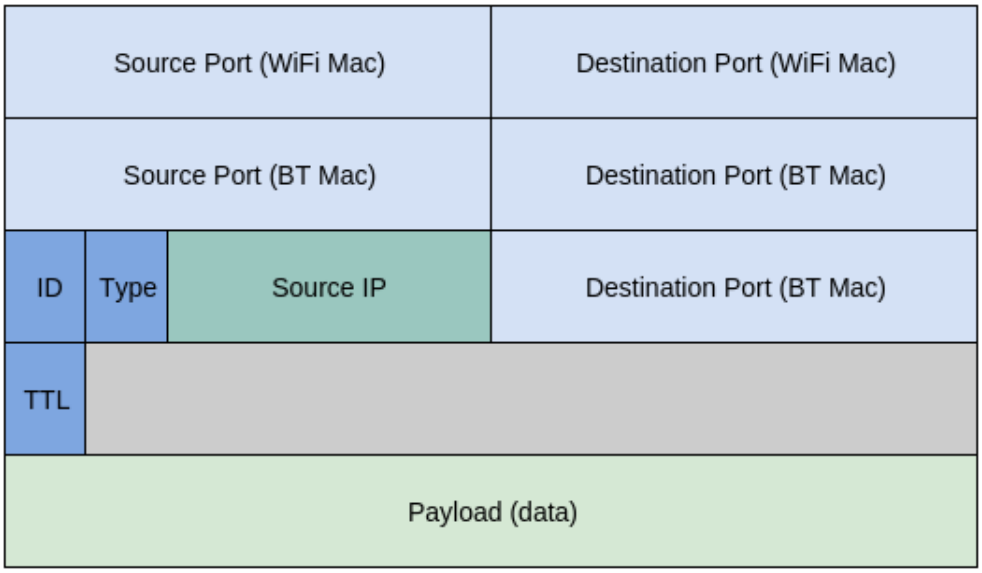


Figure 4.3: Packet structure

Type corresponds to the type of the packet, so that the packet receiver knows how to handle it. A packet can be of type:

HELLO is a mechanism triggering packet, that is sent to the group owner upon joining a group, in order to obtain the routing information;

HELLO_ACK is a response packet, for packets of type HELLO, containing the group routing information;

BYE is a notifying packet, used to warn the other clients that the sender is about to leave the network;

QUERY is a packet containing a query message;

UPDATE is a packet containing routing information, so that the receiver can update its routing table;

FILE is a packet containing file data;

FILE_COUNT is a packet that indicates the total count of files to receive. This packet is sent before sending the file type packets.

TTL is the time to live current value, and is used to determine if a packet has already reached the predefined maximum number of hops. By default the TTL is set to three.

These packet headers are useful to help building and maintaining the routing table, which begins when the client connects to a discovered peer, after the peer discovery phase, as denoted in Algorithm 6.

Algorithm 6 shows that after a connection is established, the first information that is used to populate the routing table is the client own details, so that when the table is

Algorithm 6 Connect Peer

```
1: data: RoutingTable : R
2: procedure CONNECTGROUP(peer)
3:     connection ← connectWiFiP2P(peer)
4:     R.self ← connection.info
5:     receiver ← startReceiverThread
6:     sender ← startSenderThread
7:     if !connection.isGroupOwner then
8:         send(GroupOwner, Packet(HELLO, null))
9:     end if
10: end procedure
```

transferred to other members contains this information. After storing and initiating the necessary threads to handle packets used to communicate with other clients, if the user is not the group owner, he sends a *HELLO* type packet to the group owner, that, when received, triggers the exchange of clients information, and leads both nodes to update the information in both of their last routing tables. The handling of the received packets (such has the *HELLO* type packet) is the responsibility of the *Receiver*(Algorithm 7), in which, all the routing table update logic is processed.

As seen on algorithm 7, when the *Receiver* encounters a *HELLO* type packet, it implies that a new client as joined the group and therefore, the receiver needs to update his and the other group members routing table including the new client information. To this end it sends to every group member an *UPDATE* type packet with the new information, except for the new client to whom he sends a *HELLO_ACK* type packet with the contents of his local complete routing table (which contains information regarding all devices in the group). When a *HELLO_ACK* or *UPDATE* type packet is received the routing table is updated, accordingly.

After bypassing the 1:N model with a mesh-based topology, WiFi-Direct presents another limitation, it does not support inter-group communication and a member can only belong to one group, which means that groups are isolated entities without external WiFi-Direct communication and therefore, by default, a network can only have a maximum of eight users. The workaround used in this architecture, is to establish a bridge between different groups by connecting each group member to a randomly selected member of another group, using an alternative peer-to-peer technology, in particular Bluetooth. Finding a random member from another group, requires the Bluetooth adapter to activate discovery mode, which is a heavy procedure that drains a significative amount of energy from the device battery. However, the scan only lasts a short period of time, since its goal is to find another device that belongs to a different group (there are no relevant restrictions regarding bridge connection). Considering these limitations, an algorithm (8) was implemented, in order to minimize the impact of scanning the surroundings and still be able to find several devices, until one of them satisfies the requirements for establishing a bridge connection.

Algorithm 7 Receiver - Routing Table Setup and Update

```
1:  data: Queue<Packet> : PQ, RoutingTable : R
2:  procedure RECEIVER
3:      p ← PQ.remove
4:      if p.TYPE == HELLO then
5:          new_cliente ← p.network_info
6:          R.put(new_client)
7:          for all client ∈ R do
8:              if client = me || p.sender = me then
9:                  continue
10:             end if
11:             send(client, Packet(UPDATE, new_cliente)
12:         end for
13:         send(new_client, Packet(HELLO_ACK, R))
14:     else if self is the indended recipient then
15:         if p.TYPE == HELLO_ACK then
16:             R.updateTable(p.data)
17:         else if p.TYPE == UPDATE then
18:             R.updateEntry(p.data, p.network_info)
19:         end if
20:     else
21:         ttl ← p.ttl
22:         ttl − −
23:         if ttl > 0 then
24:             p.ttl ← ttl
25:             send(p)
26:         end if
27:     end if
28: end procedure
```

Algorithm 8 Bluetooth Scan

```
1:  data: List<Device> : seen_devices
2:  procedure BLUETOOTHSCAN(tries_rate,interval)
3:      tries ← 0
4:      while bridge is not established || tries < tries_limit do
5:          found_devices ← BTAdapter.search
6:          for all device ∈ found_devices do
7:              if device ∈ seen_devices then
8:                  found_devices.remove(device)
9:              end if
10:         end for
11:         while found_devices ≠ empty do
12:             established ← connectDevice(device)
13:             if established then break
14:             end if
15:             seen_devices.add(device)
16:         end while
17:         Sleep(interval * tries)
18:         tries + +
19:     end while
20: end procedure
```

Algorithm 8 presents the detailed algorithm employed to perform Bluetooth discovery in Aether+, trying to establish a bridging connection with a random device in the surroundings. This mechanism, starts by scanning the location for possible targets. The process of scanning obtains a list of detected devices, however when the connection is not established on the first try, some of detected devices may have already been contacted. To avoid these unnecessary connections, a list with the already contacted devices is maintained, filter the devices that have been previously contacted without success. Once the list has been filtered, the remaining devices are contacted and an attempt to establish a bridging connection is made. If none of the devices is successful, meaning that they all belong to the same group as the device that executed the scan, the mechanism sleeps for a parametrized amount of time, to save resources and waits to reach a location with different neighbouring devices. When a Bluetooth connection is established, the bridge negotiation begins as described in Algorithm 9.

As seen in Algorithm 9, after connecting, devices exchange their respective group ids, in order to check if they belong to different groups, if that is not the case the connection is interrupted and the process is restarted by selecting a new peer. Upon checking that both clients have different group ids and therefore belong to different groups, they proceed to store the bridge connection details and to send one another their routing tables. Lastly, this routing table update needs to be propagated to their respective fellow group members, which is performed by sending an *UPDATE* type packet to the group owner, that in turn will distribute it to all other devices of the group

Since all the connections are established and the routing tables are updated, packets

Algorithm 9 Connect to establish Bridge

```
1: data: RoutingTable : R
2: procedure CONNECTDEVICE(connection)
3:     server ← decide who is server
4:     if server then
5:         groupID ← connection.read
6:         if groupID ≠ my_groupID then
7:             connection.write(my_groupID)
8:             my_Bridge ← new Bridge(groupID, connection.info)
9:             connection.write(R)
10:            otherTable ← connection.read
11:            R.updateTable(otherTable)
12:        else
13:            Interrupt Connection
14:        end if
15:    else
16:        connection.write(my_groupID)
17:        GID ← connection.read
18:        my_Bridge ← new Bridge(GID,connection.info)
19:        otherTable ← connection.read
20:        R.updateTable(otherTable)
21:        connection.send(R)
22:    end if
23:    send(GroupOwner, Packet(UPDATE, R))
24:
25: end procedure
```

need to be routed through existing network paths. This is achieved by the strategically encoded Algorithm 10.

Algorithm 10 Packet Routing

```
1: data: RoutingTable : R
2: output: IPBundle<Method,Address>
3: procedure PACKETROUTER(client_mac)
4:     client ← R.get(client_mac)
5:     if client == NULL then
6:         return new IPBundle(WIFI, DEFAULT_GROUP_OWNER_IP)
7:     end if
8:     if client.groupID == my_groupID then
9:         return new IPBundle(WIFI, client.IP)
10:    else
11:        if my_Bridge.groupID == client.groupID then
12:            return new IPBundle(BT,client.BTMac)
13:        end if
14:        ip ← DEFAULT_GROUP_OWNER_IP
15:        for all member ∈ R do
16:            if member.groupID == my_groupID then
17:                if member.Bridge == client then
18:                    return new IPBundle(BT,member.IP)
19:                else if member.Bridge.groupID == client.groupID then
20:                    ip ← member.IP
21:                end if
22:            end if
23:        end for
24:        if I am group owner && ip == DEFAULT_GROUP_OWNER_IP then
25:            for all member ∈ R do
26:                if member.groupID ≠ my_groupID then
27:                    return new IPBundle(WIFI,member.IP)
28:                end if
29:                dropPacket
30:            end for
31:        end if
32:        return IPBundle(WIFI,ip)
33:    end if
34:    dropPacket
35: end procedure
```

Algorithm 10 begins with the retrieval of recipient information. After this, if there is no (local) knowledge about that recipient of the packet in the routing table, the packet is sent to the group owner, since he has the higher chance of having the most updated routing table from all the group members.

When both the client and the recipient belong to the same group, the client only needs the recipients IP in order to transfer the packet. However if that is not the case, the client must resort to inter-group communication, if possible, by checking if he can use

his own or a fellow group members bridging Bluetooth connection. This may also not be achievable due to the fact that a bridge from the clients group to the destination group may not exist meaning that no entity for it will be found in the clients routing table. In this case, the client sends the packet to the group owner, that in turn repeats the process with the most updated topology information. Lastly, if the group owner has no viable path to the recipients group, he routes the packet to another random group, in order to expand the search for the recipient. If there is no knowledge of other groups or an unexpected error occurs, the packet is dropped.

Given the network topology and packet routing strategy described above, querying and transferring data between network members becomes possible by leveraging it.

Before being able to query and transmit data, the system must provide unicast and broadcast communication primitives. Since there is no global view of the networks topology, as maintaining this information for medium scale systems would be too expensive and hence, not scalable, particularly considering that a mobile-edge computing environment is potentially, dynamic, unicast and broadcast primitives have to be designed to operate with partial system information only. To overcome this challenged, considering that this partially-decentralized solution only provides a local view of the network topology to nodes, the logic used to implement these communication primitives is inspired in random walks [Lv+02] and gossip protocols [HHL06; LPR07]. The unicast communication implementation uses the same routing logic, as the one presented in algorithm 10, where the packet first checks if the recipient is within the same group or if it belongs to any of the bridged groups, before sending it to a randomly chosen group (random walk). The recipient group, in turn, repeats the process until the packets destination is found. The broadcast communication implementation is inspired in a gossip-based broadcast protocol [LPR10] and its described in Algorithm 11.

