Universidade de Aveiro
2010
Departamento de Electrónica, Telecomunicações e Informática

CARLOS ANDRÉ MARQUES VIANA FERREIRA

REDE PEER-TO-PEER PARA IMAGEM MÉDICA

PEER-TO-PEER NETWORK FOR MEDICAL IMAGING

"In all chaos there is a cosmos, in all disorder a secret order."

− Carl Jung

Universidade de Aveiro
2010
Departamento de Electrónica, Telecomunicações e Informática

CARLOS ANDRÉ MARQUES VIANA FERREIRA

REDE PEER-TO-PEER PARA IMAGEM MÉDICA

PEER-TO-PEER NETWORK FOR MEDICAL IMAGING

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Dr. Carlos Manuel Azevedo Costa, Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri

presidente

Prof. Doutor Joaquim Arnaldo Carvalho Martins
Professor Catedrático do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Prof. Doutor Carlos Manuel Azevedo Costa
Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Prof. Doutor Rui Pedro Sanches de Castro Lopes
Professor Coordenador do Departamento de Informática e Comunicações do Instituto Politécnico de Bragança

agradecimentos

Agradeço à minha família pelo apoio constante.
Também agradeço a todo o grupo de bioinformática, no qual estive inserido durante o último ano, pelo bom ambiente e cooperação, tornando todo o processo de investigação e desenvolvimento mais agradável. Em especial, ao Luís Ribeiro, prof. Carlos Costa e prof. José Luís Oliveira pela incansável cooperação e paciência.

acknowledgements

I thank to my family for the constant support.
I also acknowledge the whole bioinformatics group (in which I was involved during the last year) for the good atmosphere and cooperation, turning the investigation and development processes much more pleasant. Especially, for Luís Ribeiro, prof. Carlos Costa and prof. José Luís Oliveira for the tireless cooperation and patience.

palavras-chave

Imagem médica, PACS, DICOM , Peer-to-Peer, Cloud Computing, Motor de Busca.

resumo

Nos últimos anos, a imagem médica em formato digital tem sido uma ferramenta cada vez mais importante quer para o diagnóstico médico quer para o auxílio ao tratamento. Assim, equipamentos de aquisição digital e repositórios de imagem médica são cada vez mais comuns em instituições de saúde, podendo até haver mais que um repositório numa instituição. No entanto, esta proliferação de repositórios leva a que a informação esteja dispersa nos vários locais. Esta dispersão da informação juntamente com as diferenças no armazenamento entre instituições são claros obstáculos à pesquisa e acesso integrado a essa informação. Esta dissertação visa o estudo da tecnologia Peer-to-Peer de forma a minimizar os problemas associados à dispersão e heterogeneidade da informação.

keywords

Medical Imaging, PACS, DICOM , Peer-to-Peer, Cloud Computing, Search Enigne.

abstract

In the last years, digital medical imaging has been an increasingly important tool for both medical diagnostic and treatment assistance. Therefore, digital image acquisition equipments and medical imaging repositories are more and more common in a healthcare institution, being possible even more than one repository in one institution. However, this proliferation of repositories leads to dispersion of data between many places. This data dispersion associated with differences in the data storage between institutions are evident obstacles to the search for medical data. This dissertation aims to the study of the Peer-to-Peer technology in order to minimize the problems related to the dispersion and heterogeneity of medical data.

Table of Contents

Images

Glossary

API	-	Application Programming Interface
ARPANET	-	Advanced Research Projects Agency Network
CAN	-	Content Addressable Network
CPU	-	Central Processing Unit
CT	-	Computed Tomography
C/S	-	Client – Server
DHT	-	Distributed Hash Table
DICOM	-	Digital Image and Communication in Medicine
DoS	-	Denial of Service
EC2	-	Elastic Compute Cloud
FLOPS	-	Floating point Operations per Second
GAE	-	Google App Engine
GUI	-	Graphical User Interface
HIS	-	Hospital Information System
HTTP	-	Hypertext Transfer Protocol
ID	-	Identification
IOD	-	Information Object Definition
JVM	-	Java Virtual Machine
LSB	-	Less Significant Bit
MP3	-	MPEG1 Audio Layer 3
MRI	-	Magnetic Resonance Imaging
MSB	-	Most Significant Bit
OS	-	Operating System
OSI	-	Open System Interconnection Reference Model
P2P	-	Peer-to-Peer
PaaS	-	Platform as a Service
PACS	-	Picture Archiving and Communication System
PET	-	Positron Emission Tomography
QRP	-	Query Routing Protocol
RIS	-	Radiology Information System
SPECT	-	Single Photon Emission Computed Tomography
SQL	-	Structured Query Language
SR	-	Structured Report
TCP	-	Transmission Control Protocol
TCP/IP	-	Internet protocol suite
TLV	-	Tag-Length-Value
TTL	-	Time To Live
UDP	-	User Datagram Protocol

WADO	-	Web Access to DICOM Persistent Objects
VoIP	-	Voice over Internet Protocol
VPN	-	Virtual Private Network
XML	-	Extensible Markup Language
XMPP	-	Extensible Messaging and Presence Protocol

1 Introduction

The use of digital medical imaging systems in healthcare institutions has significantly increased, constituting a valuable tool to support medical profession, both in decision support and in treatment procedures. In the last decades, medical imaging has been taking advantage of the evolution of computer technology, for example with the introduction of new computerized imaging modalities [1].

Until the late 1980s, the equipments of digital acquisition were treated as isolated systems (each isolated system was composed by the acquisition equipment, a workstation and a printer). However, the processes of digital image acquisition began to be used at a larger scale, what brought the necessity of getting together the isolated systems into a single major system. Therefore, Picture Archiving and Communication System (PACS) concept appeared [2].

"A picture archiving and communication system consists of image and data acquisition, storage, and display subsystems integrated by digital networks and application software" [3]. In other words, PACS gather a set of technologies of acquisition, storage, visualization and distribution of medical image and data through a computer network [4].

PACS almost solved the problem of the data exchange inside a hospital (or clinic). However, the problem of communication between health care centers still remained.

Since a health care center does not possess all the different modalities that sometimes are needed to diagnose a patient, the patients need to visit other centers in order to be undergone to the necessary exams. Consequently, all images and associated data will be situated in different places and it is important to exchange those images and that data between the health care centers so that physicians would have access to all the necessary exams. Nowadays, this exchange is made through email, traditional mail, patient or virtual private networks (VPN) [4]. However, none of these solutions really fulfils all the needs of the medical environment. Therefore, this thesis proposes the peer-to-peer (P2P) technology to solve the communication problems of the medical environment.

P2P got well known with the file sharing applications, where users can where users can share data easily and quickly [5]. P2P also solves the scalability problems of the client-server architecture, allowing the provision of a service in a resilient, scalable and theoretically unlimited way. These characteristics of the services provided by the P2P technology are important requirements in a system responsible for communication in a medical environment. Besides, this technology allows the information retrieval from distributed repositories.

This thesis proposes a P2P architecture and implementation that is able to provide a medical image data search, where all nodes participate in the construction of the results list. Besides, the network also supports file transfers, making the medical data available when and where it is needed.

2 Medical Imaging and Peer-to-Peer

In this chapter, it is presented an overview of the medical environment and the Peer-to-Peer networks.

2.1 Medical Environment

Medical imaging is a technique and process of visual representations acquisition of tissues and organs. Nowadays, it plays a very important role in medical environment, being one of the most important tools for diagnosis and having an increasingly important role in treatment.

The increased importance of medical image in diagnosis and treatment, in the last decades, has followed the tendency of the availability of computational resources that brought new imaging tools such as: computed tomography (CT), magnetic resonance imaging (MRI), Doppler ultrasound, positron emission tomography (PET) and single photon emission computed tomography (SPECT) [6]. Besides the fact that computers are creating methods of acquiring medical images, they are also changing the storage, viewing and communication of images and other medical data [3].

The following sub-items present an overview of some systems and standards oriented to medical imaging.

2.1.1 PACS

A Picture Archiving and Communication System (PACS) is a system that provides digital acquisition, storage, display and communication services of medical data [3].

A PACS can be a simple system with a few components (the modality acquisition equipment, a display station and a small database) or it can be as complex as a hospital-integrated system can be. Either way, it is important that the system be built in order to be suitable with both patient workflow and diagnostic components.

The patient workflow varies from an imagiology department to another but, in general, follows the following steps:

- The patient arrives at the health care center.
- If the patient is a new patient then he is registered.
- The imagiologic exam is ordered.
- The technologists receive the exam requisition and call the patient.
- Technologist performs the imagiologic exam.
- The imagiologic exam data is stored.
- Technician previews the exam and reports.
- The report is stored and made available for clinician viewing.

A PACS is generally composed by the image acquisition equipment, a PACS controller and archive, and display workstations linked by a network.

Some years ago, there was also an image and data acquisition gateway which was the bridge between the imaging modality (image acquisition equipment) and the PACS controller. This component associated the image acquired by the imaging modality with the related data of the patient from the HIS (Hospital Information System) and RIS (Radiology Information System), so that the radiological exam could be sent to the PACS controller. This component played the important role of translation between the language spoken by the imaging modality and the one spoken by the PACS controller.

Nowadays, the greatest part of the image acquisition equipment already speaks the language of the PACS controller [7], so the image and data acquisition gateway, in most cases, became obsolete. The image acquisition equipment uses the data received from the RIS through the modality worklist, docking that data with the image and sending it to the PACS controller.

The PACS controller is the management agent of the system it receives data from the image acquisition equipment, the HIS and RIS. Its most important parts are the database server and an archive system (that in its turn can be divided into: short-term, long-term and permanent storage). The PACS controller is responsible for several functions, such as: receive images from imaging modality (or the acquisition gateway), update a database management system, check the data integrity, compress the image data and provision of the query/retrieve service [3].

The display workstations have a database, display and processing software. These workstations can request images from PACS archive and store it locally. Usually, they have only a limited set of images necessary for a patient examination. Their usability is increased with inclusion of reporting tools and measurement tools to support diagnosis. Moreover, some equipment also provides image reconstruction services.

There are three basic architectures of this kind of systems, they are: stand-alone, client/server and web-based. These architectures will be described in the following sub-items.

2.1.1.1 Stand-Alone Model

The stand-alone model consists of a system where the images sent by the imaging modality to the PACS server are saved in the PACS archive server and then a copy is automatically forward for the reading and reviewing workstations. These workstations are endowed with a cache where some images can be stored for a short period. Besides this procedure, it is also possible for a workstation to get data from the query/retrieve service of the PACS server.

Between the strong points of this architecture it is possible to find: the data redundancy (once there are multiple copies of the exam), the cache system of the workstations makes the network performance more stable, an easy access to some historical exams (they can be stored in the cache of the workstations) and there is some resilience when the system faces a server crash (once it is possible the imaging modalities send the exam directly to the workstations).

On the other hand, it presents some drawbacks, such as: the possibility of several radiologists review the same exam from different workstations (at the same time), besides, each workstation may have a different set of images what can turn to be inconvenient reviewing all exams in all workstations.

2.1.1.2 Client/Server Model

The C/S (Client/Server) model is based on a centralized archive of images, at the PACS server. In this architecture, the exams are sent by the imaging modality to the PACS server that stores them in the PACS archive. This archive is accessible from the client workstations that can choose an exam. The images of the chosen exam are then sent to the client workstation and, once

the images are read they can be flushed from the workstation. Nowadays, with the developments of storage technologies and their decreasingly cost, workstations have increasingly storage capabilities. Therefore, the images can be kept in the workstations for some time, dependently from the storage capacity of the workstation. So, it can be said that these workstations acts like micro-PACS.

This architecture provides access to all PACS exams from any workstation, at any time. Besides, there is no need for prefetching images (i.e. the retrieving of non-requested studies from the server to the local cache of the clients, expecting that the studies will be needed) once clients request exams as needed. In addition, it is possible to avoid easily the multiple readings of the same exam.

Nevertheless, PACS server acts as a single point of failure and if it fails all system fails. The clients' workstations will be unable to get the exams and the imaging modalities will store new exams images locally, until the server backs up. Systems with this architecture are very vulnerable to network performance variations.

2.1.1.3 Web-Based Model

The web-based model is very similar to the client-server model, differing mostly in the client access to the application. Here, the client application is a web-application, accessed by the browser.

The advantages of this architecture are the same of the C/S model with high portability and theoretical platform independent workstations (since they only need to support a web browser). On the other hand, a web-based implementation can bring some limitations in the performance and number of functionalities supported.

2.1.2 DICOM

The PACS concept started to proliferate in the eighties associated to the increasing use of computers in the clinical scenario and the appearance of computed tomography and other digital medical imaging modalities. However, the interoperability was restricted for equipments of the same fabricant. It was critical to make the equipments of different providers interoperable and a normalization effort resulted in international DICOM (Digital Imaging and Communications in Medicine) standard [8]. Between others, DICOM defines the network communication layers, the services commands, the persistent objects coding, the media exchange structure and the documentation that must follow an implementation [9]. This standard was well accepted, being implemented in the greatest part of actual medical imaging equipments. As a consequence, during the last years, much DICOM data has been created and stored in repositories.

The following sub-points describe some items covered by the DICOM standard, ending with a balance of the DICOM standard.

2.1.2.1 Information Object Definition

The DICOM standardize a way to represent Real-World objects and to do so, it uses the Information Object Definition (IOD) concept. An IOD is *"a way to encode physiological information into an abstract definition applicable to communication of digital medical images and related information"*[10]. However, in the Real-World the objects are not isolated islands, existing relationships between objects that, DICOM accomplish. An IOD that includes information about other Real-World Objects is called a Composite information object. Nevertheless, DICOM does not relate arbitrarily the different objects. It normalizes an Entity-Relationship Model which relates the components or Information Entities (IE) of the IOD.

In Figure 2.1, it is presented a simplified diagram of the DICOM composite IOD information model. As it is shown, there is some hierarchy in this model. The Patient is in the top, being the subject of the study (i.e. the definition of the characteristics of a medical study).This study contains one or more series (i.e. an aggregation of IOD composites into distinct logical sets), which contains raw data, structured report (SR) document, waveform, spectroscopy, measurements, presentation state, image and/or surface. Either way, each series must contain one or more: Presentation State IE, SR Document IE or Image IE [8].

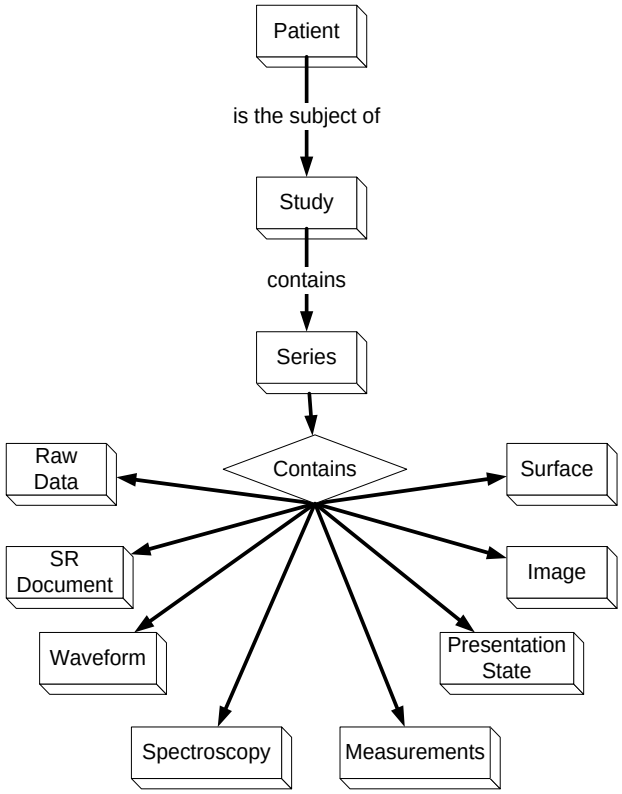

Figure 2.1 - Simplified diagram of the DICOM composite instance IOD information model

A DICOM file is an encapsulation of a persistent DICOM IOD instance. DICOM standard normalizes the format of those persistent object instances. According to the standard, these files must begin with the DICOM file Meta information block followed by the stream representing the data set (Figure 2.2).

The DICOM file Meta information begins with 128 bytes of preamble (available for specific implementations, usually padded with zeros) followed by the string "DICM", the size in bytes and other relevant data. Except the preamble and the string "DICM", all other fields of a DICOM file are represented in a TLV (tag length value) structure, which means that each element of the file is divided in three fields: the tag (the identification of the field), the length (the number of bytes of the value field) and the value. For instance: the tag 00100010 (in hexadecimal notation) represents the "patient name" field and, if the value contains "Carlos Marques" the length field must be 14. The previous example means that this DICOM file is related to the patient whose name is "Carlos Marques".

In Figure 2.2 is presented a diagram of an example of the DICOM files structure, where there are some headers that precede the image data. As previously described, these files have a sequence of elements with data about the patient, the study, the series and the image itself.

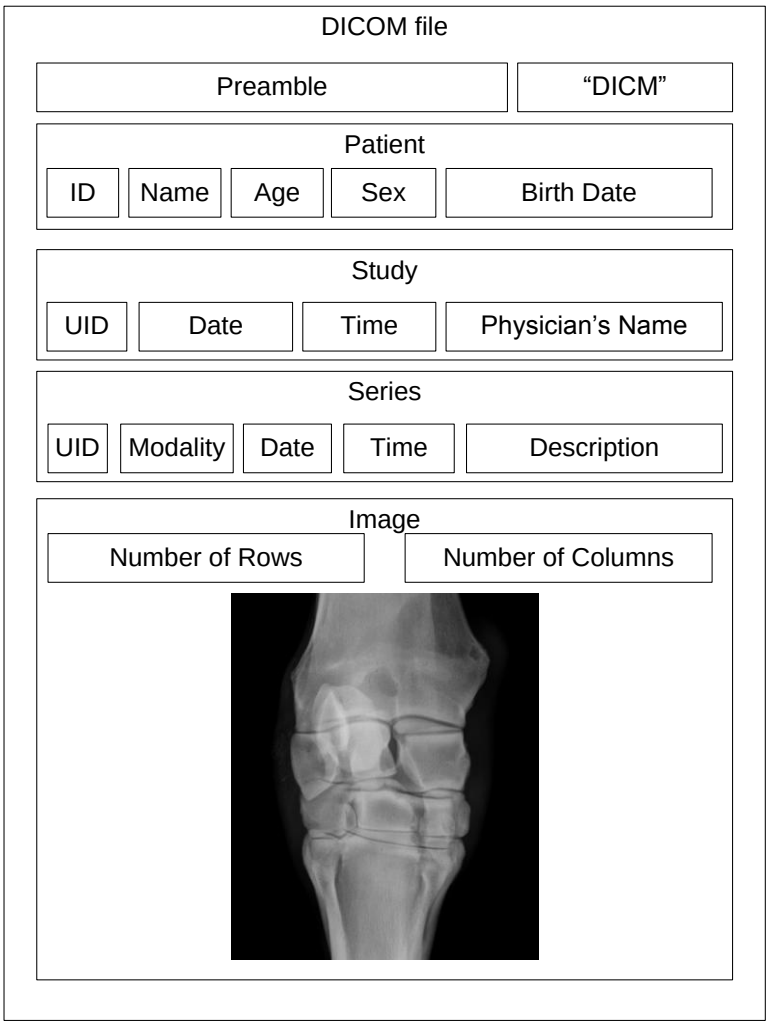

Figure 2.2 - Diagram of an example of the structure of DICOM files

2.1.2.2 WADO

Web access to DICOM persistent objects (WADO) is a standard for distribution of images or other medical data via web [11].