The logic presented for the broadcast communication mechanism (Algorithm 11), works as follows. When a client desires to broadcast a message, such as a query, the protocol selects f (where f represents a parameter named *fanout*) random neighbour clients from the routing table excluding the broadcasts creator. After randomly choosing the clients, the gathered list is iterated and the message is sent, with a time to live (also defined by an application specific parameter) to each one of these neighbours. When a node receives a broadcast message for the first time checks if the packet maximum number of hops, and if that is not the case, the protocol repeats the random clients process (now also excluding the sender of the packet) and retransmits the message to the selected ones, after decreasing the packets time to live by one unit.

Due to the inherent redundancy level of gossip protocols, its necessary to check for duplicate packet receptions. In order to be able to detect this phenomena, each packet contains an identifier, that is logged when processing the packet for the first time (this identifier was discussed previously after presenting the structure of the packet). The list of tracked identifiers is garbage collected from time to time (according to a parametrized interval), since the identifiers may repeat when there is a lot of packet transmissions in

Algorithm 11 Send Broadcast Message

```
1: data: RoutingTable : R
2: procedure BROADCASTMESSAGE(msg, packet,fanout,ttl)
3:     if packet.ttl == 0 then
4:         return
5:     end if
6:     neighbours ← R\{self, p.sender}
7:     clients ← select fanout random clients from neighbours
8:     for all c ∈ clients do
9:         if !rebroadcast then
10:            send(c, Packet(msg, ttl))
11:        else
12:            p.recipient ← c
13:            p.ttl − −
14:            send(c, p)
15:        end if
16:    end for
17: end procedure
```

the system.

4.4.2 Fallback Communication

The implementation of the fallback communication strategy is another relevant and different, aspect of this architecture in relation to the previously discussed alternatives of fully decentralized architectures, since it enables clients that are not within the reach of other clients to still interact with the system. This is achieved by making use of the centralized component, as an intermediary in client-to-client communication, when querying and transferring data. Although the strategy employed for the fallback communication is somewhat similar to the one described in Aether(Chapter 3), the difference lies on the communication model, which instead of being triggered by a device contacting the server to obtain the list of devices that own resources relevant to a given query, begins with sending the query to the GCM service, so that this last one can broadcast the query to all other devices, as exemplified in Figure 4.4.

In Figure 4.4 "Mobile Device 1" issues a query to the GCM, that in turn broadcasts the inquiry to "Mobile Device 2" and "Mobile Device 3". After receiving the query, both devices check if they own the desired data, which is found in "Mobile device 2". This data is then sent to the data web service, that in turn, generates and returns a temporary access link to that file. The obtained link address is then sent to the device requesting the files ("Mobile Device 1") through the GCM service, so that he can later download the files (similar to the Aether solution).

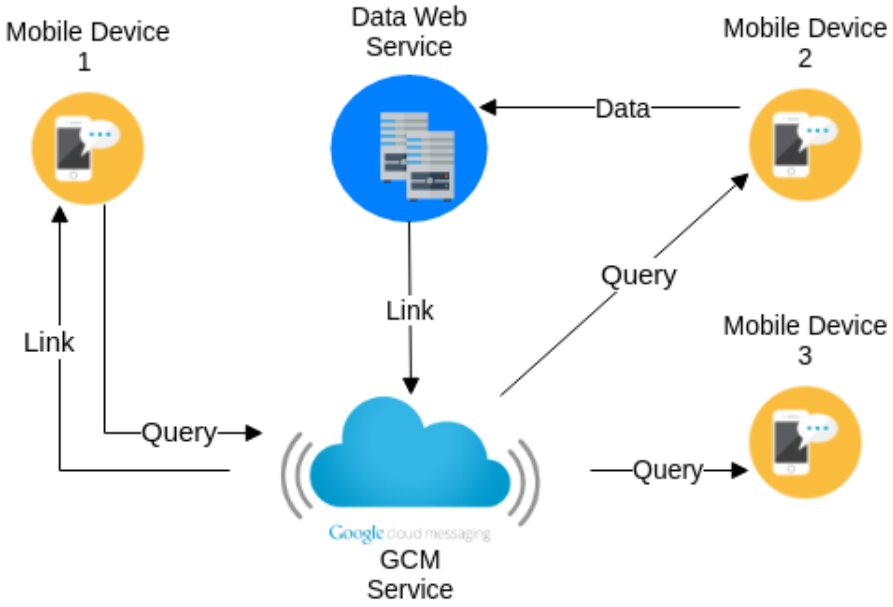


Figure 4.4: Aether+ fallback communication model

4.5 Summary

In this chapter, we present a an improved version of Aether, described in Chapter 3, the Aether+, by completely decentralizing the architecture, rendering the centralized component as optional. However, the centralized component in this architecture is still used as a fallback communication mechanism. This architecture, establishes WiFi-Direct communication channels and organizes devices into groups, based on their physical proximity. However, WiFi-Direct does not allow groups with more than eight elements, requiring some type of inter-group communication. In order to overcome this challenge, the protocol establishes bridging Bluetooth connections with random group members of different groups. The Aether+ also includes a fallback communication somewhat similar to the one present in Aether. This mechanism, enables direct communication, when there is no connectivity between devices and inter-group communication, when there are no bridge connections.

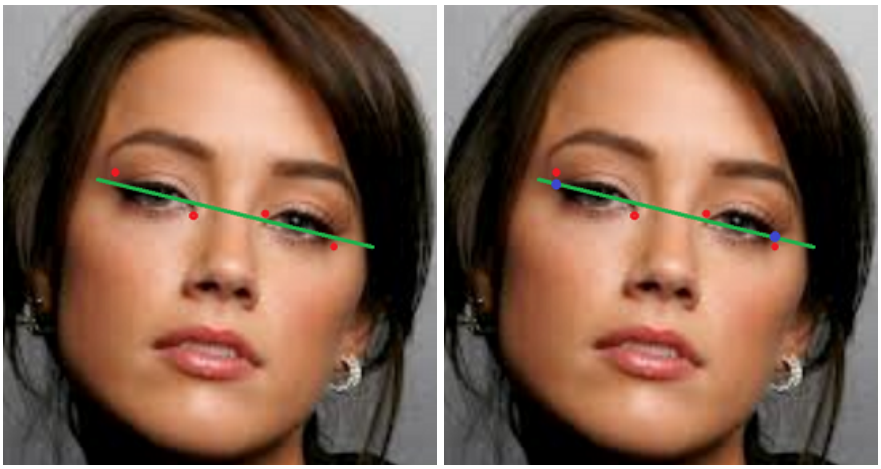
In the following chapter we introduce the, explored and implemented case study, which represents a distributed gallery, that enables users with a image sharing platform.

5

Case Study: Distributed Gallery

The explored case study aims at providing to its users a distributed gallery that collects, in an automatic way, every picture in which they appear in the context of a social gathering such as a birthday party or similar event. In the solutions that depend on the infrastructure (presented in Chapter 3), the image collecting system is materialized by the infrastructure that with the help of facial recognition algorithms, allows to identify each individual from the images registered to the server. In these prototypes, the client hands the captured photos to the server, so that the images can be processed, the individuals present in the photo be identified and properly indexed. The built media index, llows users to efficiently retrieve the images, by simply querying the server with their username. However, in the solution presented in section 4 the face recognition and the images indexation is performed, on the client side. Due to the fact, that there is no central index, the queries are executed in a different manner from the previous solutions, as it will be explained later in this section.

The reader should note that the goal of this case study (and its associated prototype implementations) is not to provide a full fledged mobile application. Instead its goal is to showcase the use and benefits of Aether and Aether+ architectures. Due to this, we have not addressed relevant issues as security and privacy, leaving these for future work. Additionally, we have employed of-the-shelf mechanisms, to perform face recognition. The optimizations of their use in mobile edge computing will also be addressed in future work.



(a) Linear regresssion line (b) Optimized points

Figure 5.1: Flandmark usage

5.1 Facial Detection and Recognition

The facial recognition algorithms used for the individual identification done in the server-side, are provided by the OpenIMAJ [HSD11] library. This library, is a compilation of libraries and tools for multimedia content analysis and generation, including state-of-the-art computer vision and advanced data clustering modules, which are used in the context of this work.

Unfortunately, facial detection and recognition algorithms do not work "out of the box" as they rely on machine learning training mechanisms. Due to this we resort to a train phase using collections of pictures,depicted as training sets, from each individual user of the system. For this case study, we employed a pattern recognition algorithm known as k-Nearest Neighbors classification algorithm (or k-NN for short) [Alt92] with a k=1 and a face recognition algorithm, named Fisherfaces [KB97] with 18 principal components (value obtained experimentally by trial and error). These algorithms, need at least 8 training images, with different face poses, for each individual user of the system, in order to obtain acceptable results.

On the client-side, the library used for facial recognition is OpenCV [Bra00], as it is more optimized for mobile devices than OpenIMAJ, that uses some libraries that are unsupported by the Android OS. OpenCV is very similar to OpenIMAJ, as it also provides tools for analysing and generating multimedia content as well as an API to manage the face recognition algorithms. OpenCVs face recognition API, is not as as simple as the one provided by the library used in the server-side, since it does not include automatic facial detection and alignment. These steps are crucial in facial recognition, due to the fact that pre-processing the images before they are analysed, greatly improves the recognition rate [DTL14]. The first phase, namely the facial detection phase, corresponds to the determining the ROIs (regions of interest), more specifically, to determining the minimum area

necessary to include, separately, each of the faces of the people who are in the photo. This is done by object detection using Haar feature-based cascade classifiers [VJ01], which is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images. Once the ROIs are delimited, each of the regions is cropped and saved. The second process, the face alignment, uses OpenCVs Flandmark (Facial Landmark) detector instead of cascade classifiers, to determine the position of both eyes, more specifically the corners. These four points are then used to create a line, using linear regression as seen on Figure 5.1a. This line allows to calculate two points (represented in blue in Figure 5.1b), that are more robust than using the detected ones, to find the necessary rotation to align the face horizontally, if required. Only after completing both processes, can the facial recognition begin, be it to train the algorithm or to identify an individual, while the system is running.

5.2 User Registration

The user registration process, is a crucial step in the setup of a new user, since the training images set is obtained through this mechanism, where:

1. A user chooses an unique username, that will work as an identifier;

2. Captures enough images from the devices camera with different poses and expressions, to obtain a rich training set, that allows the model to be trained to recognize him;

3. In the solutions that rely on the centralized component to perform the facial recognition, the training set is compressed and sent to the server, that in turn uses that data to train the recognition model. In contrast, in the semi-decentralized solution, the training set is stored in the device, due to the fact that the facial recognition is perofrmed locally in the mobile device

After the registration process is complete, the system is now able to be queried with that individuals information, more specifically his username.

5.3 Querying

The querying, or image collecting mechanism, also differs from one solution to the other. In the fully centralized and Aether (partially decentralized architectures), the queries are performed through a POST request to $/search$ with a "user" parameter, with the clients username. In contrast, the image collecting mechanism in the Aether+ architecture, works differently from the other solutions presented in this work, due to the fact that facial recognition is done locally rather than in a centralized component. Therefore,

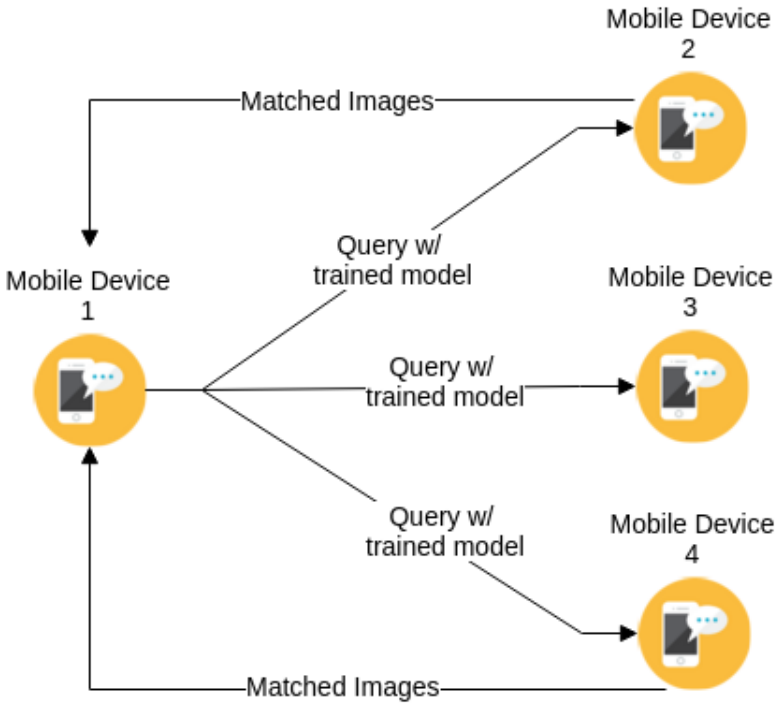


Figure 5.2: Query with the semi-decentralized architecture

there is no global face or trained model database, which increases the querying complexity. To overcome this challenge, the registration includes training the model with the users training set, converting the trained model to XML (so that later it can be parsed to a trained model) and compressing the obtained XML file. All these mentioned processes are done, so that the trained model can be sent to all the other users through the network and used to identify the user, that executed the image collection request, in the other users image set. However, this method has some limitations, introduced by the size of the trained model. Considering that, the facial recognition algorithm produces a $3.2MB$ trained model from a $120kb$ training set. Even though it is possible to compress the model in XML format to $1MB$, it still represents a considerable volume of data.

An example of the query mechanism logic is presented in Figure 5.2, where "Mobile Device 1" executes a query by broadcasting his trained model to the other users, so that they can use it to analyse their image collection and identify the user that placed the query request. After sweeping the images, with the recognition model, the pictures that match are then sent to the user, in this case "Mobile Device 1" receives images from "Mobile Device 2" and "Mobile Device 4".

Fallback Querying

This particular query mode, is only present in the solutions presented in this work that incorporate peer-to-peer communication, (i.e, Aether and Aether+) considering that when there is no connectivity between devices, it is not possible to query other users directly. In

that case, the query system resorts to the fallback mode, that handles the whole querying and data transfer process in a centralized way. In the centralized with peer-to-peer solution, the query process was already done in a centralized way, so in reality, this solution only uses the fallback mode to aid in the data transfer, and there is no need to adapt the communication model presented in Figure 3.4. However, in Aether+, the query process is not centralized, and because of this, a trained model of the user needs to be sent to other users. Since the GCM, does not support messages bigger than 1KB, the communication model presented in Figure 4.4 must suffer some slight adaptations.

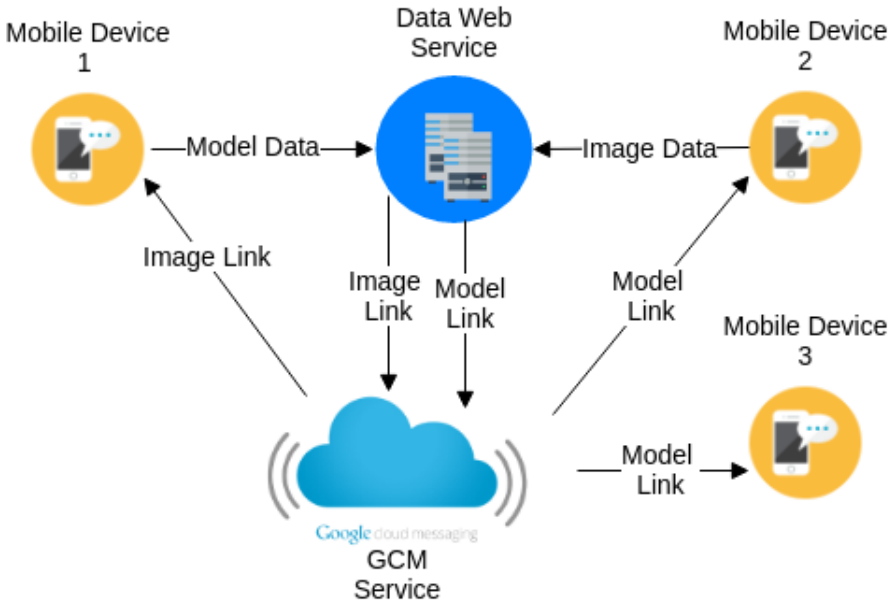


Figure 5.3: Fallback communication model adapted to gallery

The differences between Figure 4.4 and Figure 5.3, reside on the first step of the device that executed the query request, more specifically, in the transferring of the trained model data to the Data Web Service (executed by "Mobile Device 1"), that in turn generates a link to that model and broadcasts it through the GCM, starting the query process. When the other users ("Mobile Device 2" and "Mobile Device 3"), receive the model link, they download it and initiate the recognition process, that in this example only found matching results in "Mobile Device 2". This device, then sends the image to the Data Web Service, generating another link, but this time to the image file, which is sent to the device that issued the query, "Mobile Device 1", so that this last one can download the data.

5.4 File Indexing

The file indexing implementation differs in each solution, but they all answer the premise inherent to the case study, which is relating people to images. In the centralized solutions, this is done in the server, where a collection of image entities is maintained, this collection stores relevant data, such as the path on the file system or the identifier of the

devices that own that photo, as well as the people present in that image. Furthermore, it also maintains a set of user entities, each only holding the selected username but could also contain other personal information about the client. In the decentralized solution, there is no central index and therefore the index must be local and maintained by each device. This solution, requires analysing each photo that has people that have not already been identified with the obtained trained models from another users, received when they executed a broadcast. When a user is identified, this metadata is related with the image in question, in other words, a relationship between an image and an user is created or updated. This relationship, is then propagated when the images are sent, so that when a new query from another user is executed, it is not necessary to analyse all the images (which could incur in additional overhead).

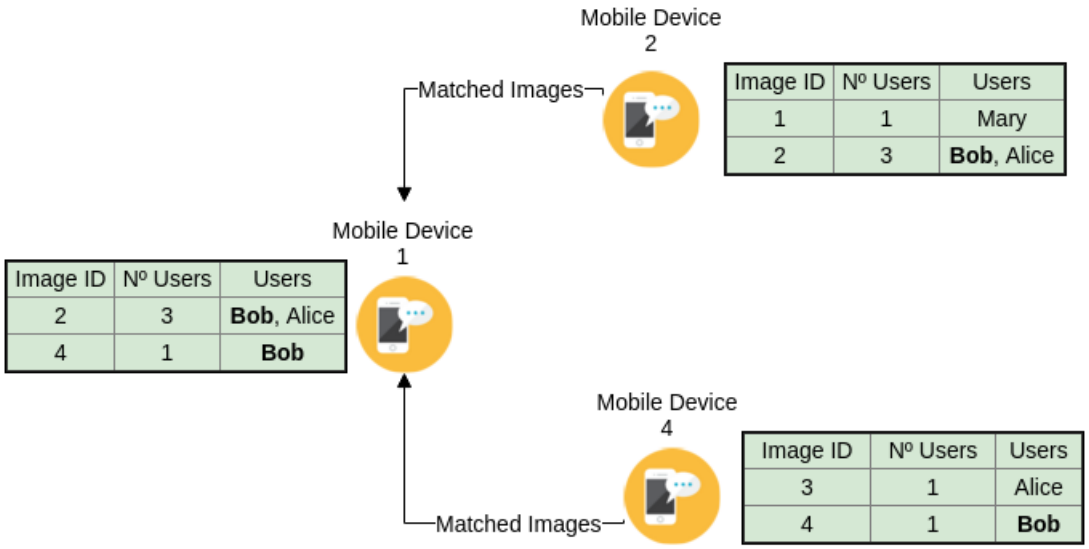
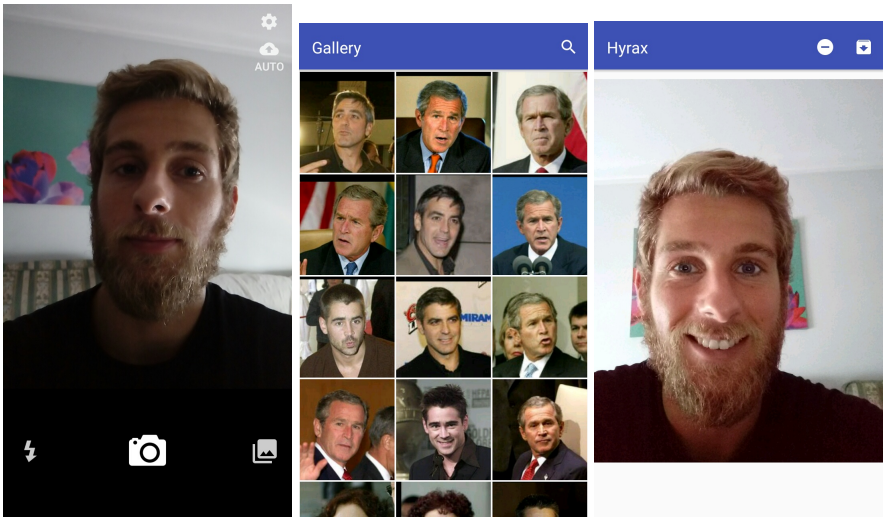


Figure 5.4: Local index update on image reception

The Figure 5.4 represents an example, where "Mobile Device 1" receives two images from separate devices. In this example, its possible to see that both mobile devices 2 and 4, own 1 photo of Bob (who is the user behind "Mobile Device 1"). When a photo is sent to the user that has executed the request, is accompanied by the respective index information, namely the image identifier, number of users and the username of already identified users in that image. This way, when "Mobile Device 1" receives a query request, he no longer has to verify the image with id 4, because it has only one individual (Bob). Furthermore, if the request is executed by Alice, the image with id 2 also avoids the recognition phase, due to the fact that $Alice$ has already been identified in that photo, and therefore it is already known that this one of the photos to be sent to $Alice$.



(a) Camera Module (b) Gallery (c) Image Fullscreen Mode

Figure 5.5: Interface Screenshots

5.5 Gallery Interface, Functions and Options

The gallery is composed by 2 main interface components, the image gallery (Figure 5.5b) and a camera module (Figure 5.5a) with which the users capture the pictures, and a auxiliary component, the login component (Figure 5.6a). By having a camera module separate from the phones native or third-party camera applications, its possible to control the pictures that are relevant and the ones that are supposed to be shared with other users. Furthermore, it allows adding custom action buttons, such as enabling/disabling the auto-upload of pictures to the server. The image gallery, displays the images thumbnails as well as the full picture. When in full picture mode (Figure 5.5c, its possible to download the image (if it is not present in the device already) and to remove a users identification from a picture, when there is a misidentification or if the user does not want that image to be shared with him. This identification removal, or untag, works by sending a POST request to $/untag$ with "user" and "image_id" as parameters. When the server receives the request he executes Algorithm 12.

It is in the untag mechanism (Algorithm 12) that the server tries to find the user based on the provided username and image identifier, and if the retrieval is successful, removes that relationship between that user and that image. In the Aether and Aether+ architecture, the untag function was not implemented.