This protocol allows the remote access to DICOM files through HTTP (Hypertext Transfer Protocol) or HTTPS (Hypertext Transfer Protocol Secure) channels. Basically, with an implementation of this standard, the client must send a HTTP Request (GET) with a URL/URI of the object wanted. The response must be a HTTP Response which message body shall be in one of the following formats: DICOM, JPEG, GIF, PNG, JP2, MPEG, plain text and HTML. It is also recommended the server also supports XML, PDF and RTF. Other formats are also possible to be supported.

This standard is very simple to implement and use, giving the possibility of receiving DICOM objects without needing to “speak” DICOM. The client has to specify the DICOM object wanted through its unique identifiers. However this service allows web accessing of a requested object, it does not allow the look up of objects that matches some query.

A PACS web-based architecture must uses a web server to work, what brings disadvantages such as: (1) the server is a bottleneck and a point of failure; (2) it is needed to

maintain the server or to trust in third parties; (3) to improve the capability of the service it is needed to improve the web-server (or other component in the server side). Besides, WADO does not allow queries such as: "all images of the patient: YYY" (where "YYY" is the name of the patient), so for object searching it is needed some secondary service that would make possible the object look up.

2.1.2.3 DICOM Balance

DICOM standard was created to allow interoperability between systems of different manufacturers. It is a flexible standard that can readjust to changes in medical imaging equipment and technologies [12].

DICOM standard normalizes object structure, coding and communication protocol. However, a problem of inter-institutional interoperability remains. This interoperability problem appeared because, in practice, some rules of the standard are ignored and, consequently, the filling of the DICOM file fields can change from hospital to hospital. For instance: (1) the patient ID of one patient can varies from one institution to another; (2) a patient whose name is "Carlos André Ferreira" can be known as "Carlos A. Ferreira" in one hospital and "Carlos Ferreira" in another; (3) some systems do not fill properly all DICOM file fields, for example, letting some fields in blank. The DICOM standard is not prepared for these situations, once it is mainly focused for intra-institutional communications. Another disadvantage of this standard is its weak file searching capability, it only supports the search for some DICOM file fields and it does not support content-based searches.

2.1.3 Medting

Medting is a web 2.0 service of sharing and publication of medical imaging [13]. That allows physicians to share medical images (or videos) in order to achieve medical cooperation for a better diagnosis.

Despite this system allows the medical image sharing, it does not solve the problem of inter and intra-institutional image transfer, because, in order to share the images, they need to be uploaded to the system, and then they will be accessible. This upload can be a really hard task, when we are dealing with massive amounts of images.

2.2 Peer-to-Peer Networks

A Peer-to-Peer (P2P) network is a distributed system, where its participants (peers) share a part of their own resources, for example: files, CPU cycles, storage, printers, etc. The set of these shared resources provides the service that the network was made for [14]. In this kind of systems, each peer acts as a "*servent"*, i.e. each peer has the capability of acting as client and server at the same time, providing and consuming resources.

The birth of this concept is wrongly associated with the appearance, in 1999, of *Napster*. However, it was already being used, for instance, in the Ethernet protocol [15], which treats all machines equally. Another example is an ancestral of internet called *ARPANET* which was built as a P2P network that linked some US universities as equal computing peers [16], where each university shared its own resources. The novelty of Napster was the ascension of the P2P concept in the OSI model to the application layer.

Until Napster, home computers were treated as dumb machines with no resources, what turned to be a bad strategy, since home computers were getting more and more powerful [17].

The Internet infra-structure usage is growing in volume and services, like, for instance: file sharing, instant messaging, VoIP, cloud computing, software publication and distribution, search

engines, backup services, multimedia streaming, storage services, multiplayer online gaming, etc. Many of those services implementation are supported by P2P technology.

The P2P strategy, exploits the computational resources of regular computer spread around the world changing the shape of the Internet. According to a survey performed by a German traffic and management company (ipoque) in 2007 the P2P was responsible for 73,79% of all German's Internet traffic [18]. Thanks to P2P, the home machines became key players in the Internet environment transferring the resource control from the data centers into the periphery of the Internet.

This concept allowed the access to resilient systems where each peer can contribute in the provision of a service. So, with many peers collaboration, it is possible to provide a service in large scale. For instance, in a processing power sharing oriented P2P network, it is possible to build powerful and cheap systems composed by machines with heterogeneous processing powers and obtain positive contributions to solve the problem even from small machines [19-25]. Moreover, if the service is provided by multiple machines it will be less probable that, sometimes, the service is unavailable than if the service was provided by a single machine as in Client-Server (C/S) architecture. Besides, the service provided can grow with the increase of the number of peers, instead of the need to improve the server capabilities in order to improve the service provided. This characteristic is called scalability and it is one of the main characteristics of the P2P networks.

Therefore, the use of P2P technologies in the design of some systems can be a valuable asset, for some reasons, such as:

- Efficient usage of computational resources, i.e. with a P2P network it is possible to use resources (for instance: CPU cycles) shared by other machines that would be wasted without a P2P technology.
- Scalability and resiliency.
- It is given, to common machines, the opportunity to share its resources, in order to collaborate with the provision of a service and, at the same time, the opportunity of using it.

The shift of the responsibility of providing resources from servers to peers also brought some problems:

- Peers are less reliable than servers, i.e. peers can join or leave the network at any time. This fact must be taken into account in the design and development of a P2P network, once it may cause stability problems to the network.
- The question: "How does a new peer know to which address he may contact in order to join the network?" is a common problem in P2P architectures known as the bootstrap problem. This is caused by the dispersion of the resources in the network that causes difficulties in the access to them.
- Some peers may be malicious, in other words, they can intend to damage the network, other peers or the service provided by the network.
- The management and control of many machines may be difficult.
- Peers with fewer computational capabilities shall not be held liable for some responsibilities, so the network performance is not compromised.

In an ideal P2P network, all peers are treated as equals and a random peer elimination will not trigger any loss of network service [14]. However, this scenario is not always possible and, in many cases, is necessary to have a machine responsible for some services. There are different

architectures of P2P networks, with distinguished advantages and disadvantages. In the next sub-items we will describe existent P2P network topologies.

2.2.1 Centralized Architecture

Centralized architectures are characterized by its similarities with the client-server (C/S) architecture, because it uses a central entity, i.e. a server, for some services, for instance: the file sharing centralized P2P networks usually uses servers for the lookup of peers or files [26] (see 2.2.1.1), but the server can also provide load balancing service (see 2.2.1.2). However, unlike C/S architecture where the resources are located in the server, in a centralized P2P network they are the peers who share the resources.

Such a P2P network can take advantage of a centralized entity in many ways, for instance:

- One of the most common problems in a P2P network is the bootstrap problem. The server can be used to solve this problem, because, usually, it has a static address that can be used by a new peer when it joins the network, as the first point of contact.
- As previously described, other common problem of the P2P networks is a certain difficulty to manage and control the network. With a server that problem may be solved because configuring or managing the whole network may be done in the server, i.e. in a single machine.
- Servers are potentially more trustworthy than clients or peers. Therefore, a P2P network that relies on a server to mediate the communication between peers could bring higher levels of trust to the system.

On the other hand, having a single machine to provide some services can bring disadvantages to the system, such as:

- The network is completely dependent on the server. If the server fails the entire network will fail. Moreover, this aspect turns this host a preferred target of Denial of Service (DoS) attacks.
- There are, also, network scalability problems related with server computational limitations. For this reason, the server is the bottleneck of centralized network. Therefore, for load balancing and redundancy reasons this P2P topology is sometimes implemented resorting to more than one server.

The Figure 2.3 presents schematically a typical centralized P2P architecture.

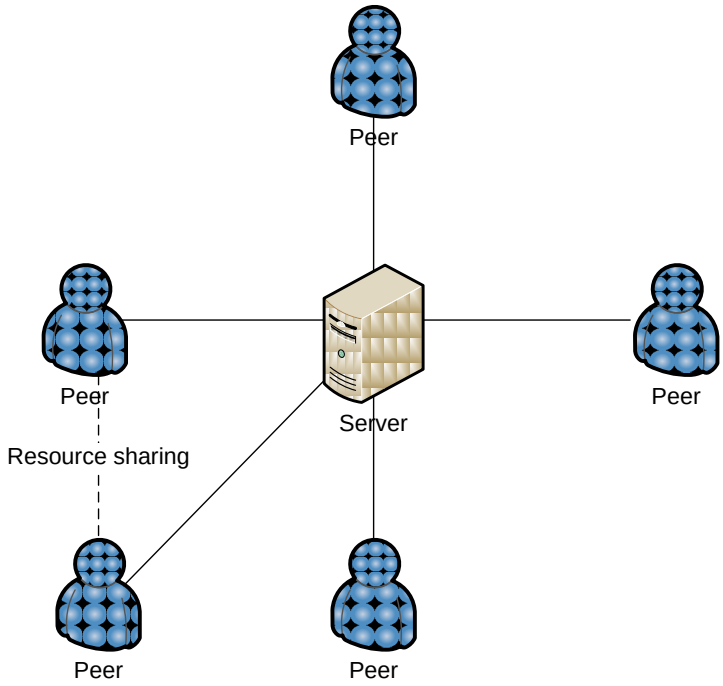

Figure 2.3 - Diagram of a typical centralized P2P architecture

2.2.1.1 Napster

One of the most well-known examples of this kind of networks is the Napster solution. Appeared in 1999, it is a peer-to-peer file sharing network specialized in MP3 music files shared by all connected peers [27]. In fact, Napster was the first P2P file sharing application. It cannot be considerate a "pure" peer-to-peer network, due its dependency of a central server to index all network files. However, it brought the possibility of domestic users share its resources, what was not possible until then.

This network has a centralized topology, since it relies on a server. This server is the entry point of the network. When a peer wants to join the network, it registers itself in the Napster server giving information about the files it shares. The server indexes the files advertized by peers. So, when a search is made, the query is sent to the server. Using the server index of the files, the central machine builds the answer and sends it back to the asking peer.

What makes Napster a P2P network is the file transfer performed between peers. The server query response brings information about files and the sharing peers, turning possible the direct communication between client and the file sharing peer.

Since the server has a full index of network files, provided by every peer, it is granted that the search results contain all resources available. However, it is only possible to search for the fields indexed on the server.

2.2.1.2 SETI@home

SETI stands for *Search for Extraterrestrial Intelligence*. It is a scientific area that aims to find evidences of the existence of intelligent life outside the Earth. One possible approach for this research is the *radio SETI*, which uses the data obtained from radio telescopes and processes it, hoping to find some sort of radio signal produced by any extraterrestrial technology [20]. Taking into account that radio telescopes provide large amounts of data, it was necessary an enormous computational power unavailable in one single machine. To answer this processing power

requirement, the SETI@home network was built, using millions of computers of volunteers that join the project offering CPU cycles [23].

SETI@home is a distributed computer system with centralized topology, since there is a server that sends to each peer a certain amount of data that will be processed by them [28]. The results are sent back to the server when the operation is completed.

This project shifts resource-intensive functions from a central server to workstations and home PCs (the central paradigm of peer-to-peer networks).

After 2 years and half of life, this project reached one Zeta-FLOPS (10^{21} FLoating point Operations Per Second), a measure of computer's performance. Today that number is growing, reaching several dozens of Zeta-FLOPS [24].

2.2.2 Decentralized Unstructured Architecture

The decentralized unstructured architecture, contrarily to the centralized topology, does not rely on any machine with special responsibilities, treating all peers as equals. This equality of treatment between peers makes peers to connect arbitrarily with each other as it is presented in Figure 2.4.

Typically, the resource discovery in this kind of networks is made through message flooding, i.e. one peer sends to all other peers he knows. Those peers dispatch those messages to all peers they know. The process continues until the TTL (time to live) limit of the message is reached, in other words, until the message has been forward a determined number of times. If there are resources that match the request, the messages that describe them are back propagated to the first sender of the request message.

The major attractions of this approach are at the expense level and at the resilience level. Due to the fact that it obliterates the central server from the network topology, it is possible to build affordable systems by reducing the infrastructure and its maintenance costs. The resilience of this type of networks is typically high, because it follows a "herd" approach: loosing several elements of one herd does not influence the herd's well being as a whole. In other words, the disconnection of random peers would not influence majorly the services provided by the network. Moreover, it also turns this strategy more robust to DoS attacks.

As it was mentioned before, bottlenecks limit the scalability of the network. Although, this strategy does not have, theoretically, bottlenecks (the work load is balanced among multiple peers) it scales poorly. Because, the search mechanism relies on message flooding that also brings scalability problems [29]. There is a gradual, yet fast, increase of the number of messages exchanged between peers, because the same message can be forward to the one peer more than once. Furthermore, due to the lack of structure of this network architectures and the message TTL limit, it is not guaranteed that every peer on the network will receive the search query and, as a consequence, existent resources can be not found by the search mechanism. However, since queries are sent and the receiving peer searches locally for resources, it is possible to perform complex search queries [30], such as regular expressions or content-based searches.

Nevertheless, a network where every peer is seen as equal could bring trust challenges, since there is no certification authority (that could be provided by the server in the centralized topology). Therefore, a malicious peer that follows the communication protocol could perform, for instance, spoofing, data modification or men-in-the-middle attacks. As a result, the trustworthiness of this type of networks is low. Moreover, the bootstrap problem is a severe issue of these networks, because when a peer is joining the network he does not know the address of anyone.

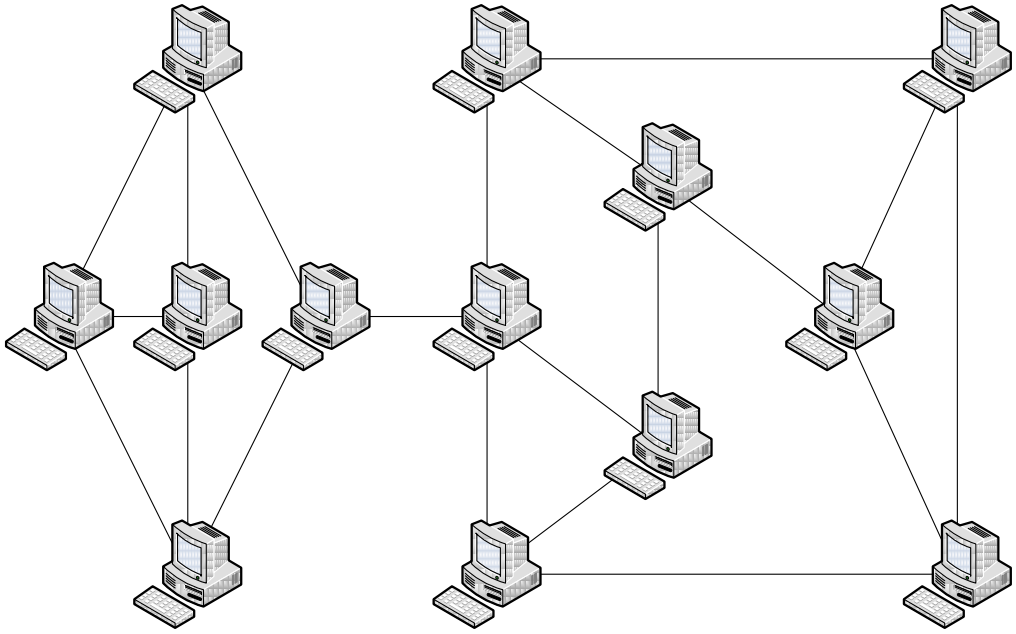

Figure 2.4 - Diagram of an example of a P2P network with decentralized unstructured architecture

2.2.2.1 Gnutella v0.4

Gnutella is a decentralized searching protocol whose participants form a virtual network communication in a pure decentralized P2P fashion [31]. The last fully decentralized Gnutella communication protocol specification [32] was very simple. It only contained five message types, more precisely: *Ping*, *Pong*, *Query*, *QueryHit* and *Push*. The messages are forward by all peers using a constrained broadcast mechanism: upon the receipt of a message the peer decrements the message TTL (typically starting at 7) field. If the TTL is greater than zero it will send the message to all peers it is connected to. Moreover, when a peer receives a *Query* message it checks its local file store and, if the query matches, responds with a *QueryHit* message. The *QueryHit* responses are routed back, following the same path of the *Query* message that triggered the search process. The eventual file transfer is done through a direct connection between peers using HTTP traffic. Gnutella protocol is oriented to the lookup of resources. In fact, the Gnutella network is not used for file transfer. Nevertheless, this is available a push mechanism to permit the file transfer when the source file peer is behind a firewall.

To solve the bootstrap problem the protocol does not specify any solution. Nevertheless, some mechanisms were implemented. For instance, host caches where the addresses TCP/IP of active peers are obtained via DNS or via a web-site [33]. After the discovery of the first peer, there are mechanisms to maintain the list of known peers updated.

2.2.3 Hybrid Architecture

The previous peer-to-peer architectures are not perfect. The centralized topology depends on a server and has all the disadvantages of that dependence: bottleneck, point of failure, scalability, server maintenance costs, etc. However, a fully decentralized topology does not guarantees that all resources are found, it has also scalability problems related to the number of messages exchanged, and so on. Therefore, in order to achieve a combination between those two topologies, the hybrid architecture appeared. The hybrid architecture's main characteristic is the splitting of peers into hierarchical layers. So, this topology gathers characteristics from:

- Centralized topology: There are machines more important than others, which are responsible for determined services.
- Decentralized topology: There is no servers, no single-point of failure neither bottleneck. Besides, the network is self-organized [26].

Most commonly, as shown in Figure 2.5, in this kind of network peers are divided into two classes: ultrapeers and leaf-peers. Ultrapeers are elected peers that have some determined responsibilities and are connected with each other as in a fully decentralized architecture. The leaf-peers are connected to some ultrapeers and treat them in a similar way as the *Napster* clients treated the *Napster* server.

Therefore, it was possible to build a network with complete search results, with no need of a dedicated server. Besides, it is also possible to support complex queries, like regular expressions.

In the other hand, the network is less trustworthy than a centralized one, because peers may not be always connected, joining and leaving the network at any time. This peers' state instability brings problems when a peer is an ultrapeer, demanding a network readjustment. Treating some peers as more important than others brings problems at the resiliency level, because if some ultrapeers crashes for some reason, the services provided by the network might be compromised. Moreover, the elected ultrapeers may be malicious, which leads to the importance of carefully choose the election process of ultrapeers [29]. Besides, this kind of networks inherits the bootstrap problem from the fully decentralized topology.