The login component, allows users to select a username, with which they will be identified in the network, as well as providing a way of specifying the address of a remote server. In the centralized solutions, the usernames availability is verified by the server, in order to avoid duplicate users. Additionally, in the Aether+ solution, a view with the available groups is displayed, as seen in Figure 5.6b. This window, allows clients to choose which device to connect to, or if no device is within reach to enable fallback

Algorithm 12 Untag user from image

```
1: data: Images : images
2: output: Boolean
3: procedure UNTAG(user_id,image_id)
4:     user ← images.find(image_id).users.find(user_id)
5:     if user ≠ NULL then
6:         images.users.remove(user)
7:         return true
8:     else
9:         return false
10:    end if
11: end procedure
```

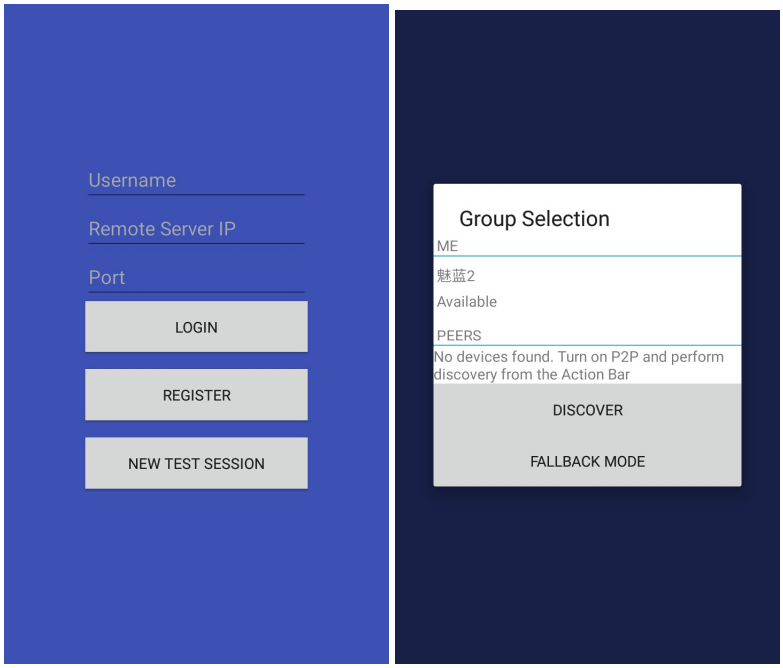
mode. The group selection window, only disappears by starting the camera module, when a group is joined or by explicitly selecting fallback mode.

The gallery also offers some performance options in terms of storage, more specifically enabling users to choose whether to cache all unique pictures in the devices storage or storing only the desired pictures, ignoring the ones that are not saved in when performing each search.

5.6 Summary

In this chapter, we present the necessary adaptations to the Aether and Aether+ architectures to support the proposed case study, a distributed gallery and some particularities from this application. Furthermore, we also describe and explain the face recognition techniques used to identify the users and specify the elements present in the galleries user interface.

The following chapter experimentally evaluated both proposed architectures (Aether and Aether+) and compares with the centralized architecture previously described. The case study presented in this chapter is also used to showcase the approach of proposed architectures.



(a) Login (b) Group Selection

Figure 5.6: Interface Screenshots

6

Evaluation

The solutions presented in Chapters 3 and 4 are evaluated in terms of latency, battery consumption, message volume (between devices, and through the centralized component). The latency and battery consumption results obtained in the experiments, serve to provide evidence to ground a efficiency comparison between centralized and hybrid architectures. The analysis of message volume (in bytes), allows to evaluate the feasibility and performance of each solution implementation. This chapter starts by describing implementation details and the used experimental setup, to perform our experimental evaluation.

6.1 Prototype Details

The prototype for each solution was developed in Java, more specifically the client-side consists of an Android (4.0+) application and the server-side is a *Web* service provider, based on the REST model and implemented resorting to Jersey [Cor16], which is a framework for developing RESTFul web services, running on Tomcat [Apa16]. For consistency, the used DBMS was Derby [Apa15], which is an in-memory relational database management system.

In order to connect to the server, the user has two options. One option consists of specifying the IP and Port of the web service, which is used when the centralized component is not in the same local network as the user. The other option consists of discovering a network service, that the server registered upon being deployed. This service provides the servers networking details, so that the user is able to connect server and interact with its REST interface.

6.1.1 Implementation Specifications and Statistics

Every prototype used Volley [Goo16b] library, which is the Android recommended library to efficiently handle and process web requests and an image loader library, Glide [Bum16], to perform the gallery image loading and rendering.

In terms of statistics, the server side java sources sum to a total of 2022 lines of code. The client side application java sources have a total of 7281 lines of code.

6.2 Experimental Setup

Every test, used the Amazon Services EC2 to materialize the centralized component for all solutions, more specifically we resorted to a EC2 t2.small instance in availability zone eu-central-1 (Frankfurt). To study the performance from the mobile clients perspective of the Aether (Chapter 3) and Aether+ (Chapter 4), the setup includes two Android (Lollipop) devices, with quad-core processors and $1Gb$ of RAM, that allow simultaneous WiFi and WiFi-Direct communication, and have Bluetooth support as well.

The experiments described below, resort to a set of training and testing images from 7 individuals that correspond to a sub-set of 300 images from the LFWTech collection [Hua+07], where the image average size is $12kb$ for the training images and $200kb$ for the testing images. Note that in the evaluation, the testing images were also $12kb$, due to the fact that the larger images were not available for all the individuals.

This experimental work main focus is split in two parts, the system comparison, where the three implemented solutions are compared, and system evaluation, where the proposed solution (Aether+) performance is analysed in more detail. The first part intends to answer the following relevant questions:

1. What is the transferred data volume between each actor in the system, more specifically, the amount of bytes and IP packets exchanged between clients and the centralized component, and client-to-client directly, in every solution? (section 6.3.1)

2. What is the observed latency from the clients point of view, when retrieving images, in each studied solution? (section 6.3.2)

3. What is the battery drainage from the clients device, when retrieving images, in the different architectures presented in this work? (section 6.3.3)

The second and last part, focuses on the Aether+ implementation and tries to answer the following questions:

1. What is the recognition rate from the algorithms used to perform facial recognition, when varying the number of training images and parameters? (section 6.4.1)

2. What is the transferred data volume (bytes and packets) between clients, when executing a query? (section 6.4.2)

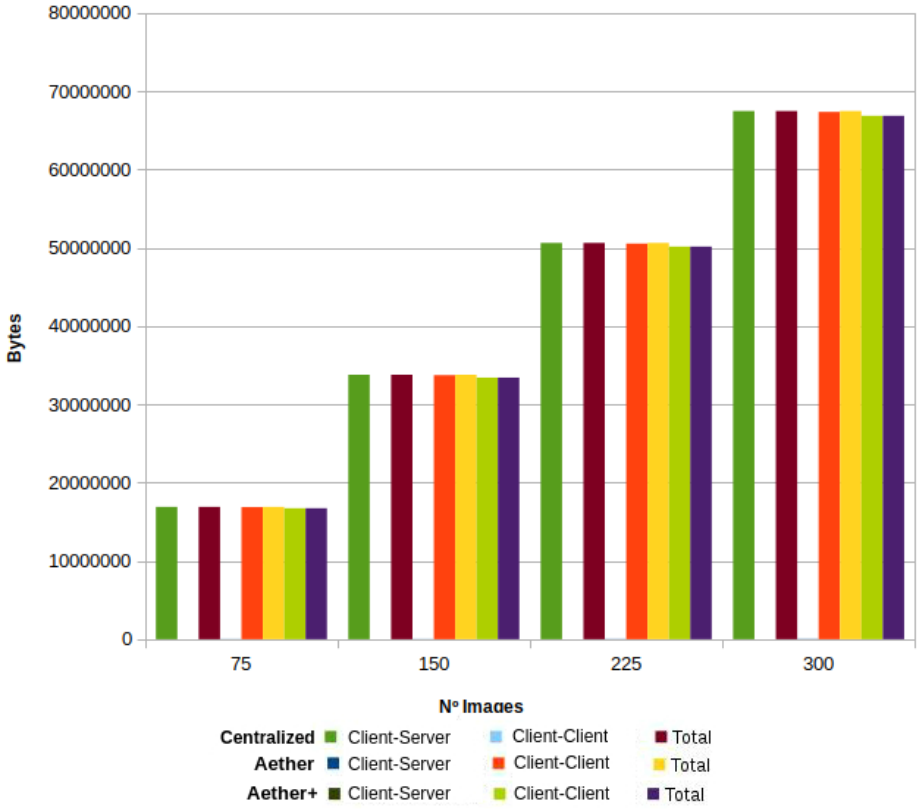


Figure 6.1: Comparison of the data volume exchanged between components

3. What is the registered latency for the whole process and more specifically, for the facial recognition algorithms execution? (section 6.4.3)

4. What is the energy usage from the query executioner and the reminder of the clients? (section 6.4.4)

6.3 Image Retrieval Comparison

In this experiment, we focus on measuring the necessary time, data volume, and battery usage from the mobile clients perspective, to execute a search and obtain (and display) all images. Note that, the facial recognition process is not taken into account in this experiment.

6.3.1 Data Transfer

To quantify the data transfer between the different system components of each solution, the volume of bytes and IP packets exchanged between client and server and among clients was measured (by prototype direct instrumentation and from inspecting the Android operating system counters).

The Figure 6.1 summarizes the results obtained under the form of a comparison between the obtained values for each solution (Aether and Aether+). In this experience the number of existing images in the system varied from 75 to 300.

As expected, in the centralized solution, there is no direct communication between clients, since every interaction among clients is mediated by the centralized component. Due to this fact, in this solution, every exchanged data involves client and server. In contrast, the data volume between client and server of the Aether solution, only represents a small fraction of the total (i.e., for 300 images it only represents 0.04% from all received and sent bytes). This fraction, corresponds to requests related to the retrieval of the image list and to central index updates. Since the image sending and negotiation process, regarding establishing a connection between mobile devices, is made through direct communication among clients, the majority of the registered data volume, corresponds to direct interaction between clients. Since all the communication in the Aether+ solution is made in a peer-to-peer fashion, the number of bytes between client and server is efficiently zero(in these experiments devices did not had to resort to the fallback strategy).

It is also noticeable, that the total amount of transferred data (regardless of the component responsible for the transfer) using the Aether and Aether+, is almost the same (regardless of the number of images in the system) when compared to the centralized solution. Typically, the solutions developed in this work that resort to peer-to-peer communication, consume about 1% to 2% less traffic than the centralized one. This is due to additional costs regarding messages from the servers REST interface. These results clearly show that the solutions proposed in this work, can achieve the objectives of minimizing the load induced in the centralized component and still use the same amount of transferred data between devices as the client-server solution. It is also noticeable that the Aether+ solution requires a slightly smaller amount of bytes than the other peer-to-peer solution (Aether) presented in this work.

The Figure 6.2 summarizes the comparison between the number of IP packets transferred between server and client for the (baseline) centralized and, the Aether and Aether+. As it can be seen, the amount of exchanged messages is inferior in the original Aether solution and non-existent in the Aether+ solution, which is consistent made previously.

6.3.2 Latency

In the centralized solution, the file transfer process latency, refers to the sum of all latencies from each image request made to the server. In the Aether architecture corresponds to the elapsed time between establishing a Bluetooth connection and obtaining all photos. Notice that the majority of the transfers used the WiFi-Direct technology in these experiments.

Figure 6.3a summarizes the obtained results from the experiment described above, that show the latency for the Aether solution being much higher. This discrepancy in the obtained results, can also be seen in Figure 6.3b, that represents the comparison of the

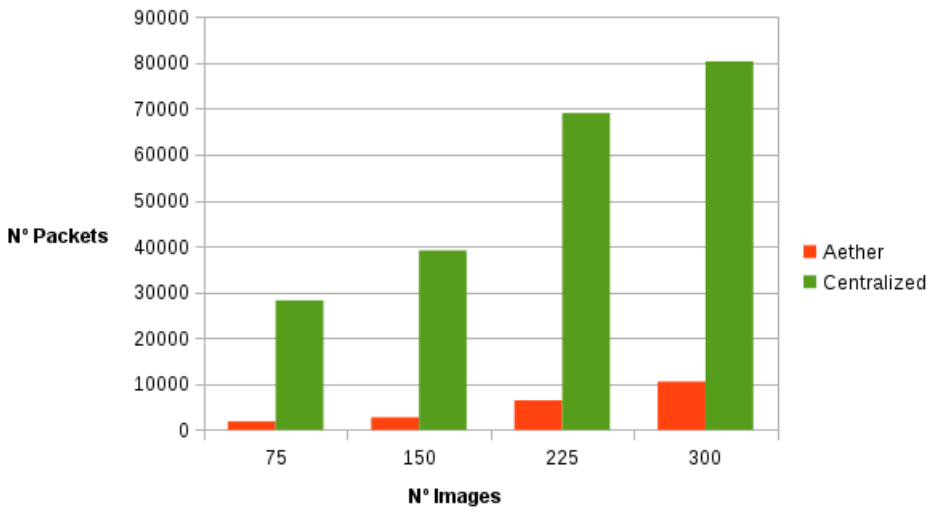
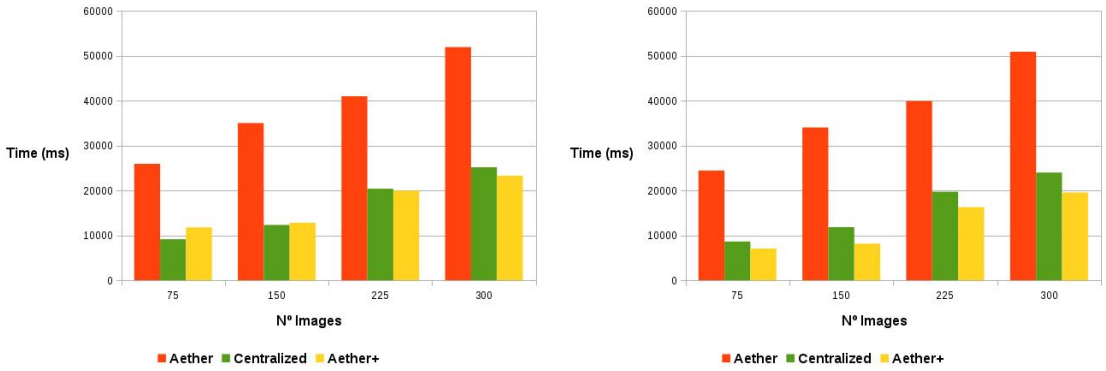


Figure 6.2: Comparison of the number of packets exchanged between client and server



(a) Total Process Latency

(b) Transfer Latency

Figure 6.3: Comparação de Pacotes IP e Latência

latency of the image transfer phase. The justification for these values, lies on the phase that precedes the image transfer, more specifically the stage where the client communication is prepared, which includes a negotiation phase (that may include negotiating a Bluetooth and a WiFi-Direct connection) that introduces additional latency to the whole process. The obtained latency values for the Aether+ solution, indicate that this solution has overcome these limitations by avoiding to constantly negotiating new data connections between devices , due to this, Aether+ registered lower transfer and, in some cases, overall latencies than its centralized counterpart. Additionally, this solution prioritizes WiFi-Direct and only resorts to Bluetooth when necessary, which further benefits latency due to the bandwith of WiFi-Direct in comparison with Bluetooth.

The results obtained for the total process and the transfer phase are very similar. This is due to the fact, that the image transfer is the most time consuming phase and the others represent a very small fraction of the elapsed time.

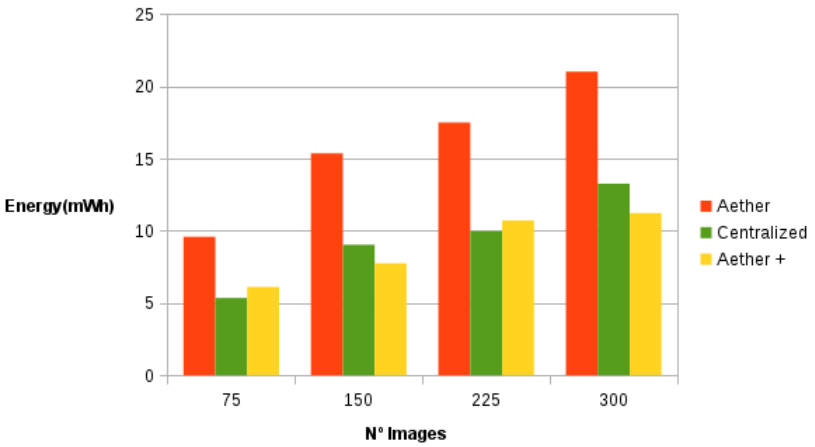


Figure 6.4: Comparison of the average battery usage

6.3.3 Battery Usage

To compare the battery usage of each solution, the prototypes energetic efficiency was registered using TrepN [Ben11]. TrepN is a profiling tool designed to measure an application effects on power, data and CPU. The measurement starts with the images search request and ends when all the images are loaded and displayed in the device.

Figure 6.4 presents a comparison between the energetic efficiency of each solution. By examining both centralized and Aether solutions, it is possible to conclude that the first uses less energy ($5mWh$ to $7mWh$) then the second. This can be explained by the way the architecture is designed and implemented. Since clients do not maintain a permanent connection to peers with which they communicate, and therefore need to establish a new connection whenever they need to communicate with a device, a process that includes negotiating a Bluetooth connection and, if the transfer is made through WiFi, using WiFi-Direct discovery mode (which is an expensive mechanism in terms of energy use, leading to high energy consumption). Furthermore, the Aether architecture also communicates with the server frequently. However, the Aether+ solution, in some cases, shows an inferior energy consumption when compared to the other two solutions, providing evidence that reinforces the fact that this solution maybe a viable alternative to the classic client-server model.

6.3.4 Fallback Mode

The fallback mode was also evaluated, even though some of the results cannot be fairly compared to the results obtained with the competing solutions, considering that this solution not only involves receiving the images in the clients device, but also the upload of those images to data web service by the other users. Nonetheless, the obtained results for the fallback mode were analysed in the same conditions as the other solutions.

The setup used in this experiment, is the same as the previous experiments presented

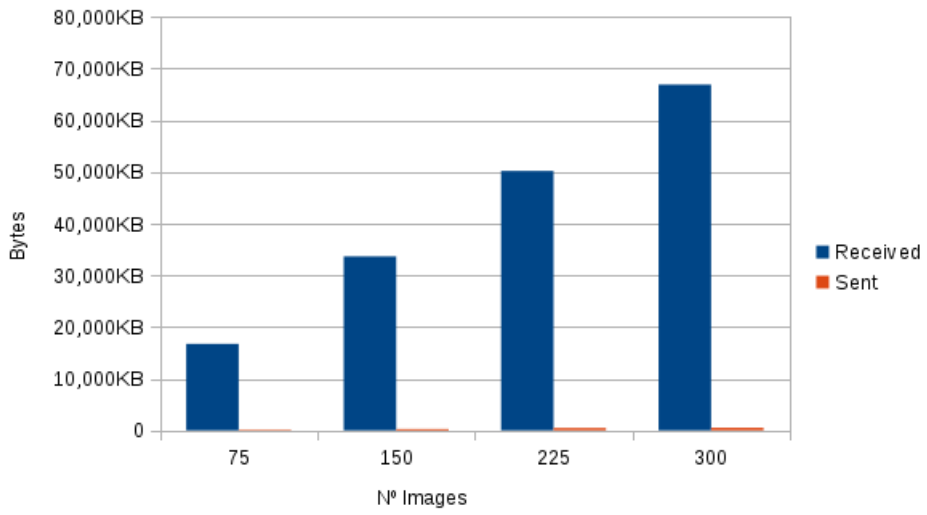


Figure 6.5: Received and sent bytes in fallback mode

in section 6.2.

Data Transfer

The transferred data volume results, in bytes, are presented in Figure 6.5, which are very similar to the ones registered in the other solutions. Furthermore, as expected the amount of received bytes is far superior than the number of bytes sent, since the experiment focuses on the receiver device. In terms of packets, this solution uses up to five times more packets than the others, as seen on Figure 6.6. This is justified by the fact that when an image transfer is in progress, the client must receive lots of packets through the GCM, considering that each image information is sent by this service, once the upload is complete. Additionally, this solution also exchanges packets related to the image data and details with the data web service.

Latency

The registered latency is one of the results that might not be compared adequately, since the latency when resorting to the fallback method includes the upload of the image to the data web service, which is inherent to the design of the architecture. In the other solutions, the upload to the server was performed previously when images are added to the system. However the observed latencies are presented in Figure 6.7. The observed latencies, are higher when compared to any of the other solutions, as a result of, both the implicit image upload latency and the necessary intermediary messages sent through the GCM.

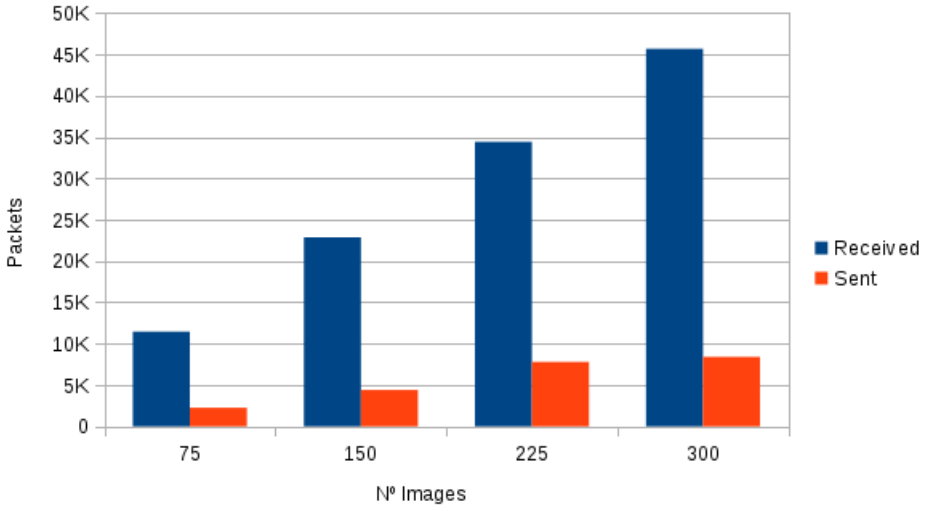


Figure 6.6: Received and sent packets in fallback mode

Battery Usage

The energetic efficiency of this solution, is also inferior when compared to any of the other solutions. The obtained results, Figure 6.8, were expected, considering that the fallback mode registered a superior number of packets and observed higher latencies, which means, that this solution not only requires more processing power to keep up with the communication but also to maintain the system running.

6.4 Query Execution Evaluation

This experiment registered and evaluated the performance of the gallery query mechanism, in terms of exchanged data volume, facial recognition accuracy, observed latency and battery usage. The obtained results correspond to the interval between the execution of the query request to the reception of all images relevant to the query.