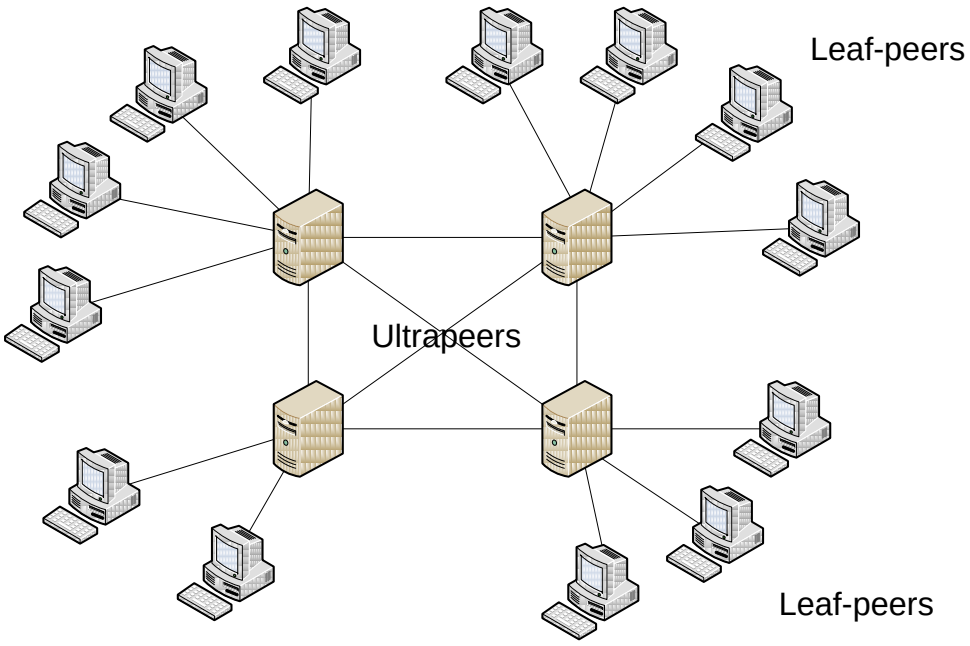

Figure 2.5 - Diagram of a typical hybrid P2P architecture

2.2.3.1 Gnutella v0.6

Since Gnutella v0.4, the Gnutella communication protocol had been suffering improvements, primarily to solve scalability problems. The most significant changes applied by Gnutella v0.6 were: (1) – The introduction of ultrapeers and Query Routing Protocol (QRP). (2) – Pong cashing. (3) – support rich search queries.

Gnutella v0.4 had two features that limited its scaling capabilities: flooding tended to unduly swamp the network and TTL values in the messages (introduced to alleviate the flooding

effect) reduced the number of peers reached by the search mechanism. To minimize these problems, Gnutella v0.6 introduces a new topology that uses ultrapeers and leaf-peers, creating a hierarchical structure. Peers with higher bandwidth may be elected to ultrapeers, keeping many connections to the Gnutella network simultaneously and, at the same time, routing more traffic. The peers with limited resources can join the network as leaf-peers, directly connected to a limited number of ultrapeers [26]. The ultrapeers are also proxies for leaf-peers, forwarding queries to leaf nodes if the leaf-peer can answer. The availability status of the leaf-peers is supported by the QRP, which specifies that each leaf-peer should upload a vector to the ultrapeers directly connected, containing the file names that the leaf-peer is sharing. The ultrapeer filters incoming queries, so that leaf-peers only receive queries when their vector contains a matching file name.

Furthermore, the Ping and Pong traffic consumed a significant portion of the available bandwidth. To reduce the Ping messages (and as a consequence the Pong messages as well) it was introduced caching Pong techniques. The peers cache several Pong messages received recently. When the peer receives a Ping message, instead of forwarding the Ping, it answers with the cached Pongs not expired. This way, the Ping-Pong was significantly reduced.

2.2.3.2 Skype

Released in 2003, Skype is a peer-to-peer network with hybrid architecture for voice over IP client, being, nowadays, the largest provider of international calling [34].

The network architecture includes a server (Skype login server) that is used to solve the bootstrap and to authenticate users. Each peer is a potential ultrapeer, since it has enough resources to fulfill its responsibilities. The buddy list and a list of the ultrapeers are stored in the machine where the peer is hosted. However, the remaining data is stored in a decentralized way. Skype assures complete searches for users that has logged up to 72 hours before [35].

2.2.4 Decentralized Structured Architecture

Decentralized structured architecture is a peer-to-peer topology where each shared resource is stored on the node which address is somehow related to it [29]. A very used data structure to support this network is a Distributed Hash Table (DHT). Each network item is stored as a pair (key, value), where value is the resource (or a pointer to the resource) and key is a hash code of the resource (i.e. a code returned by a function that converts some data, usually with variable size, into a small code).

The Table 2.1 presents an example of a hash table, where the hash function returns the initial letter of the value.

Table 2.1 - Example of pairs (key, value) using a hash function that returns the initial letter

Key	Value
A	Albert
C	Colbert
M	Mike
R	Ronnie
T	Tricia

One example of the usage of hash tables on the everyday life is the phone books that usually have the contacts organized by initial letters. These initial letters are like the keys of the

hash table, the name of the person and its respective phone number the value. The keys are the elements that allow quicker lookups.

The difference between a DHT and a usual HT is the table decentralized storage mechanism, where it is assigned an ID (that is usually in the same key space) for each peer. This ID gives responsibility over a specific range of item keys. Besides, every peer has also information about other nodes, especially the closest ones. So, a query can be progressively forward to the peer responsible for items that will matches it [36].

In Figure 2.6, it is presented three examples of implementations of P2P decentralized structure:

(a) – Pastry arranges the peers and the resources into a logical ring according to their hashing keys. One may consider the logical ring as one clock where the peers and the resources are ordered clockwise. The lowest keys are immediately after the zero hour and the highest keys are immediately before the zero hour. Each resource ID is assigned on the first peer whose ID is equal or follows the resource ID. Therefore, the algorithm assigns each resource by travelling clockwise on the ring until the closer peer is found, that peer will be responsible to hold the resource.

(b) – CAN (Content Addressable Network) it differs from Pastry as it uses a multidimensional key space as underlying abstraction for routing. Keys are mapped into regions and peers split regions while memorizing peers responsible for neighbor regions. Routing is then performed by forwarding requests to the regions closest to the position of the key. In the figure's example the key has two dimensions. Therefore, the region has a planar shape.

(c) – Kademlia protocol disposes the peers into a logical binary tree, for measuring the inter-peer distance (XOR metric) and the assignment of resources to each peer. Scaling by sub-trees, the peers in the same sub-tree are closer to each other and peers in wider sub-trees are further away. (This protocol is more minutely described in 2.2.4.1).

The decentralized structured topology is the most scalable architecture presented in this dissertation while comparing with the other P2P architectures. It allows complete and efficient lookups without the need of a central server or query flooding (two methods with limited scaling capabilities).

Nevertheless, this efficient search is only possible because it relies on an organized structure. Therefore, when a peer leaves, joins or crashes, the structure needs to readjust itself, i.e. it trades-off an efficient and complete lookup by the cost of maintaining an organized logical structure. Besides, it is assumed that the peers are trustworthy, which may be not necessarily true since there can be malicious peers. Furthermore, the structure is typically based on some specific fields (e.g. file name) therefore it only allows lookups over those specific fields and does not enable rich search mechanisms. Finally, bottlenecks can also occur on peers responsible for popular keys, being necessary to take this into account, during the development process of this kind of solution.

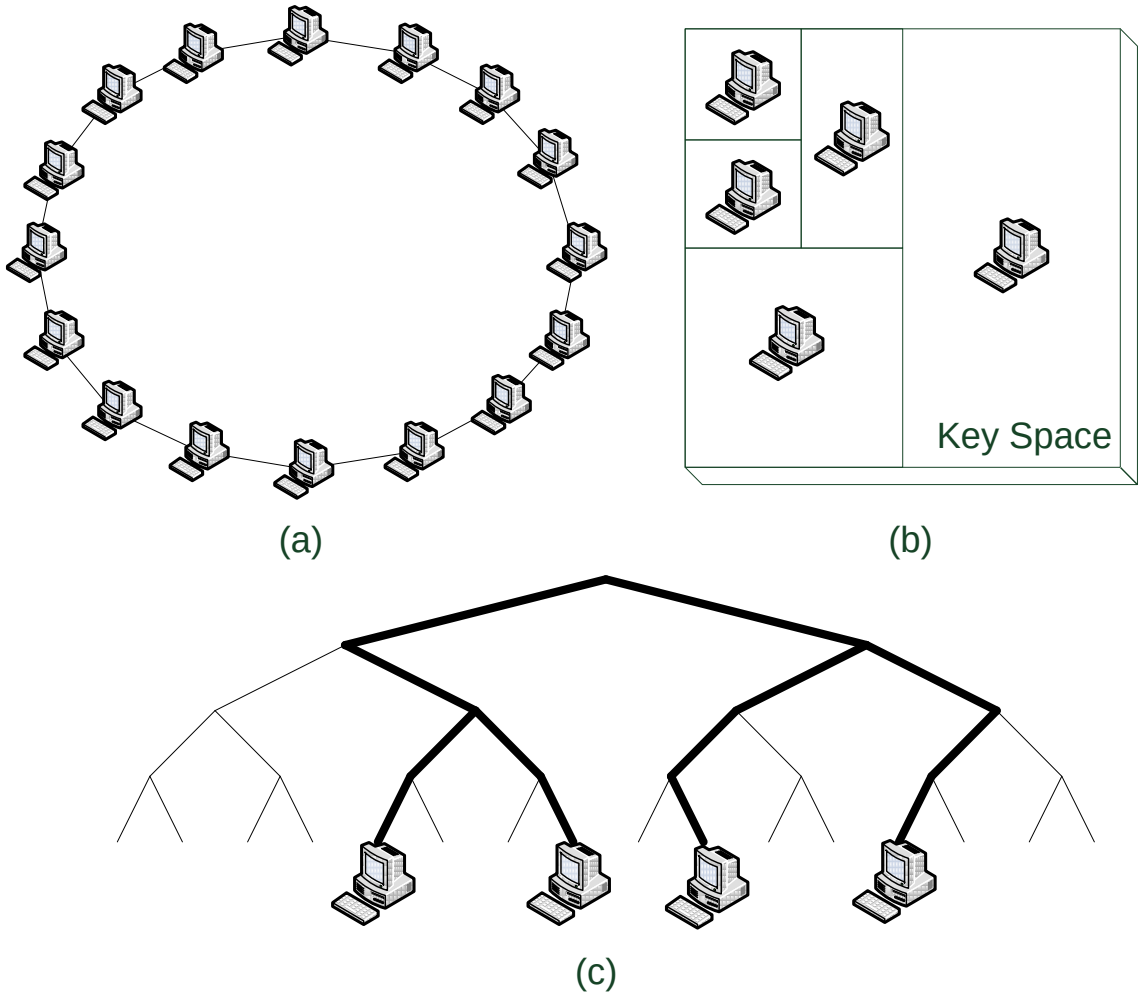

Figure 2.6 - Diagrams of decentralized structured P2P architecture examples: (a) Pastry, (b) CAN, (c) Kademlia

2.2.4.1 Kad Network

Kad network is a Kademlia-based DHT network and it is used by popular P2P applications, such as eMule, aMule and BitTorrent. Due to its enormous usage and complexity, we will describe more in detail this kind of network.

As other DHT-based protocols, Kademlia assigns a random ID to each peer [37], which makes the peer responsible for a certain range of keys. The peculiarity of this protocol is the assignment of the key range that is done using the XOR operation.

The visual way to represent the XOR operation in the distance measurements is through a binary tree. For a better understanding of the placement of the peers in the binary tree, it is presented the Figure 2.7 where it is shown how the Peer S, which ID is 011, is placed in the tree. It is important to highlight that this is not a part of the process of the Kademlia. It is only an explanation of how the peers are distributed in the binary tree. This figure is divided in four parts:

(a) – Initial state of binary tree, before introduce the Peer S with key 011.

(b) – The most significant bit (MSB) of the key is 0, therefore the right sub-tree is chosen.

(c) – The second MSB is 1, consequently the left sub-tree of the sub-tree chosen in step b is selected.

(d) – The less significant bit (LSB) is 1, so the Peer S is placed in the left leaf of the sub-tree selected in step c.

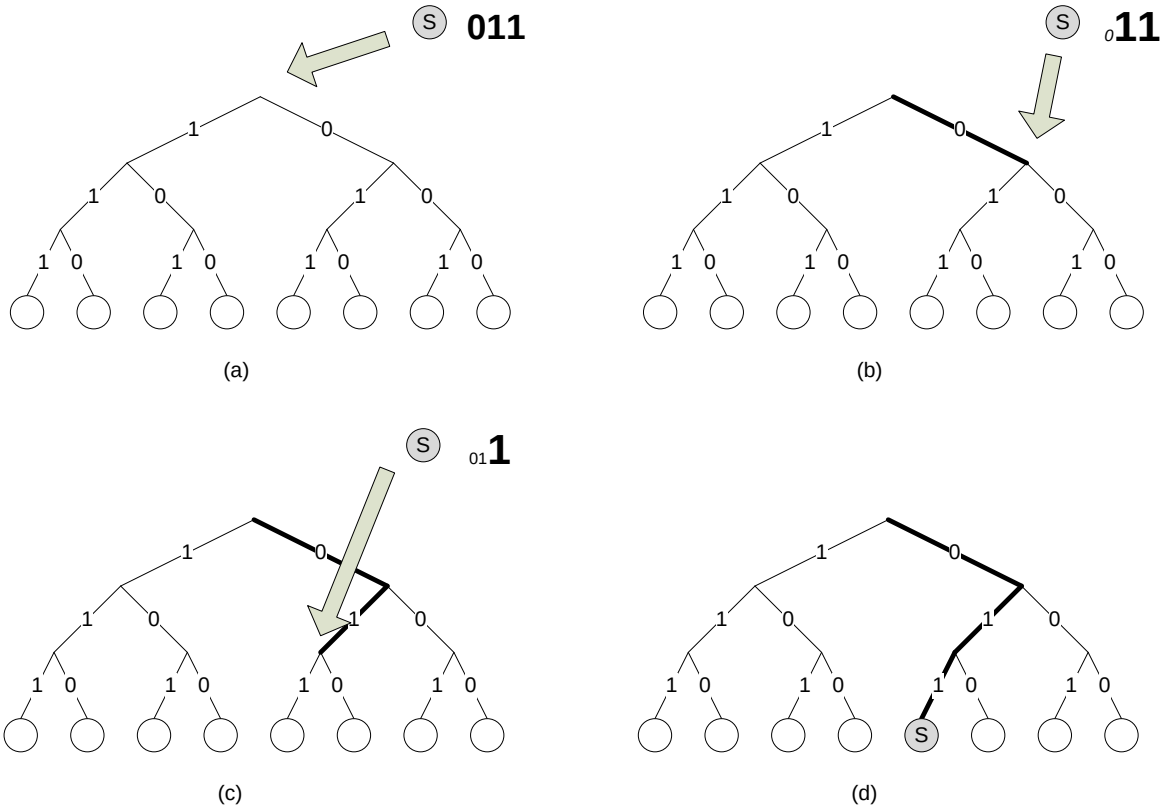

Figure 2.7 – Diagram of the peer placement steps in a binary tree.

Placing peers through a binary tree, where the division is made since the MSB until the less significant bit (LSB) of the ID (one bit per level of the tree), it is visually perceptible the notion of distance of this protocol. In the binary tree, the distance between two nodes depends on the path required to go from one peer to another. The further it needs to climb the tree to reach a node, bigger the distance between the two nodes will be.

Until now, it was only described how peers can be visually represented in a binary tree, for better understanding of the examples shown. From now on, the real processes of Kademlia protocol will be described.

As shown in Figure 2.8 where there are 5 peers: Peer 1 (1011), Peer 2 (1000), Peer 3 (0110), Peer 4 (0011) and Peer S (0111). As previously referenced, the distance calculus is made through a XOR operation, for instance: the distance between Peer S and Peer 1 is equal to: 0111 XOR 1011 that is the same as 1100 in binary notation (or 12 in decimal notation). As we can see, peers in the same sub-tree have a minimal distance, for instance, Peer S (0111) and Peer 3 (0110). On the other hand, peers with different MSB have a big distance, for instance, Peer S (0111) and Peer 1 (1011).

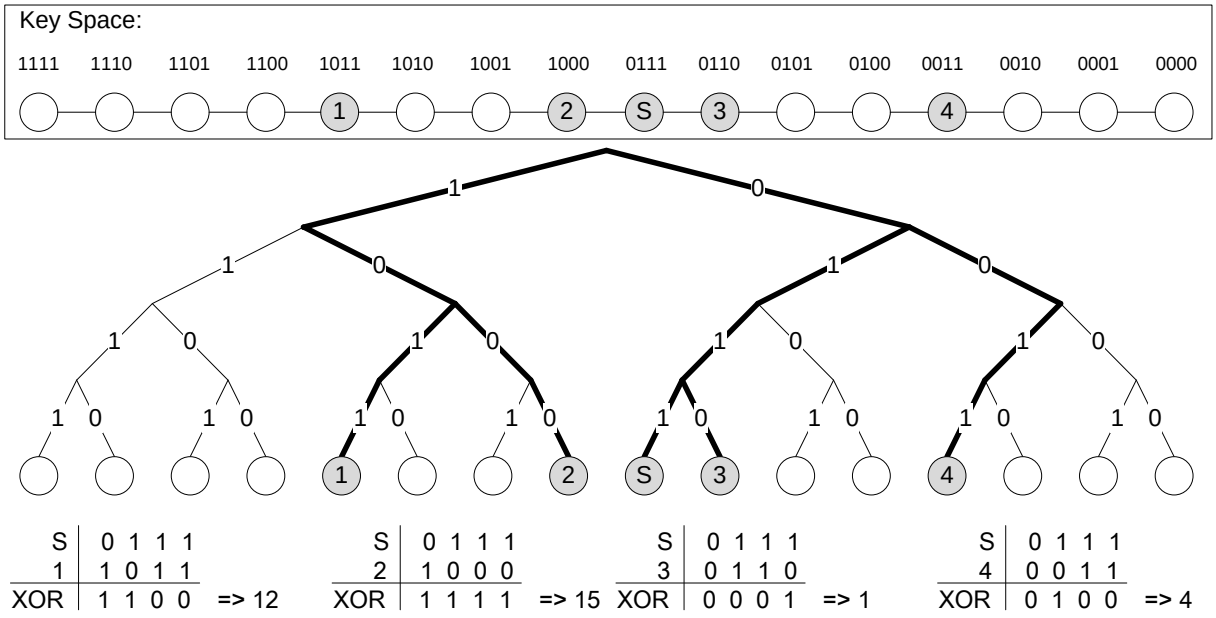

Figure 2.8 - Binary tree representation of a routing table example based on XOR-metric

In Kademlia, each peer has a partial view of the network, also known as routing table. This partial view is formed by a set of K-buckets, which one of those contains information about up to K nodes. For instance, a 2-bucket will have at maximum information about 2 nodes, containing the ID of the nodes and their respective TCP/IP addresses. Each one of these K-buckets is responsible for a specific range of IDs. However, they are not equally distributed along the key space. There are more K-buckets to list the nodes closer than the nodes further.