6.4.1 Facial Recognition Performance

In this experiment, the facial recognition algorithms accuracy is evaluated in an image-restricted setting, meaning that only binary "matched" or "mismatched" classification is given to each test image. The evaluation is done by registering the amount of matches, true positive (TP), and mismatches, true negative (TN) identifications from the test images set and applying them to the following formula:

$$ACC = \frac{TP + TN}{TOTAL} \tag{6.1}$$

The results obtained show an accuracy of up to 81%, as shown in Figure 6.9, when using 100 images. Which is a satisfactory result when compared to values obtained with

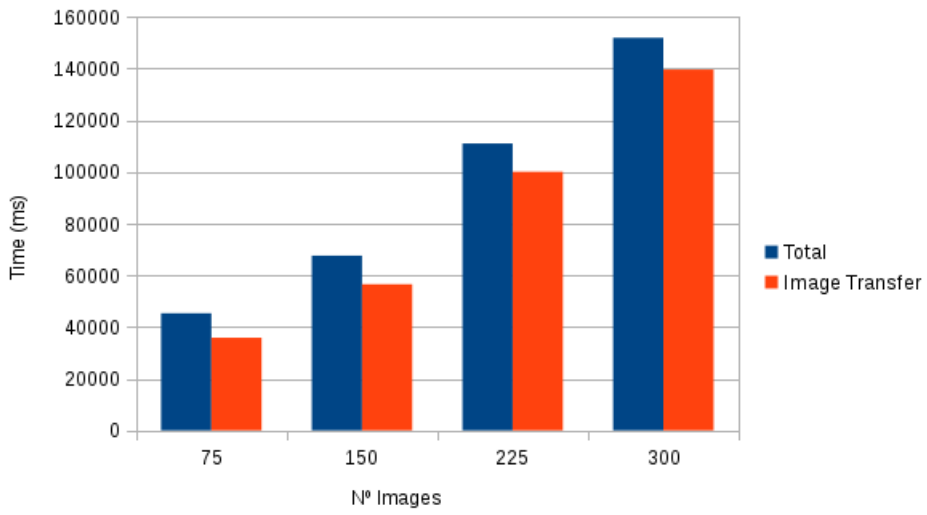


Figure 6.7: Observed Latencies in fallback mode

commercial recognition systems using the LFW collection, such as MERL [HJL08] and considering that the training images have a low resolution and were not staged for the purpose of facial recognition.

In order to compare the accuracy of the used facial recognition algorithm in different usages, we compiled a training set of 50 images and a test set of 140 images. These training images were then used to train the recognition model, in increments of 10 images. While maintaining a fixed number of 140 test images from 7 unique individuals where 84 correspond to the trained person and the rest to the other 6 people (about 10 pictures per person). Furthermore, the threshold (corresponding to an upper limit of the euclidean distance from the test image and the trained model) used to determine if a person is or is not in a certain picture was also varied, from 2600 to 2800.

Threshold	Avg. TP	Avg. TN	Avg. MisId	Avg. Acc (%)
2600	69.6	30.9	39.6	71.7
2700	75.0	27.8	37.2	73.4
2800	79.2	23.8	37.0	73.6

Table 6.1: Threshold comparison

Analysing the true positive, true negative and misidentifcation average rates, displayed in Table 6.1 helps to understand the trade-offs of each threshold. By choosing a stricter threshold the correct amount of negative recognitions increases but the positive one decreases, in contrast the broader threshold provides a higher positive count. Considering that the test set contains more images from the trained model individual than from the others, the average accuracy is higher with broader thresholds. Which indicates that, if there is knowledge of the test sets contents, it may be possible to benefit from this fact and increase accuracy by choosing a suiting threshold.

In Figure 6.9, its possible to observe that the a thresholds accuracy is not linear. Since

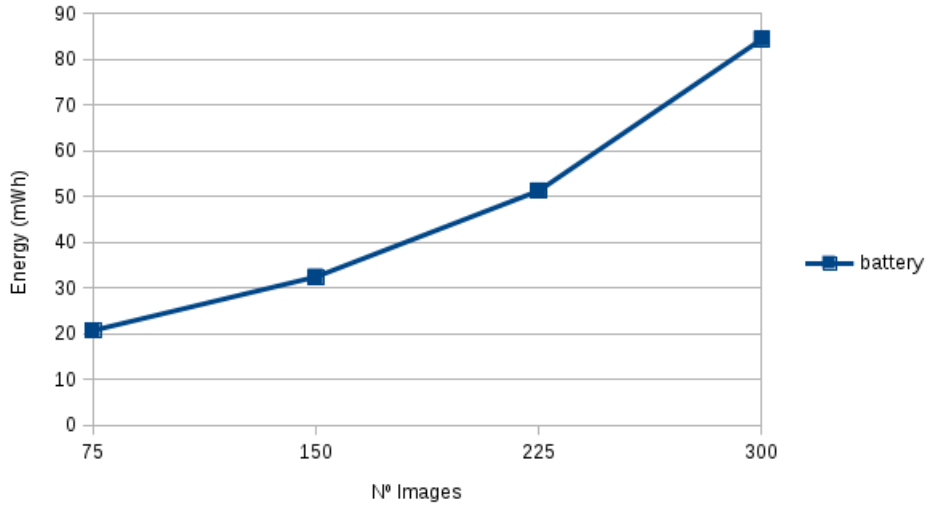


Figure 6.8: Average battery usage in fallback mode

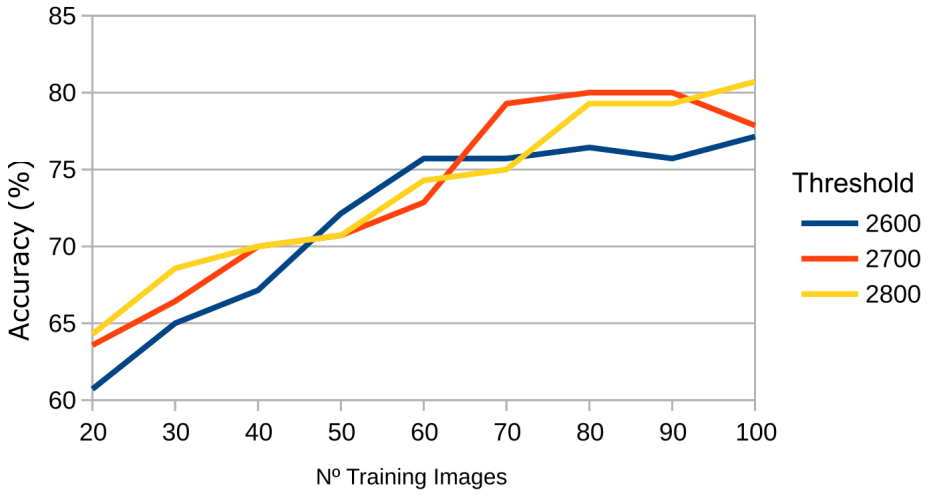


Figure 6.9: Accuracy comparison with different thresholds

it, fluctuates according to the number of training images. When the number of images is moderately low (20 to 50), the broader threshold (2800) provides better accuracy, while the stricter threshold (2600) maintains consistent until it stabilizes and the broader thresholds take over. This is justified by the fact, that certain training images characteristics such as lighting, facial pose or expression for example, can influence the model negatively and induce in error. Suggesting that a dynamic threshold, adapted to each situation, could provide an overall better accuracy.

These results show that the facial recognition mechanism employed in the design of our case study application is a viable approach that yields correct results with an acceptable accuracy.

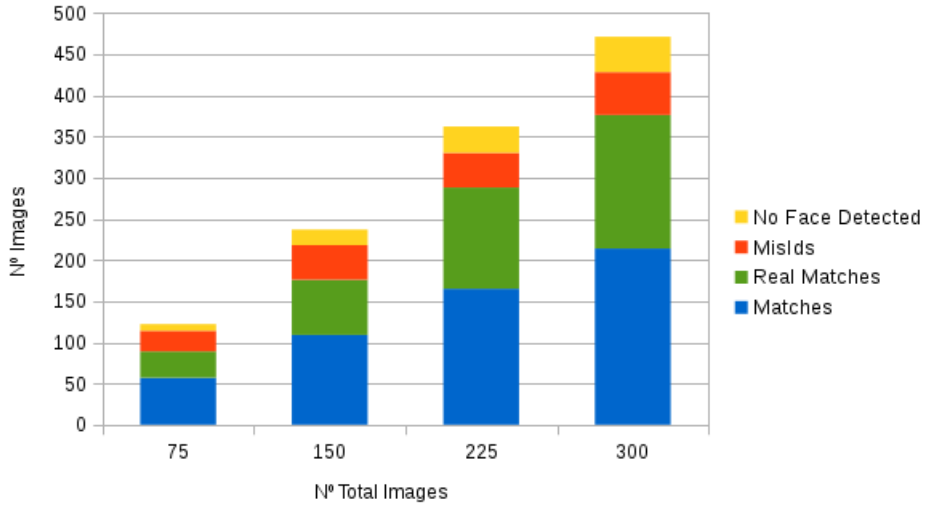


Figure 6.10: Facial recognition process results

6.4.2 Data Transfer

In this experiment, the exchanged data volume between the entities of the system, was registered, in form of bytes and packets. The number of images used varied from 75 to 300, however this does not indicate that the number of transferred images correspond to the total set, due to the fact that only the matched images are effectively transmitted.

Figure 6.10, presents the obtained values for the facial recognition process, more specifically, the total number of matches, which is the sum of the correct matches with number of incorrect matches, and the number of images where no face was detected. By analysing the results it is possible to conclude that about 70% of the images are sent to the receiver. These matched images are then sent to the respective user, which registered the received (RX) data volume presented in Table 6.2. This table also shows the sent (TX) data volume, which is very similar, independent from the number of images, which is justified by the size of the used trained model. The trained model consists of a compressed file with about $1.2Mb$, which is sent to other users, when a query request is made. Additionally, Table 6.3 represents the number of exchanged packets.

Nº Images	RX(Kb)	TX(Kb)	Total(Kb)
75	853	1240	2093
150	1644	1245	2889
225	2460	1251	3711
300	3201	1290	4490

Table 6.2: Received and sent bytes volume

Nº Images	RX	TX	Total
75	921	1041	1963
150	1430	1206	2637
225	1999	1255	3255
300	2492	1770	4262

Table 6.3: Received and sent packets volume

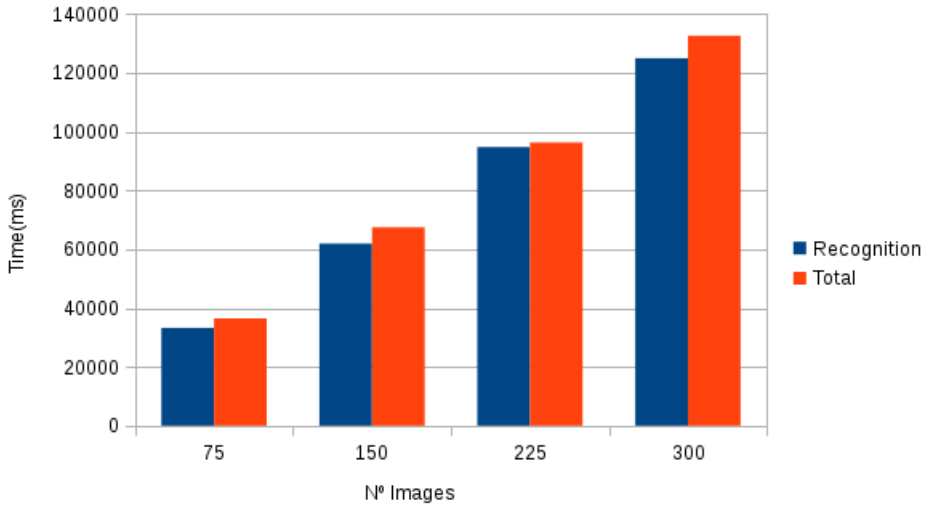


Figure 6.11: Total and recognition process latency

6.4.3 Latency

In this experiment, the time necessary for the whole query process was registered, taking special attention to the facial recognition phase latency.

Figure 6.11, shows the obtained latencies for each of the image sets. The majority of the time, is spent in the facial recognition process, because the images from the LFW collection have a low resolution and consequently the image transfer is very fast. However, the image transfer latencies has been already discussed in with more detail in section 6.3.2.

6.4.4 Battery Usage

This experiment registered the energy consumption on both involved actors, the receiver, corresponding to the user that executed the request, and the sender, corresponding to the user that replies to the query. In the receiver, the results were obtained in the interval between the execution of the query until the receipt of all photos. However, in the sender, this interval ranges from the reception of the query request to the dispatch of all recognized photos.