In Figure 2.9 it is shown an example of the K-buckets distribution in the key space. In this particular case, it was used 2-buckets of the routing table of Peer S, which key is 0111. As it is presented, there is one 2-bucket responsible for all items with the MSB equals to "1", so it is responsible for 8 IDs. On the other hand, there is a 2-bucket that is only responsible for the items with the three MSBs equals to "010", so it is responsible for only two keys. The Table 2.2 presents the relationship between the size of the key range and the distance of the peers, taking into account the example shown in Figure 2.9. The table helps to understand, once more, that the most distant peers have less probability to be listed in the k-bucket than the closest ones.

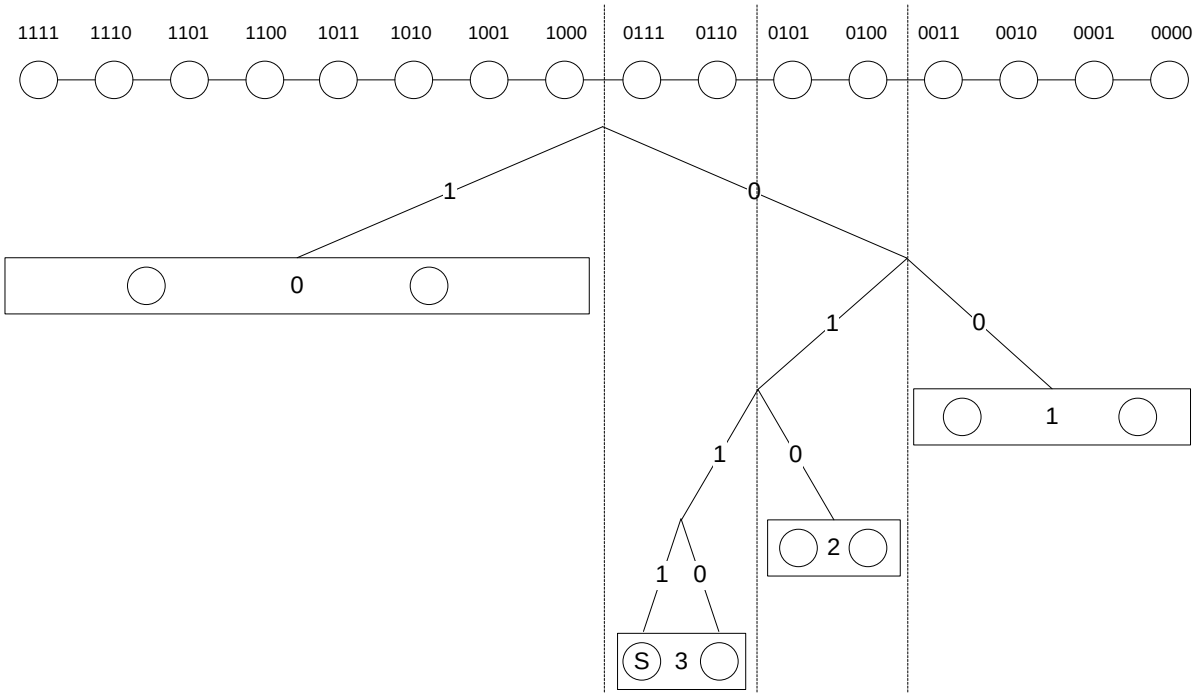

Figure 2.9 - Diagram of the 2-bucket distribution, in the perspective of the peer S (0111) in a Kademlia network with 4-bit key space

Table 2.2 - Table of the attribution of 2-bucket in a 4-bit key space in Kademlia routing table

2-bucket number	Distance of peers	Key Range Size	Number of peers not listed in the 2-bucket
0	Greater or equal to 8	8	6
1	Greater or equal to 4 and less than 8	4	2
2	Greater or equal to 2 and less than 4	2	0
3	Less than 2	2	0

Thus far, the routing table of each peer was explained. However, the following questions still remain: How is the routing table populated? And how does it solve the lookup of resources?

The Kad network developers solved the bootstrap problem with a cache system, where the nodes are stored from session to session and additionally lists with dozens of nodes can be downloaded from web-sites. Anyway, when a new peer makes the first contact with an active peer, the peer contacted sends data about the peers it contains in its routing table. Therefore, the new peer stores that data in its routing table. From time to time, each peer in the routing table is contacted to verify if they are still active. Every time a peer receives a message from a peer that does not belong to the routing table, the peer checks if the K-bucket responsible for the range that contains the ID of the new peer is not full and if it is not it stores the new peer in the K-bucket.

For the lookup of resources, the peer sends the request to the peer in its routing table that is closest (the one who is in the same (or the closest one) K-bucket of the item searched). In its turn, it forwards the request to the peer in its routing table that is closest, and so on until the item is found. This is possible because the items are stored in the peer with the closest ID.

2.2.5 P2P Architectures Balance

There is a diversity of P2P architectures each one of them has its own advantages and drawbacks. It is this diversity that makes peer-to-peer technology so much adaptable and adequate for so many different use scenarios.

The purpose of Table 2.3 is to present a notion of the potential characteristics of each P2P topology. It is important to highlight that it is not a deterministic evaluation of the architectures, but a subjective one, that can varies on the implementation. This table evaluates the different architectures in the following aspects:

- Search scope: associated with the capacity to retrieve all resources of the network that matches the query, from partial list of results until complete;
- Query type: associated to the query filters granularity, from limited searches to specific fields (poor queries) until content-based searches (rich queries);
- Scalability: the capability of the architecture to grow accordingly to the needs;
- Fault tolerance: the ability of the network to stay active if one peer fails, from low (bad) until high (good) tolerance;
- Bootstrap: the difficulty to connect the network, from easy (good) until difficult (bad) to connect;
- Trustworthiness: this factor is medical data oriented. In general, those environments share confidential data, so it is important to insure that the peer can trust the network.

Table 2.3 - Balance table between the different P2P architectures

Architecture	Search Scope	Query Type	Scalability	Fault Tolerance	Bootstrap	Trustworthiness
Centralized	Complete	Poor	Medium	Low	Easy	High
Hybrid	Complete	Rich	High	Medium-High	Medium	Medium
Decentralized Unstructured	Partial	Rich	Medium	High	Difficult	Low
Decentralized Structured	Complete	Poor	High	Medium	Difficult	Low

In conclusion, some implementations have high complexity of messages interchanged and others are difficult to implement and manage. Ones give support to complex queries and others guarantees that the search results contain all resources of the network. The main conclusion is that there is no general best solution, but there are better solutions for particular issue or scenario.

2.3 Peer-to-Peer Technologies in the Medical Scenario

Many healthcare centers have only some modalities available. Therefore, sometimes it is necessary that the patient visit other medical institutions in order to be undergone to the required exams. Not to mention the natural mobility of patients from one institution to another for diverse situations and reasons. These reasons cause the dispersion of data and the consequent problems of access it when and where it is needed. Many solutions have been used: mail, email, patient or private solutions over virtual private networks (VPN) [5]. However, all these solutions bring drawbacks that can make the data exchange impracticable. For instance: (1) VPN can be hard to set up and manage, for bureaucratic and technical reasons; (2) Traditional mail can take too long; (3) the patient can forget, lose or damage the exams; (4) through email either it is necessary to

previously know where the data will be needed, or it is necessary to wait until the responsible for the data answer the request.

To facilitate and promote inter-institutional share of data, we propose the Peer-to-Peer (P2P) architecture and technology to DICOM-based medical imaging repositories. It is not a brand new concept, as the following sub-item shows, which presents a project that uses a P2P network for share of medical images, in order to provide a platform for medical cooperation. The purpose of this thesis is to provide a flexible platform that can be used for diagnostic, medical collaboration and academic purposes.

2.3.1 P2P Platform for Sharing Radiological Images and Diagnoses

The P2P platform for sharing radiological images and diagnoses [38] is a study that intends to be a sharing platform for medical cooperation in order to help physicians in diagnosis, giving to them a set of examples.

The main idea is: sharing exams and reports to the virtual community of physicians, physicians would recognize more easily a rare pathology if they have access to exams of other patients with the same pathology.

This project is based on P2P technology and uses a server for authentication purposes, after the authentication, all peers are able to communicate directly to all the others through JXTA. With this kind of architecture, each peer is responsible for its files, allowing then context-based searches. However, for patient's privacy measures, the images are not tagged with data about the patient.

In conclusion, this project allows the cooperation among the medical community in order to help physicians to diagnose pathologies, giving to them examples previously diagnosed. On the other hand, this project is not oriented for the transport of exams of one patient from one health institution to another.

3 Requirement Analysis

This chapter presents an analysis of the requirements and workflow of the P2P medical network.

3.1 Functional Requirements

This thesis' work was inserted in Dicoogle which is an Open Source and platform independent project. The Dicoogle project is claimed to be a possible alternative to the traditional database of the PACS systems [7]. This project uses index technologies in order to index the DIM field and, consequently, to allow content based searches. These searches retrieve not only the DIM fields, but they can also retrieve other associated data, for instance, a thumbnail. However, Dicoogle was only able to run under stand-alone environment. One of the aims of this thesis was to endow Dicoogle with the query and retrieve capabilities under a distributed system environment.

The following sub-items describe the main functional requirements of a P2P network for medical imaging.

3.1.1 Search

One of the most important requirements of this kind of networks is the capability of searching for resources in the network. These searches must be complete, i.e. if there is one resource in the network that matches the query, it must be listed in the search results. If physicians do not have access to all patient history, they may not be able to provide a proper medical care [39]. Moreover, in the context of medical imaging sharing, if the network did not offer a complete search feature, then an exam possibly would not be found when it is needed and the physician would be forced to request a new one. This would bring several disadvantages, such as: (1) the patient would be exposed to more radiation; (2) the money that would be avoidably spent; (3) the time spent in the acquisition of the new image is the time that the acquisition equipment is not available for other patients, etc.

Another important issue is the non-uniform Patient ID along distinct players, i.e., a patient can be known distinctly from one institution to another. For instance: a patient whose name is "Carlos André Ferreira" can be known as "Carlos A. Ferreira" in the hospital A and as "Carlos Ferreira" in the hospital B. This means that the search mechanism must be able to retrieve the exams entries of the patient "Carlos A. Ferreira" from the hospital A, and the exams of the patient "Carlos Ferreira" from the hospital B.

3.1.2 File Transfer

The file transfer is another very important feature that must be taken into account. The network must have mechanisms that enable users to share images between them. Otherwise, the network would be almost useless to medical imaging visualization utilizations.

After a search has been made and the results have been presented to the user, he must be able to choose a result entry and request the study, series or images transfer.

A medical image can contain large amounts of data that must be transferred as quickly as possible. This aspect is especially important, in urgent cases where the physicians need to have access to the image in time. Here, the solution efficiency and performance must be an important issue in the network architecture planning.

3.2 Non Functional Requirements

The objective of this thesis is to create communications platform to establish bridges between peers under the medical scenario. This environment is a particular one, once it manages very sensible information where confidentiality and integrity of data must be dealt carefully. For this reason, hospital networks are typically firewall protected which only allows communications through a restricted number of TCP/UDP ports. Besides, peers in a hospital are usually behind routers, which is an obstacle for communications from the outside to the inside of the network, once the NAT protocol does not support TCP sessions initiated from the outside. One of the non-functional requirements is: to facilitate and promote the communications between institutions, even under adverse conditions such as the previously described.

The network must be as stable and reliable as possible. On the one hand, the network shall be stable, because, at any time, the medical images may be needed, if the communications are not stable, then physicians probably will not trust in the network. On the other hand, communications between peers must be reliable, so the integrity of the data exchanged does not be compromised.

A P2P medical network commonly varies from a typical file sharing network in two major aspects:

- A medical imaging network has, in general, few peers instead of the millions of peers that a file sharing network can have.
- The peers of a typical file sharing network usually share a small number of files, in opposition to a medical imaging network peer that can have several millions of files.

These differences are visually described in Figure 3.1.

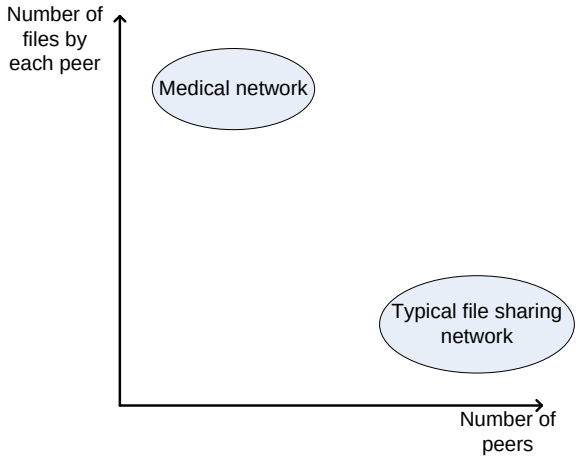

Figure 3.1 - Graph of the location of the medical networks and typical file sharing networks in the space defined by number of files by each peer and the number of peers

3.3 Workflow

In order to develop a system that could respond the requirements needed. As presented in Figure 3.2, the application developed should allow each peer:

- Join the network - this is the first requirement for any other network operation. This process warns the other peers for the existence of the peer.
- Search in the network - Once the peer is joined, it should be possible that he searches the network and consequently receive from the other peers the results.
- File transfer – If a user is interested on receiving one or more files described in the search result list retrieved in the last search, it must be possible for him to request their download. To do that he must select the studies, series or images from the search results and proceeds with their download.
- Leave the network - At the end, if user does not want to download, search or share files anymore, he shall be allowed to leave the network.

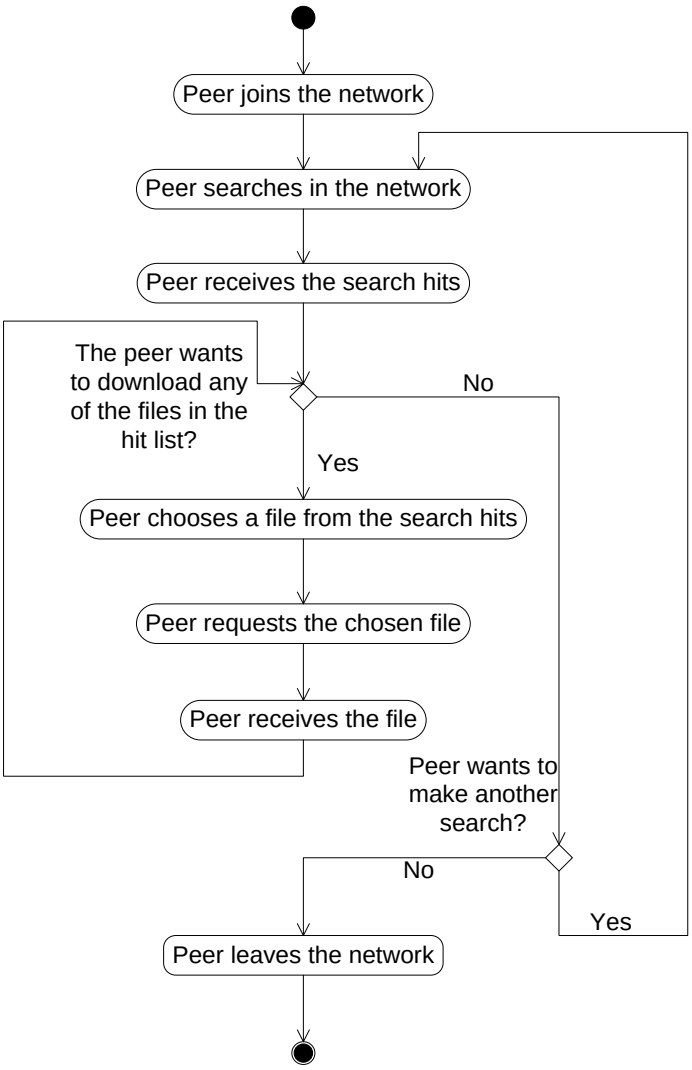

Figure 3.2 - Activity diagram of the application workflow

4 Architecture of the Peer-to-Peer Network for Medical Imaging

This chapter describes the P2P network implemented for medical imaging: its architecture and messages protocol. We will start describing the peer's internal organization and network interaction. Next, it is presented and discussed the network P2P proposal, divided in two distinct operation blocks: LAN and WAN. The LAN process is based on JGroups platform and the WAN process is done through an Internet service.

4.1 Peer

A peer is the elementary component of the network. They are the entities that share and consume resources, i.e. the medical images files in DICOM format.

As described in 2.1.2, the DICOM standard allows the interoperability between systems of different manufacturers. However, this standard does not allow a rich content-based file search that makes impossible some searches as, for instance, "all images of patients with the pathology YYY". Moreover, many times the interoperability between systems of different institutions, in practice, is not very well adapted because of differences in the filling of the DICOM file fields from one health care center to another. In other words, inter-institutional medical environment is ruled by chaos. This chaos is also present in Internet, where data is distributed in a lot of places, under many format syntaxes. The problem comes when it is time to access the information[40]. For that problem, a solution appeared: search engines, such as Google. Those search engines are important tools for look up of information under the chaos of Internet and it uses an indexing strategy. It indexes all data it can find, and when a search is made, the hits are returned by the index.

Therefore, like Google, that uses an index strategy to solve the chaos of Internet, the P2P network implemented uses the same strategy to solve the chaos of the inter-institutional medical environment. However, with an index strategy there were still two possible paths:

1. Global index - the network is responsible to maintain a single index. The index can be: centralized (like Napster, described in 2.2.1.1) or decentralized (like Kad Network, illustrated in 2.2.4.1);
2. Local indexes - each peer is responsible to index its own files. The network queries are performed over a set of distributed indexes, which are logically viewed as a single federated unit. This strategy is used, for instance, in Gnutella (referred in 2.2.2.1 and 2.2.3.1).

When analyzing these two options, the second one fits the best with the medical imaging environment requirements. Due to peer's repository nature which has huge collections of files, it is impracticable to send and synchronize information about all shared files in a single index. Joining the network would bring large amount of data to be sent to the network. So, a local indexing strategy was chosen, where each peer indexes its own files. When a query is made, the messages shall be received by each peer that, in its turn, sends data about its shared files that matches the query.

The adoption of local indexes has advantages: (1) reduces drastically the messages interchanged when a peer joins the network; (2) makes cosmos from chaos, because it allows to search for a specific field and also for DICOM metadata content, even in the adverse conditions of the medical environment.

With local index option, the architecture of the peer is divided in three parts (see Figure 4.1):

1. The index – its responsibility is to index all file repository and retrieving data about the files that matches a query. This layer is implemented with Apache Lucene [41]. This tool is a text search engine API that allows the creation of an index to support content-based searches.
2. The P2P layer – it is the layer responsible for communications. This layer interacts with the index layer (to respond to external search requests) and with the application layer to send local query requests to the network. The eventual network responses from the other peers will be delivered to application layer.
3. The application layer - is the responsible for the interaction with the user and platform management. It uses the index layer in order to retrieve the local results of a query, and the P2P layer in order to query and retrieve data from network.