In Figure 6.12, its possible to observe the energy consumption of each actor. As expected the sender, consumes more energy than the receiver, due to the fact, that the first is responsible not only for the whole facial recognition process, but also for sending the

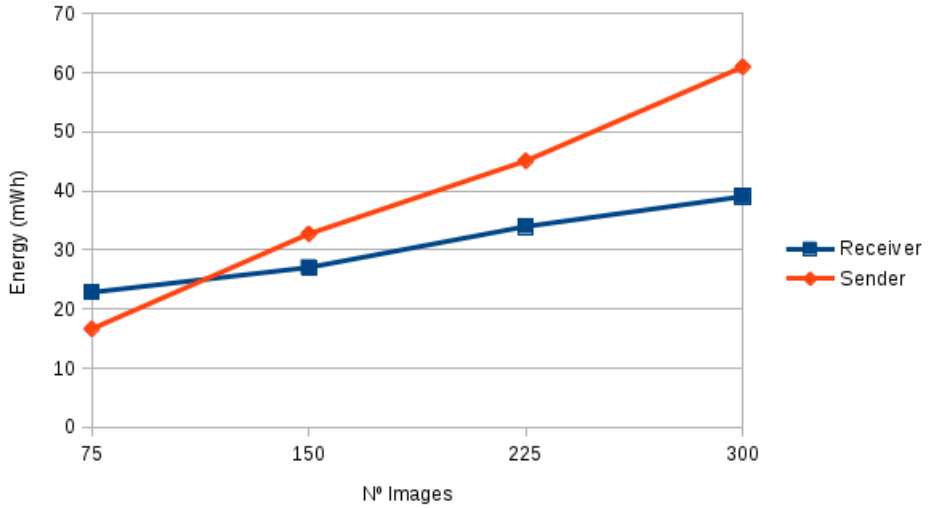


Figure 6.12: Average energy usage from the receiver and sender perspective

images, reaching about $61mWh$ with 300 photos.

Nº Images	CPU Load(%)
75	55.18
150	60.67
225	68.24
300	69.7

Table 6.4: Average CPU Load on the sender

For a matter of curiosity, the average CPU load on the sender was also registered, and the results are presented in Table 6.4. These results help justify the energy consumption values, considering that the CPU load varies from 55% to 70%, when the number of images varies from 75 to 300 respectively. This happens due to, the facial recognition phase, which very demanding in terms of processing power.

6.5 Summary

In this chapter, we present our evaluation and comparison of the prototypes, representing our case study and the presented architectures.

The prototypes, are evaluated in terms of transferred data volume, latency and battery usage. The registered results and observed latencies for each solution where then compared. Additionally, the performance of the used facial recognition algorithms was also evaluated and analysed.

The obtained results for this experiments, show promising results, regarding the Aether+ architecture as viable a alternative to centralized solutions.

7

Conclusions

File sharing between mobile device users, is an habit that is becoming increasingly common. Especially with multimedia content, such as photos and videos. However, this sharing usually is made through services that rely on an intermediary entity to index, search, and transfer those files, more specifically a centralized infrastructure. The infrastructure may have limited resources or the devices may not be able to establish a stable connection to it. Considering that in the mobile paradigm, the Internet connection may be intermittent or with low quality, due to the lack or overload of wireless access points, or due to the costs associated with 3G/4G data traffic, that usually is limited in most plans provided by telecommunications operators. Furthermore, these infrastructures represent a contention point on which all interaction between clients depends.

In this thesis, we have explored alternative architectures to the classic client-server that supports the sharing, querying and file transfer between mobile devices by combining the use of a central component with direct communication between users. This approach has the benefits of reducing the overall data traffic, especially between the clients and the server, as well as improving battery usage, while maintaining the same latencies as the centralized solution. This approach can also support client operation even in periods where the centralized infrastructure is not available, even though it might be limited to users that have low connectivity.

This work also includes the development of a case study application prototype based on Android, involving a distributed collaborative gallery, used for photo sharing in social events, such as concerts and parties, that leverages the partially-decentralized architecture presented in this thesis. This gallery features an automatic image collecting mechanism based on facial recognition, with which users are able to retrieve all the photos, in which they are present. The case study showcase the applicability of these alternative

architectures.

To evaluate the performance of both the architecture and the distributed gallery, the prototype works in three modes, representing three different architectures. A centralized mode, corresponding to the classic client-server that serves as a baseline to evaluate our work, a centralized with peer-to-peer communication for the file transfer and the partially-decentralized (Aether), that a decentralized network that uses the central component as fallback, when devices do not have connectivity between them (Aether+). Each of this solutions image retrieval mechanism were tested in terms of exchanged data volume, bytes and packets, between system actors, as well as in terms of latency and battery usage. Furthermore, the facial recognition performance was also evaluated. Additionally, the prototype mode using the partially-decentralized, Aether+, whole search process (from executing a query to receiving all the images) was also evaluated in terms of data volume, latency and battery consumption.

In summary, the main contributions of the work presented in this thesis are as follows:

Partially-Decentralized Architectures: an hybrid logical network topology designed to reduce the load imposed to the centralized component and reduce the users dependency to an Internet connection in file sharing systems, that consists of a decentralized network with a fallback support mode that resorts to a central component when there is no connectivity.

Distributed Photo Gallery: a crowd-sourced photo gallery populated by media retrieved from nearby devices, which extracts relevant features from this media by facial recognition and indexes it accordingly. Implemented as an Android application with three different network architectures.

Facial Recognition Querying: a feature that enables searching over a distributed photo gallery using facial recognition techniques, showcasing its feasibility in an mobile edge computing environment.

Publications

The work presented in this thesis was also accepted for presentation as a *Communication* in the track of Mobile and Distributed Computation (*Computação Móvel e Ubíqua*) at the Portuguese national conference INForum 2016.

Aether: Uma solução híbrida para a pesquisa e partilha de conteúdos em redes móveis (Comunicação Oral). André Sampaio, Nuno Preguiça e João Leitão in Simpósio de Informática (INForum), Lisboa, 2016

7.1 Future Work

In the course of this work, a number of directions for future improvements have been identified.

The results obtained for the partially-decentralized network meet the initial expectations, but could be improved even further and tested more thoroughly. One improvement could be introducing better negotiation when establishing group owners, based on resource availability or current battery. Furthermore, the tests could have used more devices to simulate bigger networks. The distributed gallery could also benefit from challenges not explored in this work, such as security and privacy. Additionally, the facial recognition querying has a limitation related with the size of the data transferred through the network when executing a query, due to the size of the trained recognition model. Furthermore, the facial recognition accuracy could be improved with better preprocessing and/or distributed facial recognition, by taking advantage of the resources provided by the devices to the network.

Bibliography

[Ahe+07] S. Ahern, D. Eckles, N. S. Good, S. King, M. Naaman, and R. Nair. "Over-exposed?" In: *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '07*. New York, New York, USA: ACM Press, Apr. 2007, p. 357. ISBN: 9781595935939. DOI: 10.1145/1240624.1240683. URL: http://dl.acm.org/citation.cfm?id=1240624.1240683.

[AA16] A. Ahmed and E. Ahmed. "A Survey on Mobile Edge Computing". In: JANUARY (2016). DOI: 10.13140/RG.2.1.3254.7925.

[Aky07] I. F. Akyildiz. *NETWORKING 2007. Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet: 6th International IFIP-TC6 Networking Conference, Atlanta, GA, USA, May 14-18, 2007, Proceedings*. Springer Science & Business Media, 2007, p. 1252. ISBN: 3540726055. URL: https://books.google.com/books?id=r4V2G7yPLIAC%7B%5C&%7Dpgis=1.

[Alt92] N. S. Altman. "An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression". In: *The American Statistician* 46.3 (1992), pp. 175–185. DOI: 10.1080/00031305.1992.10475879. eprint: http://www.tandfonline.com/doi/pdf/10.1080/00031305.1992.10475879. URL: http://www.tandfonline.com/doi/abs/10.1080/00031305.1992.10475879.

[Apa15] Apache. 2015. URL: https://db.apache.org/derby/.

[Apa16] Apache. 2016. URL: http://tomcat.apache.org/.

[BKN05] R. A. Baratto, L. N. Kim, and J. Nieh. "THINC: a virtual display architecture for thin-client computing". In: 39 (2005), pp. 277–290. URL: http://dl.acm.org/citation.cfm?id=1095837%20http://www.cs.unm.edu/%7B~%7Ddarnold/classes/papers/Baratto05THINC.pdf.

[Bec+14] M. T. Beck, M. Werner, S. Feld, and T. Schimper. "Mobile Edge Computing : A Taxonomy". In: *Afin* c (2014), pp. 48–54.

[Ben11] L. Ben-Zur. *Developer Tool Spotlight-Using Trepn Profiler for Power-Efficient Apps*. 2011.

[Bli16] Blizzard. 2016. URL: http://us.battle.net/d3/en/.

[Bon+12] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. "Fog Computing and Its Role in the Internet of Things". In: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing* (2012), pp. 13–16. ISSN: 978-1-4503-1519-7. DOI: 10.1145/2342509.2342513. URL: http://doi.acm.org/10.1145/2342509.2342513$%5Cbackslash$npapers2://publication/doi/10.1145/2342509.2342513.

[Bra00] G. Bradski. In: *Dr. Dobb's Journal of Software Tools* (2000).

[Bum16] Bumptech. *Glide an image loading and caching library for Android focused on smooth scrolling*. 2016. URL: https://github.com/bumptech/glide.

[Cor16] O. Corporation. 2016. URL: https://jersey.java.net/.

[Cor01] M. Corson. "Differential destination multicast-a MANET multicast routing protocol for small groups". English. In: *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*. Vol. 2. IEEE, 2001, pp. 1192–1201. ISBN: 0-7803-7016-3. DOI: 10.1109/INFCOM.2001.916314. URL: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=916314.

[CLN04] Y. Cui, B. Li, and K. Nahrstedt. "oStream: asynchronous streaming multicast in application-layer overlay networks". In: *Selected Areas in Communications, IEEE Journal on* 22.1 (2004), pp. 91–106.

[DD08] O. Das and A. Das. "Performability evaluation of mobile client-server systems". In: *Proceedings of the 2008 ACM symposium on Applied computing - SAC '08*. New York, New York, USA: ACM Press, Mar. 2008, p. 2197. ISBN: 9781595937537. DOI: 10.1145/1363686.1364210. URL: http://dl.acm.org/citation.cfm?id=1363686.1364210.

[DTL14] K. Dharavath, F. A. Talukdar, and R. H. Laskar. "Improving face recognition rate with image preprocessing". In: *Indian Journal of Science and Technology* 7.8 (2014), pp. 1170–1175.

[DB04] D. DJENOURI and N. BADACHE. "A Survey on Security Issues in Mobile Ad hoc Networks Djamel". In: February (2004).

[FLR13] N. Fernando, S. W. Loke, and W. Rahayu. "Mobile cloud computing: A survey". In: *Future Generation Computer Systems* 29.1 (Jan. 2013), pp. 84–106. ISSN: 0167739X. DOI: 10.1016/j.future.2012.05.023. URL: http://www.sciencedirect.com/science/article/pii/S0167739X12001318.

[Fra00] P. Francis. *Yoid: Extending the internet multicast architecture*. 2000.

[Fre15] Freenet. *Freenet is free software which lets you anonymously share files, browse and publish "freesites" (web sites accessible only through Freenet) and chat on forums*. 2015. URL: https://freenetproject.org/.

[Gam10] S. Gammeter. "Server-side object recognition and client-side object tracking for mobile augmented reality". In: *Cvprw* C (2010), pp. 1–8. DOI: 10.1109/CVPRW.2010.5543248.

[Goo16a] Google. 2016. URL: https://developers.google.com/cloud-messaging/gcm.

[Goo16b] Google. *Android Volley*. 2016. URL: https://android.googlesource.com/platform/frameworks/volley.