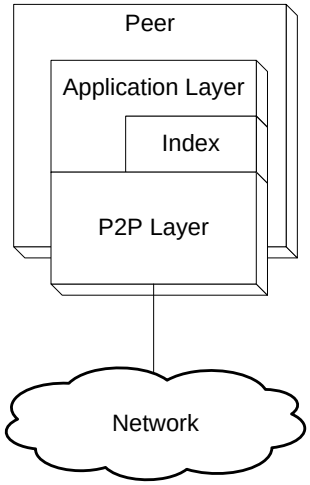

Figure 4.1 - Diagram of the architecture of each peer

4.2 LAN Architecture

LAN (Local Area Network) refers to the internal network of a hospital. Typically, an internal network of a hospital has a little number of active repository peers. Consequently, P2P decentralized architecture is a good option for this scenario, because of the following reasons:

- As previously described, the strategy chosen was a local index strategy: each peer is responsible for an index of its own files (see 4.1). Due to the impracticability of the maintenance of a global index.
- The costs brought by a server that maintains a network with only some peers should be avoided.
- The P2P decentralized topology is the most likely to be resilient.
- There are few (or none) obstacles for communications between peers in a LAN.

Existing P2P platforms were evaluated, namely JGroups [42], JXTA [43], Xeerkat [44] and XMPP [45], to select the one that best fulfils the requirements of DICOM services, in particular:

- Discovering peers: Peers find each other through a common registering mechanism, typically managed by a central server that keeps a list of all peers and resources. However, other methods are available, such as network broadcasting or discovery algorithms.
- Querying peers for resources (i.e. DICOM services): Once these peers have been discovered, the application can ask them for the content desired by the application.
- Sharing resources: In the same way that a peer can ask other peers for specific content, it can also publish and share content after it has been discovered.

The first version of P2P platform was based on Sun's Microsystems JXTA framework version 2.5 [46]. However, exploratory trials showed that this solution was not stable enough for building a robust P2P network. The final decision was to adopt JGroups [42], an open source toolkit for group communication.

4.2.1 JGroups

JGroups is an open source toolkit developed in Java for reliable group communication. The communications are based on two kinds of entities: *groups* and *members*, where a member (also known as *node*) is a process in some host that is a part of one or more groups. On the other hand, a group (also known as *cluster*) is a set of nodes identified by its name. It is not needed to create explicitly a group, once it is created automatically when a node joins a non-existing group.

Members can join or leave a cluster, send messages to all (or single) members, receive messages from other nodes in the group, be notified when other nodes joins or leaves the cluster, have access to a list of the current members of the group. In order to perform such actions, a process has to create a *channel*, which is an infrastructure that acts similarly to a Berkeley sockets, and connects with the group [47]. Once connected, it is possible to send and receive messages and be notified when other node joins or leaves the group. The view provided by JGroups (list of all the current active members of the group) turns possible to send to a particular node a message, selecting one of the members of the list [48].

One of the most important JGroups' characteristics is that messages are interchanged through reliable multicast. Therefore, it guarantees that each message is always received by its destiny and in the right order.

As shown in Figure 4.2, the architecture of JGroups consists in three parts: (1) the building blocks, (2) the channel and (3) the protocol stack.

The building blocks are the highest abstraction level of the JGroups and it provides sophisticated APIs (Application Programming Interfaces) on top of a channel, saving the application programmer from spending time in the development of some features.

Channel is an API, as previously described, that provides important operations such as: join and leave a group, send a message to all current members of the cluster, send a message to a single node, receive messages and see the list of current nodes of the cluster.

The protocol stack is a set of protocol layers that processes every message sent or received. Each layer can modify, reorder, pass, drop, add headers and fragment messages [47]. The sent messages pass from the higher level protocols until the lower level protocols of the stack, on the other hand the received messages passes through the stack in the opposite direction. The protocol stack properties are defined in a XML file that is passed in the creation of a channel. JGroups already provides some implemented protocols and also gives to the programmer the possibility to implement new ones.

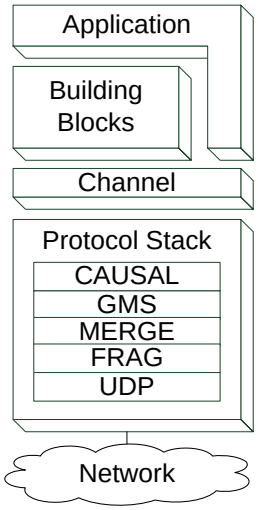

Figure 4.2 - Block diagram of the JGroups' architecture [47]

This technology is used in some different projects such as: tomcat, jetty, Java Caching System (JCS), OpenSymphony OSCache, Group Communication Toolkit and JBoss [49-50].

4.2.2 LAN with JGroups

As described in 4.2.1, JGroups uses a protocol stack that can vary from one implementation to another. The one used in the network implemented has the following protocols:

- UDP – is a transport protocol that implements IP multicast transport based on UDP (User Datagram Protocol). With this protocol it is possible to receive messages and to send messages both unicast and multicast messages.
- PING – it is a discovery protocol, when a peer wants to join a group, it sends a PING request to an IP multicast address, and receives from the coordinator the PING response with the view of the group (i.e. a list of all members of the group).
- MERGE2 – JGroups also allows the existence of sub-groups, the Merge2 protocol makes possible the discovery of all sub-groups. This protocol uses a discovery protocol (in this case PING) to discover all coordinators of a group, if more than one coordinator is found it sends to the upper layers a MERGE event with the views of the sub-groups.
- FD_SOCK – FD_SOCK stands for failure detection based on sockets. With this protocol, each peer creates a server socket and announces its addresses (the JGroups address and server socket address) to the multicast channel. This protocol is ring-based, each peer connects to its neighbor on the right, when the peer detects that the neighbor socket is closed it considers the neighbor a suspect.
- FD_ALL – This protocol is, like the FD_SOCK, a failure detection protocol, but unlike FD_SOCK, it uses a heartbeat protocol. Each member of the group maintains a table of all members, and every time it receives a heartbeat from one peer, it resets the timestamp of that peer. If the timestamp exceeds a determined amount of time, it is considered a suspect.
- VERIFY_SUSPECT – When a member is considered a suspect by a failure detection protocol, this protocol checks if the member is really dead, if it is dead than a SUSPECT event is sent to the upper layers.

- NAKACK – This protocol assigns a monotonically increasing sequence number to all multicast messages, when a member detects a missing multicast message it requests the message retransmission. The message retransmission is made by unicast.
- UNICAST – UNICAST is responsible for the reliability of the unicast messages, it works similarly to TCP, providing lossless transmission of unicast messages. Like NAKACK each message is tagged with a sequence number. However, in UNICAST protocol, there are no message retransmission requests, because there is acknowledgements for each received message instead. The sequence number of each message is used for the message sorting, and to identify the message received in the acknowledgment. When the sender detects that a message was not received (when the ACK was not received) it retransmits the message.
- STABLE – This protocol is responsible to check if a multicast message was rightly received by all members. Each member sends, periodically, its highest message sequence number, when it is detected that a message was received by all members. A STABLE event is sent to the upper layers.
- GMS – the group membership protocol, it is the protocol that handles with joins, leaves, suspicions of crashes ("suspect" messages) and sends new views of the group. It defines also the group coordinator, usually the first node of the view, which is the responsible for the update of the view.
- FC – This is a simple flow control based on a credit system. Each peer has a number of credits, in other words, a number of bytes it can send. If it runs out of credits, then the sender blocks, until the receiver gives more credits to it.
- FRAG2 – This protocol fragments one large message into many small messages, on the sender side, and it reassembles the fragments into the large message, on the receiver side.
- STATE_TRANSFER – This protocol allows the state transfer between members. A member "A" shall send a state request to another member "B", and "B" will responds with its current state.

With the protocol stack previously described, it is possible to have reliable multicast and unicast connections, with the possibility to have access to a list of current active peers (thanks to the GMS protocol), transference of states (being the STATE_TRANSFER protocol responsible for that), automatic discovery of other peers (using the PING protocol), failure detection (thanks to the FD_ALL, FD_SOCK and VERIFY_SUSPECT protocols), preventing at the same time the starvation of a peer if another peer had large amounts of data to transfer (being the FC protocol responsible for this prevention measure).

In Figure 4.3 is presented the architecture of the network implemented. As it is shown, it is a fully decentralized architecture, once there is no central entity. To achieve this architecture we had to assure that each peer had knowledge of all the others. This presupposition was only reachable thanks to the typical small number of peers in a LAN of a medical institution. Otherwise, with a big number of peers, it would possibly be unaffordable to maintain a shared list of all peers.

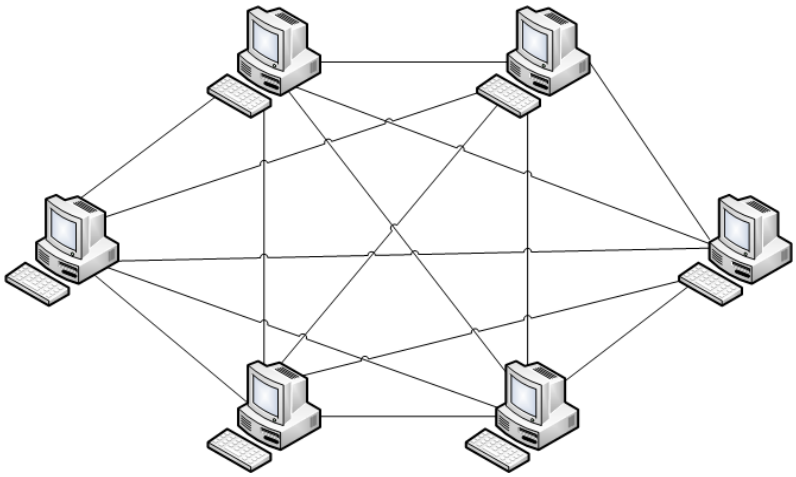

Figure 4.3 - Diagram of the architecture of the LAN network implemented

4.2.3 Messages Specification

After a reliable basis was assured with JGroups the first step was to define the messages interchanged. In order to provide the search and the file transfer features, four kinds of messages were defined: Query Request, Query Response, File Request and File Response. In Table 4.1 is presented which kinds of messages are used in each service. The Query Request and the Query Response are used for the lookup of resources and the File Request and the File Response are used for the file transference.

Table 4.1 - Table of the relationship between the messages and the features they were made for

Feature	Request	Response
Search	Query Request	Query Response
File Transfer	File Request	File Response

The Query Request message carries the query and associated info, in order to specify the desired search. The Figure 4.4 presents an example of this kind of messages. They are in XML (Extensible Markup Language) which is a flexible text format designed to meet the needs of large-scale electronic publishing, playing an important role in data exchange on Internet. These messages have the following parameters:

- MessageType : this is the parameter that defines the message type. In Query Request messages, this field is always filled with the string value "Query".
- QueryNumber: this parameter is used for the identification of the query. It is filled with the string representation of an integer which is a unique identifier of the queries sent by one peer.
- Query: this field is filled with the question of the search. This query must be compatible with the Lucene engine syntax, once each peer will use it to query its local engine.
- Extrafield: a query request message can have a variable number of extrafields. Each one of these elements must have the name of the indexed fields. The message sender wants to receive existent information about those fields, for each hit of the search. Taking the example of the Figure 4.4, each result that matches the query

must be described with: patient name, patient ID, modality, study date and thumbnail.

```
<?xml version="1.0" encoding="UTF-8"?>

<Message>
        <MessageType>Query</MessageType>
        <QueryNumber>1</QueryNumber>
        <Query>*:*</Query>
        <Extrafield>PatientName</Extrafield>
        <Extrafield>PatientID</Extrafield>
        <Extrafield>Modality</Extrafield>
        <Extrafield>StudyDate</Extrafield>
        <Extrafield>Thumbnail</Extrafield>
</Message>
```

Figure 4.4 - Example of a Query Request message

Query Response is a message that contains the hits of a search. When a peer receives a Query Request message, it interrogates local index engine, and sends the results in a query response message. As it is presented in Figure 4.5, these messages have a XML structure with the following fields:

- MessageType: as the Query Request, the Query Response has also this parameter. The response messages will fill this parameter is the string “Query Response”.
- QueryNumber: this parameter is the responsible for the association between Query Request and its response. This field is necessary to be able to ignore responses to expired queries.
- SearchResults: this element consists of a SearchResult items list.
- SearchResult: the SearchResult element contains all the information requested about a single query hit. It has the fields: FileName (the name of the file that matches the query), FileHash (a 16-byte hash of the file), FileSize (the number of the bytes of the file), Extrafields (list of the elements requested by the Query Request message, in this example: PatientID, StudyDate, PatientName, Modality and Thumbnail. The last one is encoded in a base 64 code).

```
<?xml version="1.0" encoding="UTF-8"?>

<Message>
        <MessageType>Query Response</MessageType>
        <QueryNumber>1</QueryNumber>
        <SearchResults>
                <SearchResult>
                        <FileName>6.dcm</FileName>
                        <FileHash>c521e1e632ec5de9a035eddfe1f004e4</FileHash>
                        <FileSize>18881936</FileSize>
                        <Extrafields>
                                <PatientID>20030672</PatientID>
                                <StudyDate>20030521</StudyDate>
                                <PatientName>Rocha^Camilo</PatientName>
                                <Modality>XA</Modality>
                                <Thumbnail>/9j+q2g…uo25JgAf//Z</Thumbnail>
                        </Extrafields>
                </SearchResult>
        </SearchResults>
</Message>
```

Figure 4.5 - Example of a Query Response message

The File Request is a message that shall be sent when a peer wants a file of another peer. The Figure 4.6 presents an example of a file request. As it is shown these messages are XML messages, and have the MessageType field (filled with the string “File Request”) and the element “FileRequested” that describes the file wanted by its name and hash.

```
<?xml version="1.0" encoding="UTF-8"?>

<Message>
        <MessageType>File Request</MessageType>
        <FileRequested>
                <FileName>6.dcm</FileName>
                <FileHash>c521e1e632ec5de9a035eddfe1f004e4</FileHash>
        </FileRequested>
</Message>
```

Figure 4.6 - Example of a File Request message

The File Response message is sent in response to a file request message, and contains the stream of the file requested, the name of the file, the number of the file fragment and a flag indicating if it is the last fragment or not. The stream can be sent in one message, or fragmented, depending on the size of the file.

As previously referred, XML is a text format. Therefore, the data contained in a XML document are interpreted as a sequence of characters. However, some characters are forbidden by the standard. Therefore, the content of a file being interpreted as text could contain forbidden characters and then the XML be considered invalid. To solve this problem, the solution would be to encode the file stream with the base-64 encoding. This kind of encoding, encodes binary data

(which text representation could contain forbidden characters) into a sequence of allowed characters. Such an encoding brings problems of performance:

- The encoding and decoding can be a slow process, increasing the latency of communications;
- This encoding increases the amount of data to be sent in about 33%, consequently, the time taken in the file transfer will be greater.

Taking into account the previously described reasons, we have decided to send the content of the files as binary data, instead of sending it in a XML document like the other messages.

4.2.4 Message Builder

After the definition of the messages it was necessary to develop a mechanism that could build the messages. For that reason, it was created a class responsible to build the messages. This class (MessageBuilder) has a public method for each kind of messages. Those methods receive the information needed to the construction of that kind of messages and returns the Message ready to be sent. Therefore, the builder has the following methods:

- BuildQueryRequest: this method receives the string formed by the query and a list of extrafields. This method assigns an ID to the query and then builds the XML document using the Dom4J library. This method returns an instance of MessageI which is an interface that must be implemented by all messages.
- BuildQueryResponse: this method receives a list of search results and the number of the query this message answer to. Each search result has a string with the filename, a string with the filehash, another with the size of the file and a hashtable that relates each extrafield with its value. Like the BuildQueryRequest, this method uses Dom4J to build the XML document and it returns an instance of MessageI, i.e. the message to be sent.
- BuildFileRequest: the parameters of this method are: the filename string and the filehash string. With both strings the XML document is built using the same library as the other two building methods previously described. At the end, an instance of MessageI is returned as well.
- BuildFileResponse: it receives a byte stream containing the data to be sent, the number of the file fragment and a boolean indicating if this message contains the last fragment of the file. All these parameters are used in order to constructs an instance of a MessageI that is returned by this method.

4.2.5 Message Handling

After the definition of the messages and their building, it was necessary to develop mechanisms that could handle all kinds of messages. To achieve that, it was created a handler mechanism for each kind of message. Firstly, an instance of a generic handler is invoked. Secondly, the generic handler gets the type of the message received and delivers the message to the handler specialized in that kind of message. In other words, if the message is a Query Request it is delivered to the Query Request Handler, if the message is a Query Response it is delivered to the Query Response Handler, and so on. For a better understanding of this process, it is visually presented in Figure 4.7.

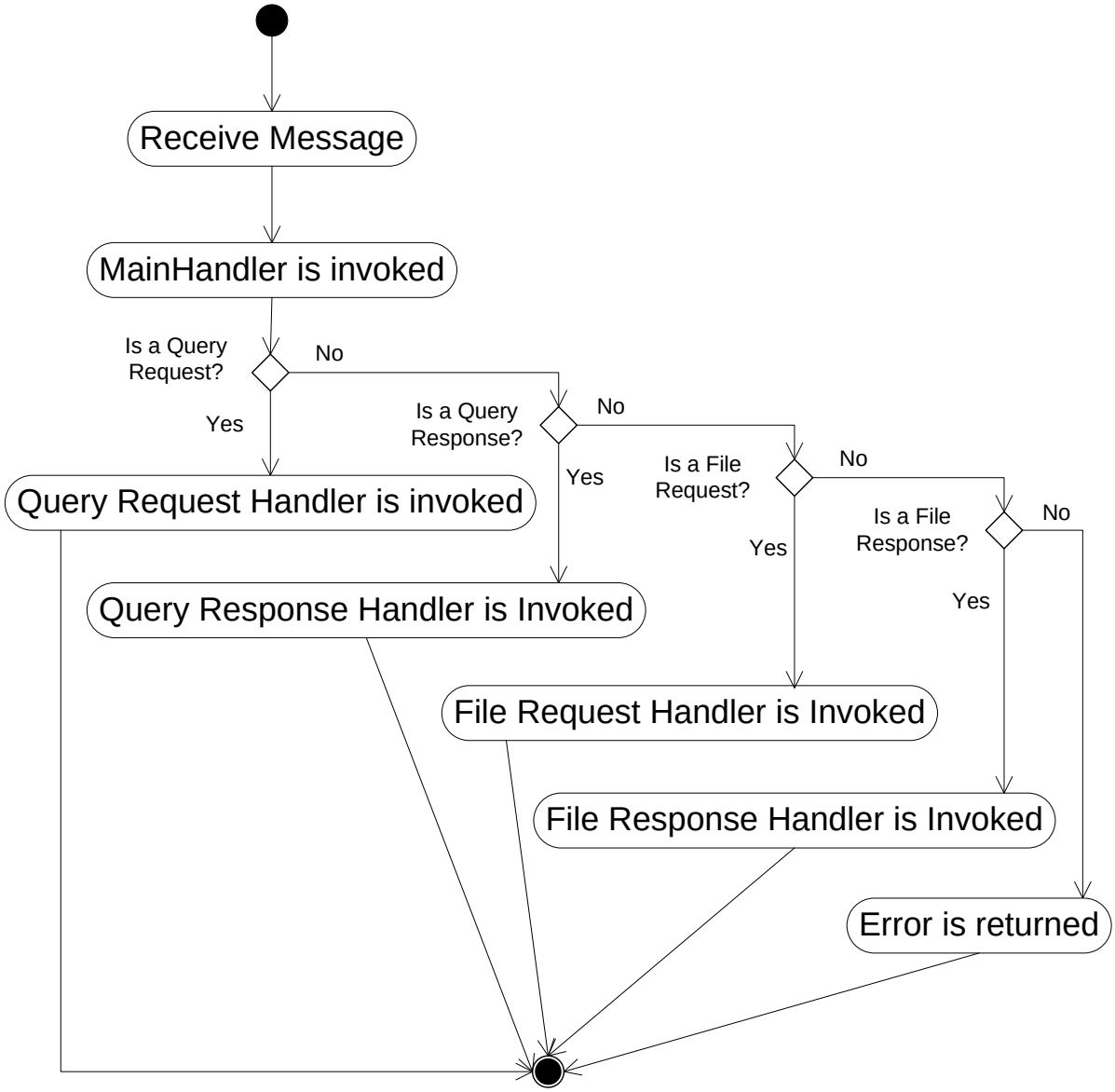

Figure 4.7 - Activity diagram of the message handling mechanism

The mechanism was built in this way, in order to facilitate future upgrades in the implementation. Once, creating and integrating the handler of new kinds of messages is as easy as develop a class to handle the new kind of message and make simple changes in the generic handler so that it can recognize the new kind of message and forward the messages of that kind to their handler.

4.2.6 Searching Mechanism

At that moment of the implementation, it was already implemented a network structure to build and handle messages. By this time, it was possible to build higher-level services and the first one was the searching mechanism.

The possibility to search for a medical image is one of the most important features of the application. As it is described in 3.1.1, when a search is made it is important that the results

actually contains all the resources that match the search parameters. This was possible to achieve with the infrastructure JGroups-based previously described, once JGroups guarantees that every message is received by its destinations. Each Query Request is sent to all network peers and, in their turn each contacted peer queries the local index and sends the results to the origin of the query.

In Figure 4.8, it is presented the process steps of the search mechanism implemented. Firstly, it is needed to build the Lucene Query based on the search parameters specified by user. Secondly, the Query Request message is built containing the query and other needed data and then sent to all the other peers through the multicast feature of JGroups. When a peer receives a message it dispatches the message to the handling mechanism. In this case, Peer 2 and Peer 3 receive a Query Request message, so the QueryRequestHandler would be invoked to handle that message. This specific handler gets the Lucene Query and the Extrafields (explained in 4.2.3) from the message. With that information, each peer that received the message (Peer 2 and Peer 3) queries his local indexes. From the results returned by the local index it is built and sent one or more Query Response messages to the query sender peer, depending on the number of Search Results. When the query sender peer receives a Query Response message, the handling mechanism gets the Search Results from the message and returns to the application layer the results through an Observable. An Observable is a model-view paradigm where an entity has a list of observers and some data. When the data of that entity is changed, the observers are notified. Therefore, with the Observable it was possible to have low coupling between the P2P layer and the application layer.

When a Query Response message arrives, the search results are collected from the message, they are put in the observable and, consequently, the observers are notified. In this implementation case, the observer is the application layer that once notified gets the search results from the observable. Those results are added to a list with all the results and finally presented to user.

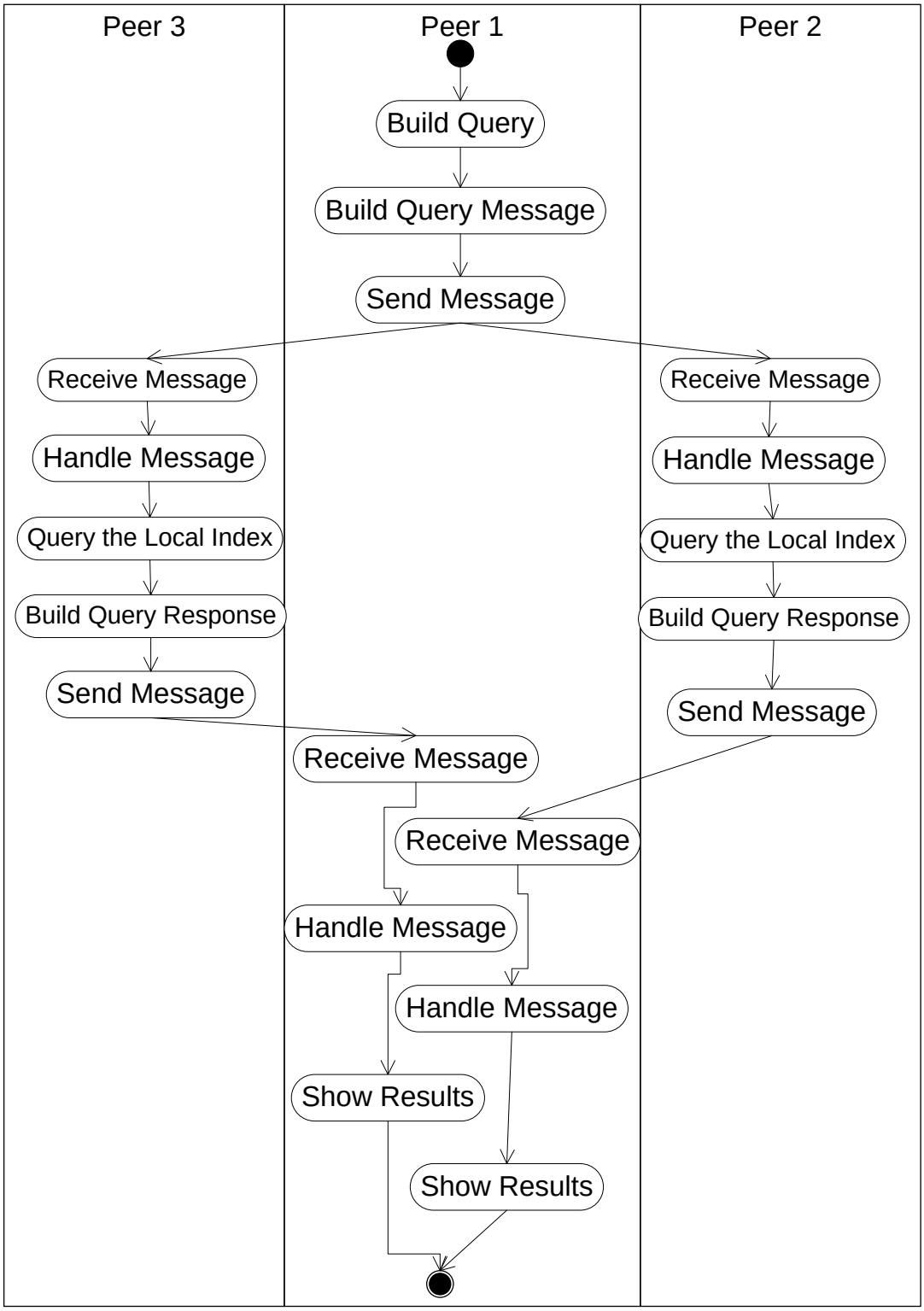

Figure 4.8 - Activity diagram of the searching mechanism

4.2.7 File Transference Mechanism

The search mechanism would be of little use if the peers could not share the files between them. Therefore, another high level service implemented was the file transference mechanism.

The Figure 4.9 describes the steps of the file transfer process. As described in 4.2.3 the files requested are identified by its name and hash. So, firstly, the receiver (Peer 1) gets from the search results list the filename and filehash to build the File Request message. That message is sent to the peer that advertised the file (Peer 2). When Peer 2 receives the message, it is dispatched to the handling mechanism. The handling mechanism gets the filename and filehash from the message and queries the local index for the path of the file with those characteristics. Once the file path is known, the File Response is built with the content of the file and sent to Peer 1. When Peer 1 receives the message it dispatches the message to the handling mechanism. That creates a new file with the content of the message, saves it, indexes it on the local index and, optionally, opens the file with the default DICOM file viewer.

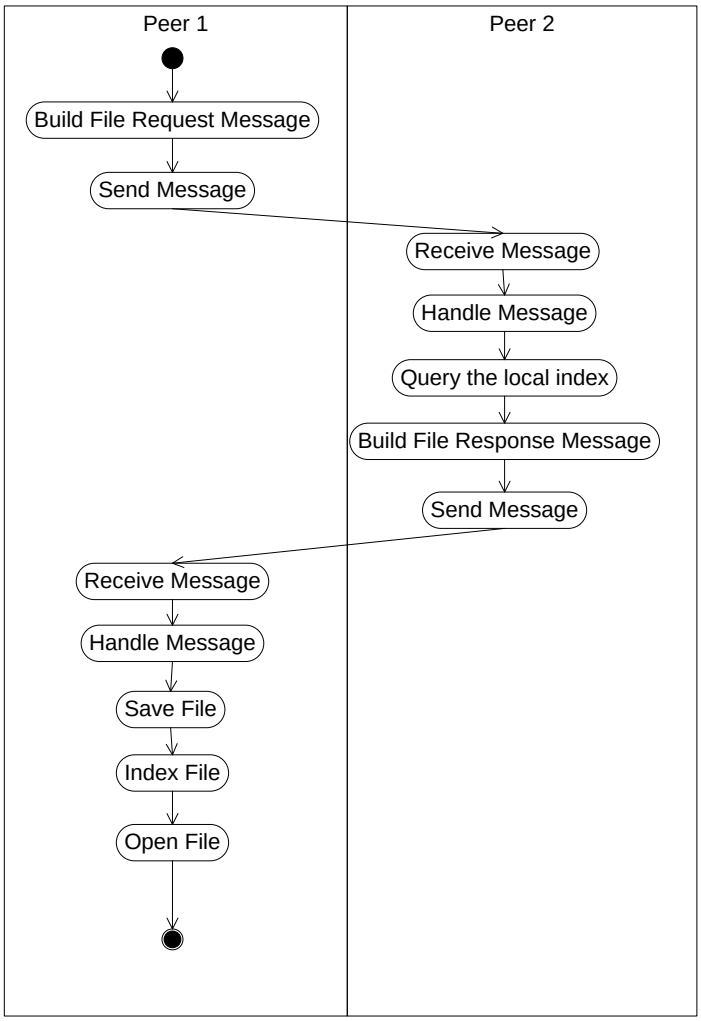

Figure 4.9 - Activity diagram of the file transfer mechanism

4.3 WAN Architecture

The WAN scenario is very different from the LAN one. As described in 3.2, a hospital network can be highly restrictive in external communications either by firewall either by a router. On one hand the firewall restricts the incoming and outgoing traffic, where only the traffic that follows certain rules is allowed to pass. During the work of this thesis it was considered that the TCP port 80 (HTTP port) is open for HTTP requests from the inside to the outside and also its respective responses from the outside to the inside. On the other hand, the router (if it was not

previously configured) prevents the communication from the outside to the inside. If a peer is behind a router it is necessary that the peer starts a session in order to be able to receive messages. With no previous configurations, it is only possible to start a session from the inside to the outside. These facts make direct communications between peers of different hospitals impossible, if the two hospitals are protected by firewalls and routers. It is unthinkable to design such a network for medical environment not taking into account the common restrictiveness of hospital networks.

The aim of this part of the project was to develop the application layer of a network that would support communications between peers of different institutions, even in adverse scenarios. The solution found was the relay strategy. In other words, we used a host somewhere in the Internet that is able to accept communications from anywhere. Therefore, this point acts as a bridge of communications between peers. The peers send the messages to the relay service and, in its turn, the relay service forwards messages to their destination.

In Figure 4.10, there are represented two sequence diagram. In (a) it is represented the impossibility of communications scenario. In this case, the peers of different institutions are protected by a router and/or firewall. Without previous router/firewall configuration is not possible to send directly a message from one peer to another. On the other hand, in (b) it is represented the solution to the inter-institutional communications problem, i.e. a relay service. This entity allows the communications between peers even if they are behind a firewall or a router (if the traffic matches the firewall rules).

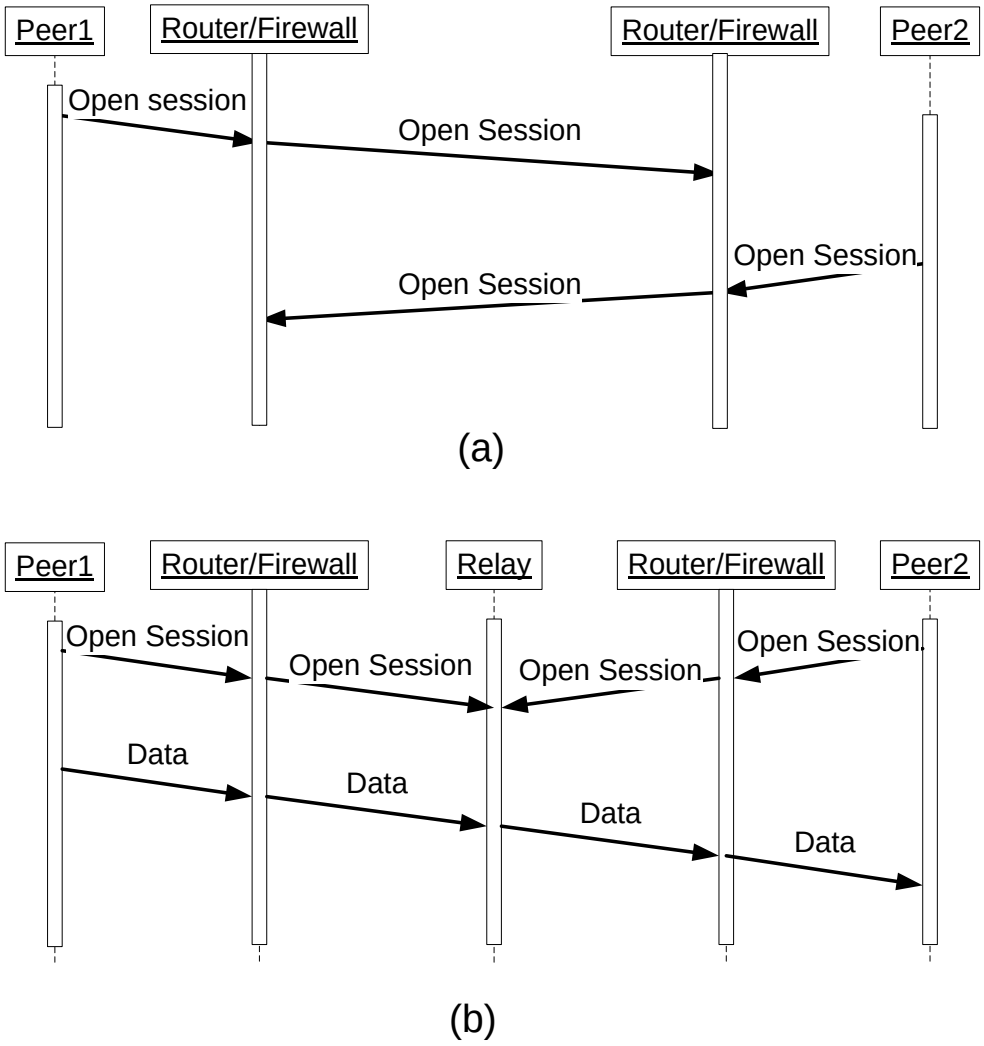

Figure 4.10 - Sequence diagrams of inter-institutional communications: (a) without relay service and (b) with relay service

During the investigation and development process, two alternative implementations of the relay service were made in distinctive environments: dedicated server and cloud. However the dedicated server implementation was discarded, both implementations will be described in the following sub-items.

4.3.1 Relay Service in a Server

The first implementation used a server as a relay peer. Each peer started a TCP session with the port 80 of the server. Once the TCP session was created, the peer was already able to send HTTP Requests to the server. Then, the server would transform the HTTP requests into HTTP responses and forwards it to the destination.

To make this happens, it was used the Socket abstraction. On the server side, for each request to open a session, a new socket was created, to receive and send messages. The server maintains a list of opened sessions. When it receives a Query Request message, the server forwards the message to each member of the list of opened sessions. Each of the other messages (Query Response, File Request and File Response) is sent only for one peer.

In Figure 4.11, it is presented a sequence diagram of a possible example of message interchanging between two peers and the relay server. As we can see, initially, each peer open the TCP session, when the server receives a message and has no one to forward it, the message is discarded. Otherwise, the message is sent to its destination immediately.

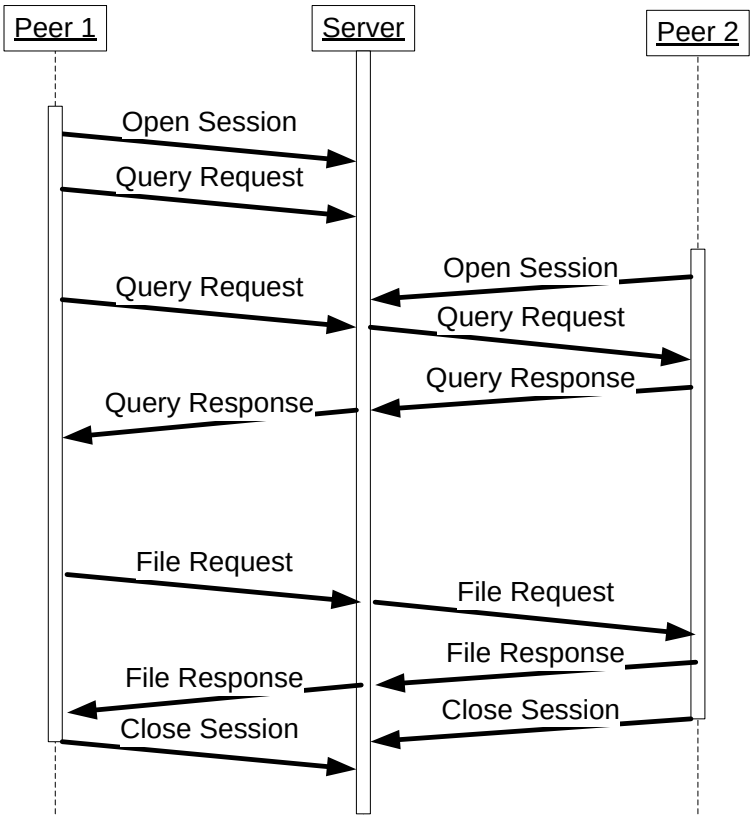

Figure 4.11 - Sequence diagram of an example of message communications between two peers with a relay server

In conclusion, with a relay server the messages are immediately forwarded to its destination. However, an architecture based on a server has also several disadvantages, such as: (1) the server is a point of failure, (2) the performance of the network is limited by the server capabilities (i.e. the server is a potential bottleneck of the network), (3) vertical scalability (in other words, to improve the network capabilities it is needed to upgrade the server) and (4) it is necessary a server with associated costs (infra-structure, maintenances, etc). As a consequence, this solution did not seem a good one for this purpose. So, the next step was the study of relay solutions based on services infrastructures.