[Gud+13] A. Gudipati, D. Perry, L. E. Li, and S. Katti. "SoftRAN: Software Defined Radio Access Network". In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*. HotSDN '13. Hong Kong, China: ACM, 2013, pp. 25–30. ISBN: 978-1-4503-2178-5. DOI: 10.1145/2491185.2491207. URL: http://doi.acm.org/10.1145/2491185.2491207.

[GSB02] M. Günes, U. Sorges, and I. Bouazizi. "ARA-the ant-colony based routing algorithm for MANETs". In: *Parallel Processing Workshops, 2002. Proceedings. International Conference on*. IEEE. 2002, pp. 79–85.

[HHL06] Z. Haas, J. Halpern, and L. Li. "Gossip-based ad hoc routing". In: *IEEE/ACM Transactions on Networking (ToN ...* (2006), pp. 1–12. ISSN: 1063-6692. DOI: 10.1109/TNET.2006.876186. arXiv: 0209011v1 [arXiv:cs]. URL: http://dl.acm.org/citation.cfm?id=1143399.

[HSD11] J. S. Hare, S. Samangooei, and D. P. Dupplaw. "OpenIMAJ and ImageTerrier: Java Libraries and Tools for Scalable Multimedia Analysis and Indexing of Images". In: *Proceedings of the 19th ACM International Conference on Multimedia*. MM '11. Scottsdale, Arizona, USA: ACM, 2011, pp. 691–694. ISBN: 978-1-4503-0616-4. DOI: 10.1145/2072298.2072421. URL: http://doi.acm.org/10.1145/2072298.2072421.

[HJL08] G. B. Huang, M. J. Jones, and E. Learned-Miller. "LFW Results Using a Combined Nowak Plus MERL Recognizer". In: *Workshop on Faces in 'Real-Life' Images: Detection, Alignment, and Recognition*. Erik Learned-Miller and Andras Ferencz and Frédéric Jurie. Marseille, France, Oct. 2008. URL: https://hal.inria.fr/inria-00326728.

[Hua+07] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller. *Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments*. Tech. rep. 07-49. University of Massachusetts, Amherst, Oct. 2007.

[Jac+01] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot. "Optimized link state routing protocol for ad hoc networks". In: *Ieee Inmic 2001: Ieee International Multi Topic Conference 2001, Proceedings: Technology for the 21St Century* (2001), pp. 62–68. DOI: 10.1109/INMIC.2001.995315. URL: http://apps.webofknowledge.com/full%7B%5C_%7Drecord.do?product=UA%7B%5C&%7Dsearch%7B%5C_%7Dmode=GeneralSearch%7B%5C&%7Dqid=2%7B%5C&%7DSID=1A5x8KrqG7PGKaNLOLm%7B%5C&%7Dpage=1%7B%5C&%7Ddoc=2.

[JGZ03] S. Jiang, L. Guo, and X. Zhang. "LightFlood: An efficient flooding scheme for file search in unstructured peer-to-peer systems". In: *Proceedings of the International Conference on Parallel Processing* 2003-January.5 (2003), pp. 627–635. ISSN: 01903918. DOI: 10.1109/ICPP.2003.1240631.

[JHE99] J. Jing, A. S. Helal, and A. Elmagarmid. "Client-server computing in mobile environments". In: *ACM Computing Surveys* 31.2 (1999), pp. 117–157. ISSN: 03600300. DOI: 10.1145/319806.319814.

[JNA08] D. Johnson, N. Ntlatlapa, and C. Aichele. *Simple pragmatic approach to mesh routing using BATMAN*. en. Oct. 2008. URL: http://researchspace.csir.co.za/dspace/handle/10204/3035.

[Joh+03] D. B. Johnson, D. A. Maltz, Y.-C. Hu, and J. Jetcheva. "The dynamic source routing (DSR) protocol for mobile ad hoc networks". In: *IETF Draft, draft-ietf-manet-dsr-009. txt* (2003).

[KBK] G. Kalic, I. Bojic, and M. Kusek. "Energy Consumption in Android Phones when using Wireless Communication Technologies". In: ().

[Kir03] P. Kirk. *Gnutella - A protocol for a Revolution*. 2003. URL: http://rfc-gnutella.sourceforge.net/developer/stable/index.html.

[KB97] D. J. Kriegman and P. N. Belhumeur. "Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection ~". In: 19.7 (1997), pp. 711–720.

[Lei12] J. Leitão. "Topology Management for Unstructured Overlay Networks". PhD thesis. 2012.

[LR14] J. C. A. Leitão and L. E. T. Rodrigues. "Overnesia: a resilient overlay network for virtual super-peers". In: *Reliable Distributed Systems (SRDS), 2014 IEEE 33rd International Symposium on*. IEEE. 2014, pp. 281–290.

[LPR10] J. Leitao, J. Pereira, and L. Rodrigues. "Gossip-based broadcast". In: *Handbook of Peer-to-Peer Networking*. Springer, 2010, pp. 831–860.

[LPR07] J. Leitão, J. Pereira, and L. Rodrigues. "HyParView: A membership protocol for reliable gossip-based broadcast". In: *Proceedings of the International Conference on Dependable Systems and Networks* (2007), pp. 419–428. DOI: 10.1109/DSN.2007.56.

[LDT07] P. Lu, R. Di, and F. Tr. "Epidemic Broadcast Trees". In: (2007).

[Lun00] J. Lundberg. "Routing security in ad hoc networks". In: *Helsinki University of Technology* (2000), pp. 1–12. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.144.7589.

[Lv+02] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. "Search and replication in unstructured peer-to-peer networks". In: *ACM SIGMETRICS Performance Evaluation Review* 30.1 (2002), p. 258. ISSN: 01635999. DOI: 10.1145/511399.511369.

[Mah12] A. Maheshwari. "Hybrid Approach of Client-Server Model and Mobile Agent Technology to Drive an E-Commerce Application". In: 2.4 (2012), pp. 733–738.

[Mar09] E. E. Marinelli. "Hyrax : Cloud Computing on Mobile Devices using MapReduce". In: *Science* 0389.September (2009), pp. 1–123.

[Mic16] Microsoft. 2016. URL: https://www.office.com/.

[Mir07] H. Miranda. "Gossip-based Data Distribution in Mobile Ad Hoc Networks". MA thesis. Faculdade de Ciências da Universidade de Lisboa, June 2007.

[MLR06] H. Miranda, S. Leggio, and K. Raatikainen. "A power-aware broadcasting algorithm". In: *The 17th Annual IEEE International Symposium on Personal, Indoor andMobile Radio Communications (PIMRC'06)* (2006).

[Mon09] D. Monica. "Thwarting the sybil attack in wireless ad hoc networks". In: *Master's Thesis at the Universidade Tecninca de Lisboa* (2009).

[Pat+14] H. M. Patel, Y. Hu, P. Hédé, I. B. M. J. Joubert, C. Thornton, B. Naughton, I. Julian, R. Ramos, C. Chan, V. Young, S. J. Tan, and D. Lynch. "Mobile-Edge Computing". In: 1 (2014), pp. 1–36.

[Pat01] L. Patrick. 2001. URL: http://www.napster.com/.

[Per+06] T. Pering, Y. Agarwal, R. Gupta, and R. Want. "CoolSpots: Reducing the Power Consumption of Wireless Mobile Devices with Multiple Radio Interfaces". In: *Proceedings of the 4th International Conference on Mobile Systems, Applications and Services*. MobiSys '06. Uppsala, Sweden: ACM, 2006, pp. 220–232. ISBN: 1-59593-195-3. DOI: 10.1145/1134680.1134704. URL: http://doi.acm.org/10.1145/1134680.1134704.

[Rip01] M. Ripeanu. "Peer-to-peer architecture case study: Gnutella network". In: *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*. IEEE. 2001, pp. 99–100.

[RD01] A. Rowstron and P. Druschel. "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems". In: *Middleware 2001* 2218.November 2001 (2001), pp. 329–350. ISSN: 03029743. DOI: 10.1007/3-540-45518-3. URL: http://www.springerlink.com/index/10.1007/3-540-45518-3.

[Sch01] R. Schollmeier. "A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications". In: *Proceedings First International Conference on Peer-to-Peer Computing* (2001), pp. 2–3. ISSN: 1479-5876. DOI: 10.1109/P2P.2001.990434.

[SWS12] B. D. Shivahare, C. Wahi, and S. Shivhare. "Comparison Of Proactive And Reactive Routing Protocols In Mobile Adhoc Network Using Routing Protocol Property :" in: *International Journal of Emerging Technology and Advanced Engineering* 2.3 (2012), pp. 356–359.

[Sin01] A. Singla. 2001. URL: http://rfc-gnutella.sourceforge.net/Proposals/Ultrapeer/Ultrapeers.htm.

[Sto+01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications". In: *Conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '01)* (2001), pp. 149–160. ISSN: 01464833. DOI: 10.1145/383059.383071. URL: http://portal.acm.org/citation.cfm?doid=383059.383071.

[Su+09] B. Su, H. Yu, Z. Ma, C. Yang, and Y. Zhu. "Research on Overlay Multicast Routing Protocols for Mobile Ad Hoc Networks". In: *2009 5th International Conference on Wireless Communications, Networking and Mobile Computing* (2009), pp. 1–4. DOI: 10.1109/WICOM.2009.5302328. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5302328.

[TP03] E. T. Clausen and E. P. Jacquet. "Optimized Link State Routing Protocol (OLSR)". In: (2003). ISSN: 2070-1721. URL: http://www.rfc-editor.org/info/rfc3626.

[Tan96] A. S. Tanenbaum. *Computer Networks*. Vol. 52. 169. 1996, pp. 349–351. ISBN: 0130661023. DOI: 10.1016/j.comnet.2008.04.002. URL: http://www.ietf.org/rfc/rfc169.txt.

[Teó+15] A. Teófilo, D. Remédios, H. Paulino, and J. Lourenço. "Group-to-Group Bidirectional Wi-Fi Direct Communication with Two Relay Nodes". In: *Proceedings of the 12th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services on 12th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*. MOBIQUITOUS'15. Coimbra, Portugal: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2015, pp. 275–276.

ISBN: 978-1-63190-072-3. DOI: 10.4108/eai.22-7-2015.2260272. URL: http://dx.doi.org/10.4108/eai.22-7-2015.2260272.

[Tse+02] Y.-c. Tseng, S.-y. Ni, Y.-s. Chen, and J.-p. Sheu. "The Broadcast Storm Problem in a Mobile Ad Hoc Network". In: (2002), pp. 153–167.

[VJ01] P. Viola and M. Jones. "Rapid object detection using a boosted cascade of simple features". In: *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. Vol. 1. IEEE. 2001, pp. I–511.

[WZM10] L. Wang, D. Zhao, and L. Ming. "An Energy Efficient WLAN Skype Deployment Using GSM Wakeup Signals". In: *Proceedings of the 2010 IEEE/ACM Int'L Conference on Green Computing and Communications & Int'L Conference on Cyber, Physical and Social Computing*. GREENCOM-CPSCOM '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 470–473. ISBN: 978-0-7695-4331-4. DOI: 10.1109/GreenCom-CPSCom.2010.79. URL: http://dx.doi.org/10.1109/GreenCom-CPSCom.2010.79.

[YLL15] S. Yi, C. Li, and Q. Li. "A Survey of Fog Computing". In: *Proceedings of the 2015 Workshop on Mobile Big Data - Mobidata '15*. New York, New York, USA: ACM Press, June 2015, pp. 37–42. ISBN: 9781450335249. DOI: 10.1145/2757384.2757397. URL: http://dl.acm.org/citation.cfm?id=2757384.2757397.

[YCM12] X. Yong, D. Chi, and G. Min. "The Topology of P2P Network". In: *Journal of Emerging Trends in Computing and Information Sciences* 3.8 (2012), pp. 1213–1218.

[ZG96] J. Zhou and D. Gollman. *A fair non-repudiation protocol*. IEEE, 1996.