4.3.2 Relay Service in a Cloud

A cloud computing service is a distributed system technology that consists on the aggregation of resources that are distributed into one single system, aiming to virtualization (in other words, decoupling the business service from the infrastructure needed) and scalability (i.e. the capability of the system to grow when it is needed) [51]. Besides, one of the greatest advantages of the cloud computing is about its resiliency. In theory, the cloud computing services are built in such a way that if a machine fails, the system readjusts itself so the user will never know that one

machine failed. Taking into account this resiliency, cloud computing seems a useful technology to ensure some level of stability of the network that a single server cannot provide.

Therefore, the relay service built over a cloud computing service would be, theoretically, more stable and more suitable to the concept “share medical imaging at anytime, at anyplace”, once the relay service is more likely to be functional when it is needed if it is built over a cloud service than in a single server.

Some cloud services were considered, such as: EC2 (Amazon Elastic Compute Cloud) [52], Rackspace Cloud [53], Google App Engine [54] and Windows Azure Platform [55]. The Google App Engine (GAE) was the platform chosen, once it provides a platform relatively stable with automatic scaling, not needing any system administration. Those were important features we were interested on, once we were trying to achieve a solution off-the-shelf. Besides, GAE provided the possibility to create an account and publish an application free of charges. This advantage of GAE made it easy to test and facilitate small usage scenarios of the system implemented.

4.3.2.1 Google App Engine

Google App Engine is a “PaaS” (i.e., Platform as a service) cloud computing service. Therefore, GAE enables users to run and host their own web applications on Google’s infrastructure [56], bringing to the applications the advantages that such an infrastructure can provide as, for instance, automatic scaling and load balancing, the geo-distributed architecture and fault tolerance [57].

In Figure 4.12, it is presented a diagram of the GAE Architecture. Basically, the application developed has access to some services, and uses them in order to respond a request done by the application client. Currently, the application can be developed in standard Java technologies (including JVM , i.e. Java Virtual Machine, Java servlets and the Java programming language) or Python (it is provided a Python runtime environment that includes a Python interpreter and the Python standard library) [54].

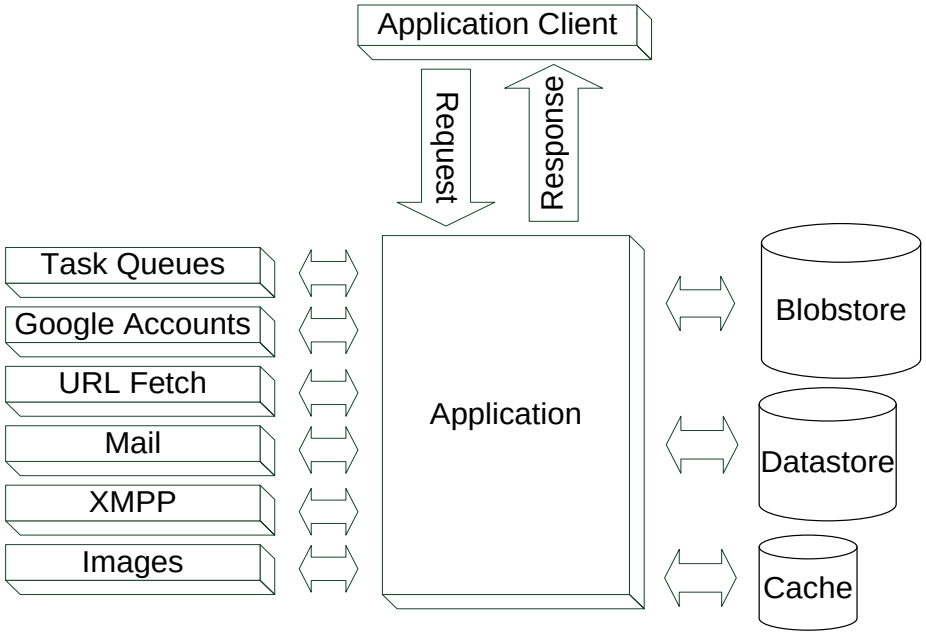

Figure 4.12 - Block diagram of the Google App Engine Architecture

As presented in Figure 4.12, GAE applications developers have available several complementary services:

- Task queue is a service available for the applications hosted in GAE that gives the applications opportunity of process some small pieces of background work (tasks) outside the scope of a request. However, these tasks have to be initiated by a user request.
- GAE also permits the authentication of users with the Google accounts. So, it is possible to test if the user is signed in with a Google account, or redirect the user to the Google accounts sign-in page.
- URL fetch is a way of communicating in a single request to a third party host on the internet, using GAE. Once it is not possible, using this service, to make connections directly using sockets.
- It is also possible to send email messages using the mail service. The email messages are received through HTTP requests.
- Extensible Messaging and Presence Protocol (XMPP) [45] (i.e. an open technology for real time communications, widely used in, for instance, instant messaging and collaborative tools) is also supported by this platform through the XMPP API, therefore it is possible to send (and receive) messages to (and from) other XMPP-compatible service.
- The image service API deals with image manipulation like, for instance, resize, rotate, crop and flip. The file formats supported are JPEG and PNG.

Besides the services described, GAE supplies three data storage: datastore, cache and blobstore. The datastore is a robust and scalable repository, focused in read and query performance. It replicates all data between multiple storage locations to guarantee the access to the information at any time. This service uses an optimistic concurrency control, i.e. it assumes that concurrent requests do not interfere, but if it happens rollback is made. Besides, it does not use a traditional relational database, once the data unit is not a row of a table, but an entity (aggregation of a kind and properties). Moreover, there is no relational table (there is kinds instead). The queries are made using a language similar to SQL (Structured Query Language) named GQL that enables the filtering and sorting of the entities stored.

The GAE cache service is another storage mechanism used to improve the performance of the application. It is a temporary storage area of rapid access [58]. However, the biggest disadvantage of this storage area is the low scalability. It has limited size and when the maximum capacity is reached, the oldest entities are discarded.

Finally, the blobstore service allows the storage of entities larger than the maximum size allowed in datastore (one megabyte). In blobstore the data is not indexed. It works as an associative memory, so it is only possible to get from the blobstore with its key that it is returned when the blob is stored.

Other important characteristic of GAE service is that applications hosted in this service run in a sandbox, once they have limited access to the underlying operating system. This secure environment includes several restrictions such as:

- Only through URL fetch it is possible to access other computers on the Internet.
- Applications are only available through HTTP or HTTPS requests on the standard ports (80 and 443 respectively).

- It is not possible to write to the file system, and it is only possible to read files that were uploaded with the application code. To store information there must be used the App Engine datastore.
- The application code only runs to process a request and it has only 30 seconds to respond, after that time the operation is terminated by the environment.
- There is only one thread for each request, being not possible to run sub-processes.
- It is not possible to open sockets or access directly to another host.
- Datastore only allows inequality filters on a single entity property for each query.

Despite these restrictions, GAE provides an easy of using and scalable platform for applications hosting, with many services available that saves developers from spending time for example: in the set up of a SQL database and the consequently creation of tables.

The characteristics previously described were evaluated and it was concluded that GAE seemed to fulfill the identified needs of the relay service. Therefore, the relay service was implemented with the GAE API, as described in the following sub-item.

4.3.2.2 Relay Service in Google App Engine

As described in 4.3.2.1, GAE is a platform where it is possible to deploy an application that reacts to a request, processing the request and sending the response. With this technology there is no access to Sockets and we are limited to servlets technology to receive and send messages. Once the service only sends messages in response to another one, it is not possible to implement in GAE the strategy of automatic forwarding of messages implemented in a server (described in 4.3.1). In order to solve this problem, it was used a polling strategy, i.e. from time to time each peer asks the application hosted in GAE if there is any messages for it.

In Figure 4.13, it is presented a sequence diagram of an example of messages interchange between two peers, through the relay service deployed in GAE. As it is presented, from time to time, each peer sends the question: "are there any messages?" in the form of a HTTP get request. If there are any messages, the oldest one of the non-expired messages will be returned encapsulated in a HTTP response. Otherwise, the response will have no content. The messages sent from peers are encapsulated in HTTP post requests. Consequently, GAE stores the content of the HTTP post requests in the datastore service.

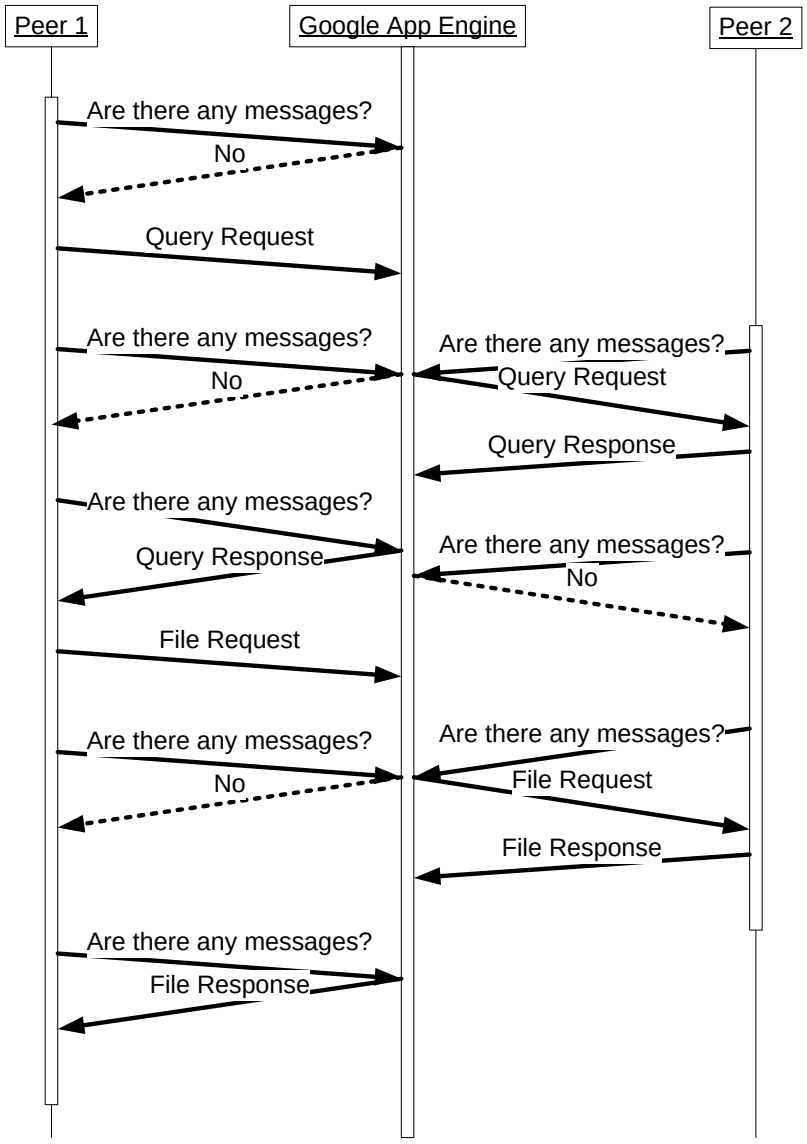

Figure 4.13 - Sequence diagram of an example of messages interchange between two peers, using the relay deployed in GAE

With this implementation the only things that changed are the way the messages are sent and the File Response messages. The building, handling and the higher level methods are the same from the LAN implementation. The File Response messages are different for security reasons. In LAN it is not very relevant if the files are shared encrypted or not. However, in WAN it is very important that the files cannot be accessible by others but the receiver. To achieve some level of confidentiality it was used an encryption algorithm that uses a hard coded key to encrypt shared by all peers the content of the files shared.

The limitations of the GAE (described in 4.3.2.1) brought some problems and engineering challenges. One of the biggest one was the time between the requests for messages. In other words, if each peer took too long to ask the application hosted in GAE for messages, the peer would take too much time to notice the existence of messages. On the other hand, if it took too short time to ask for messages, there would be too many messages interchanged. In both cases, the performance

of the network would be compromised. The solution found was an adaptive amount of time, between messages requests. If no messages are expected (i.e. the peer did not send File Requests of Query Requests) then the amount of time would be greater, otherwise it would be smaller.

Other problem was the recognition of messages that have already been sent to a specific peer to avoid duplicated messages transmission. To solve this problem, the messages with only one destination (for instance: File Response, File Request and Query Response) are automatically deleted when they are sent to their destinations. However, the messages that must be sent to all peers (Query Request) cannot be deleted after the forwarding to one peer. For that reason, two solutions were considered: (1) the application in GAE creates a copy of the message for each active peer and stores them, or (2) each peer identifies the last message received and the relay sends the next one, if it exists. The second considered solution was chosen, because storing a copy for each peer is not a scalable solution. If the number of peers grew, the number of messages stored would grow too. Besides, with the second solution, when a peer joins the network, it is possible for this peer to receive queries that had been sent before, if they had not expired. For those reasons, the second strategy was chosen. Therefore, the identification of the Query Requests is made in the following way:

1. When the application in GAE receives a Query Request, it stores the message with a timestamp of the moment that the message was received.
2. When a Query Request is sent from the relay to the peer, it contains the timestamp stored and the name of the peer that sent it to the relay.
3. When a peer requests for messages, it sends in the HTTP headers the name of the peer and the time of the last Query Request message received.

The Figure 4.14 presents a sequence diagram of an example of the query identification mechanism. When peer 1 joins the network it asks the GAE if there are messages with a timestamp bigger than 0. At that time, GAE has no stored messages so, it answers with a negative response. After some time, Peer 1 sends a Query Request message. Consequently, GAE stores that message with the timestamp of that moment (156). After a few time, Peer 2 joins and asks if there are messages with a timestamp bigger than 0. GAE answers with the Query Request stored and append to it the message timestamp (156) and the name of the peer that sent it. Therefore, Peer 2 receives the message and sends the response. After that, when Peer 2 checks for any messages stored, it asks for messages more recent than the one with timestamp 156 and sent by Peer 1.

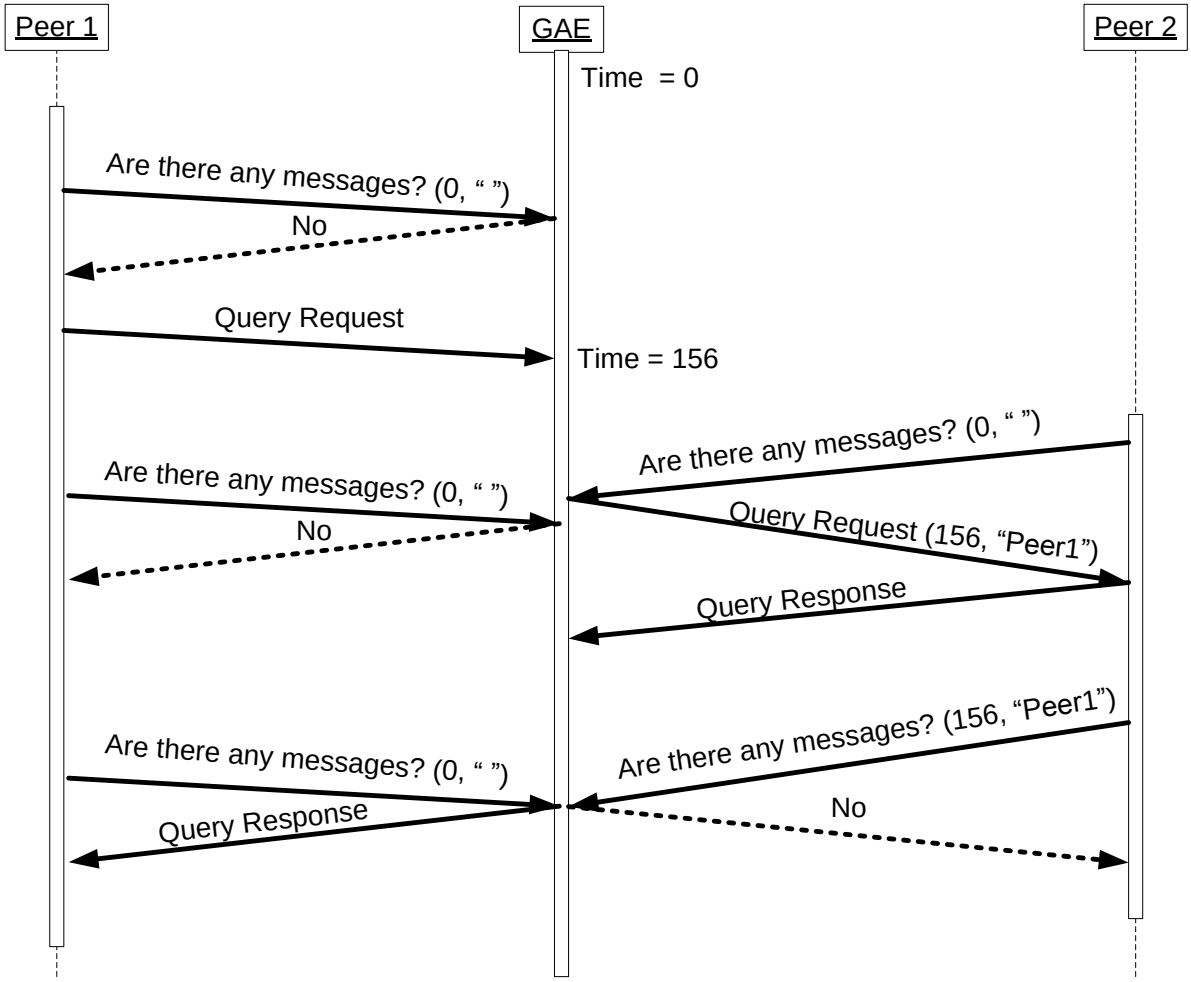

Figure 4.14 - Sequence diagram of messages interchange between two peers, presenting the solution for identification of the last received Query Request

Another important constraint is that Google App Engine restricts the size of incoming and outgoing messages to 10 megabytes. Besides, at the time the implementation was being developed, each entity stored had a maximum size of 1 megabyte, for free users. For this reason, the files are divided into segments of 900 kilobytes or less. This number was chosen in order to make possible the storage of other fields associated with the image. However, very recently the blobstore was availed for free accounts being one of the possible improvements that can be done in the future.

As far as the file transfer is concerned, it was firstly implemented with a single-threaded strategy. On the one hand, the sender sent each file fragment each time and waited for the HTTP response from GAE between each fragment. On the other hand, the receiver made polling for one fragment each time. If that fragment was already in GAE, the response would be that fragment. Otherwise, the receiver would wait some time and request again the same file fragment.

In Figure 4.15 is presented a file transfer sequence diagram between two peers. As it is shown in the figure, the file upload is faster than the download. This happens, mainly, because of

the time that Google App Engine takes to query the datastore and retrieve the result that is greater than the time that GAE takes to insert a new entity in the datastore.

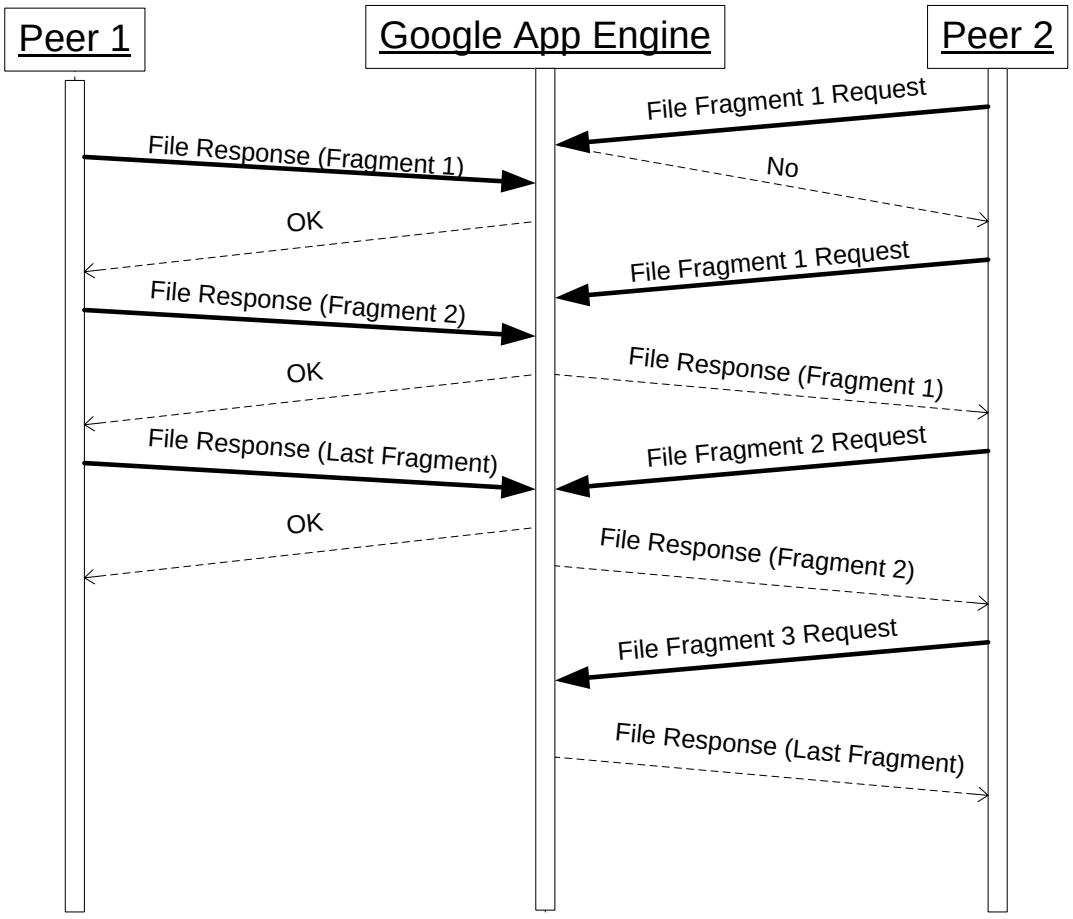

Figure 4.15 – File transfer sequence diagram between two peers, divided into three fragments.

The results of the tests made with single-threaded peers (senders and receivers) were not very promising results. So, it became critical to find a mechanism to improve the transfer performance. Taking into account that the download was visibly slower than the upload, it was clear that the download was mainly conditioned to the response time of GAE. Once GAE guarantees that its infrastructure can respond to some simultaneous requests without any loss of performance, multi-threaded receivers seemed the logical path to walk, i.e. the receivers would send more than one request at the same time. Therefore, the download rate would be more approximated to the upload rate, improving the performance of the file transfer. This mechanism is schematically exemplified in Figure 4.16. In that figure, Peer 1 is sending a file (divided into 5 fragments), one fragment at a time, while Peer 2 is a multi-threaded receiver (in this case a double-threaded receiver, i.e. it is requesting two fragments at a time).

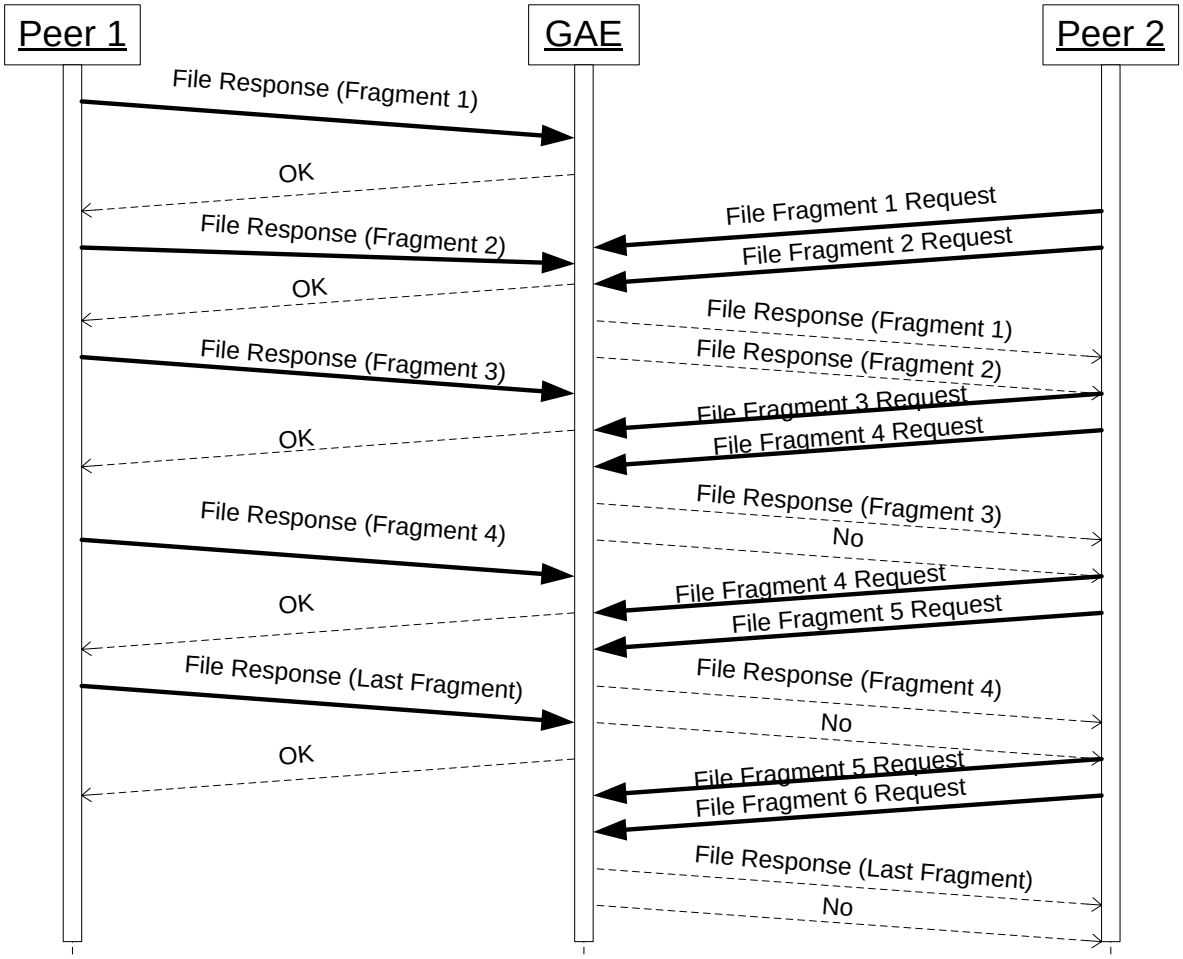

Figure 4.16 - Sequence diagram of a file transfer between a single-threaded sender and a multi-threaded receiver

The multi-threaded receiver improved the transference performance of the network. Therefore, it was empirically proved that this mechanism was a valuable asset to the system and, at the same time, a concept that was worth to be explored.

For that reason, the same mechanism was implemented in the sender, in order to achieve better upload speed and, consequently, making the file fragments available for download more quickly. Therefore, the download would be possibly faster (depending on the bandwidth and the number of threads in the receiver).

In conclusion, it was implemented multi-threaded sender and multi-threaded receiver, in order to achieve higher levels of performance of file transfers.

4.4 P2P Global Architecture

In conclusion, the proposed peer-to-peer network topology for medical imaging is hybrid (Figure 4.17).

On the one hand, the LAN is fully decentralized, where each peer communicates directly with all the others. The queries messages are flooded in the network in such a way that it is guaranteed that all peers receive the message and that query sender peer receives all results.

On the other hand, the WAN network is centralized in a cloud service. All messages passes through an intermediary (Google App Engine) in order to makes possible the communication between all peers even if they are behind firewalls or routers.

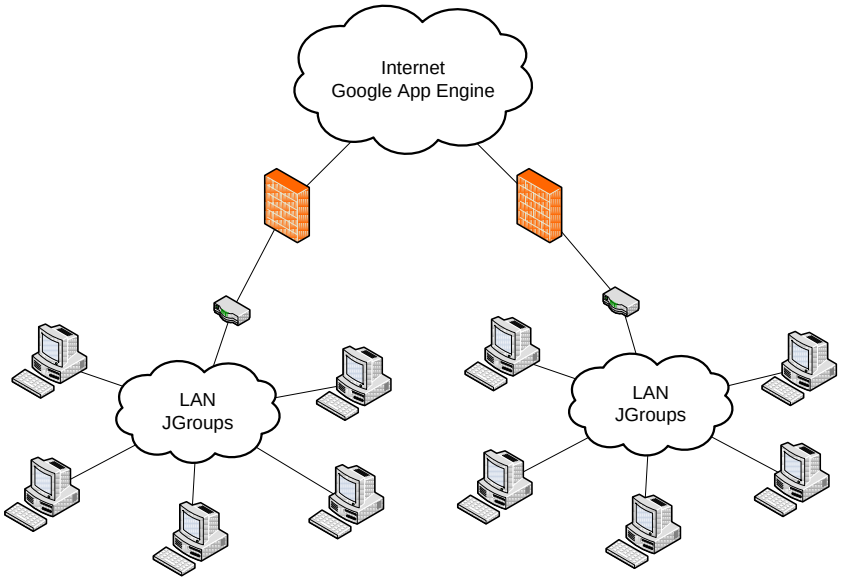

Figure 4.17- Diagram of the global P2P architecture

The application developed provides to users the opportunity to choose the search range: Local, LAN and WAN. If the user just wants to get the results indexed in the local index, he must select only the Local checkbox. If the LAN checkbox is selected the peers in LAN are queried. If the WAN checkbox is selected the query will be sent to the GAE so the search results retrieved will contain the results of the peers that are connected to GAE. Therefore, to obtain all the results possible it is needed to select all three checkboxes.

Since the beginning of the development, the mechanisms and processes were implemented in such a way that they were independent from how the messages were exchanged. Therefore, the association between LAN and WAN was relatively facilitated with solution designed. However, some problems appeared with the joint between these two parts of the network and the biggest challenge was associated with the propagation of queries. If the LAN and the WAN checkboxes were selected how to manage the Query Request messages so other peers of the LAN, that are also connected to the WAN, do not answer twice to the same query. The implemented solution is in WAN flooding process. The queries are flooded in LAN and sent to the relay service. However, the relay service only forwards Query Request messages for peers that have different IP addresses from the sender of the message. Therefore, only peers with different IP addresses from the query sender peer are able to receive the query and, consequently, answer to it (see Figure 4.18).

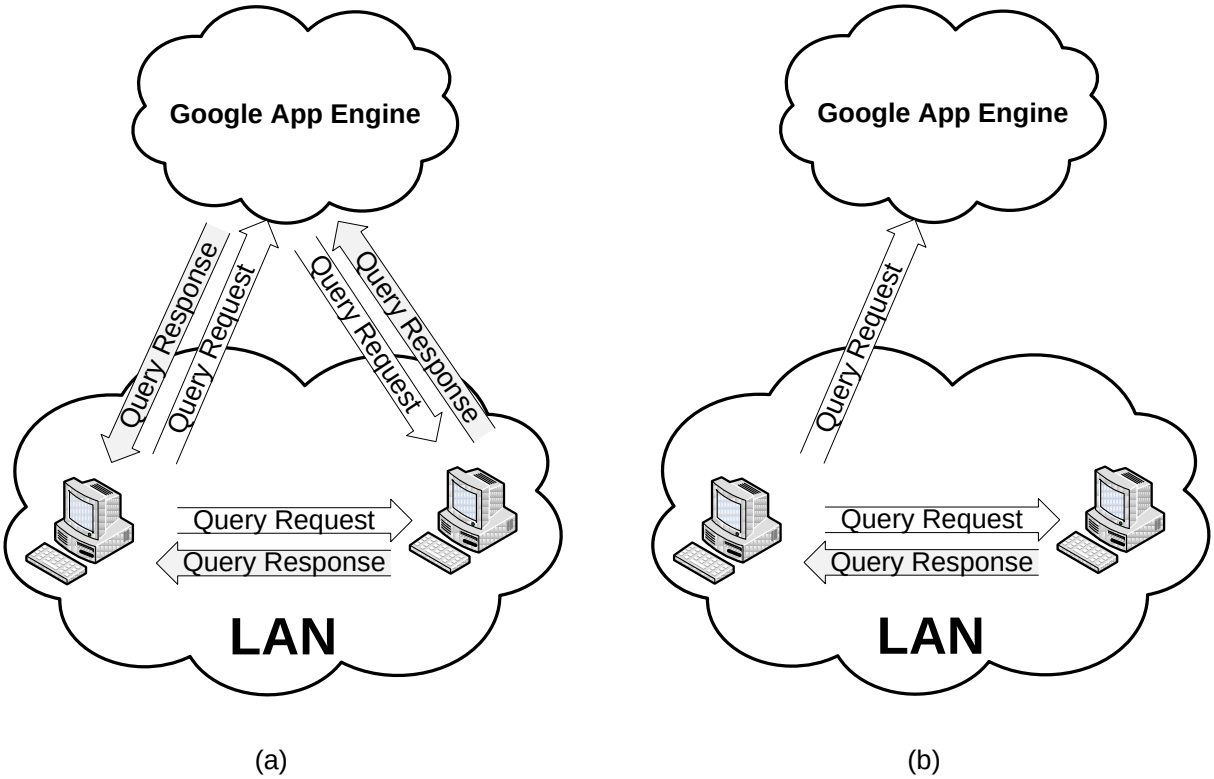

Figure 4.18 - Query propagation in a network with two peers in the same LAN and both connected to GAE: (a) with no IP control and (b) with IP control

5 Results

This chapter will present the final application results and will evaluate the peer-to-peer solution performance.

5.1 Applicational Solution

The application was developed in Java as an Open Source and OS (Operating System) independent project. This project conciliates P2P technologies with cloud computing services, in order to provide a file sharing platform that meets the needs of the medical imaging scenario.

The Figure 5.1 presents the GUI (Graphical User Interface) of the application. As it is shown, the interface is divided into three main areas: the options, the search form and the search results.

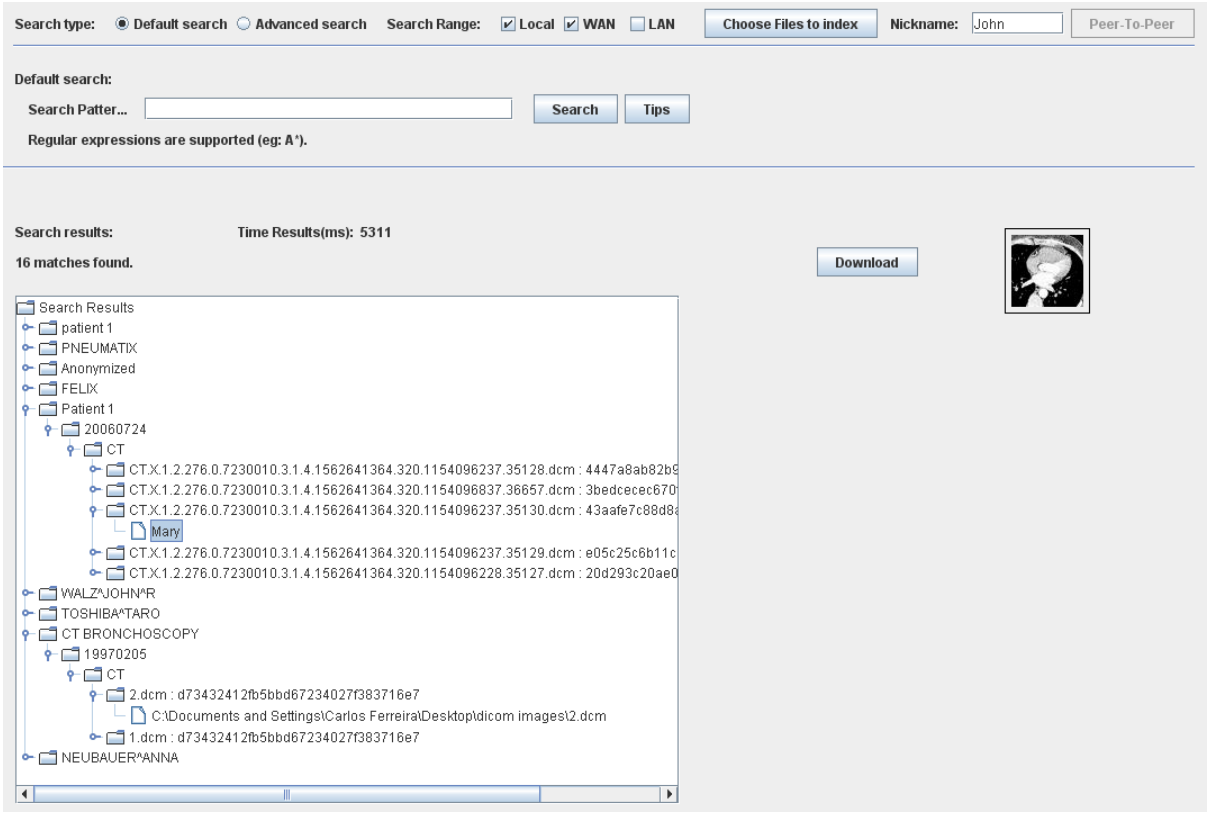

Figure 5.1 - Application GUI

In the options we can define:

- The Type of Search: if it is a Google-like search or a parameterized one.
- The Search Range: each search can cover three distinctive areas: the local index, the peers in LAN and the peers in WAN. This parameter defines which area will be covered by the search. If all three checkboxes are selected, than the search will cover LAN, WAN and local index.
- Choose files to index: this button leads to a window where it may be chosen the files index by the search engine.
- Nickname: in this text field it may be written the name by which the peer will be known in the network. If it already exists, the network will attribute a complement to the nickname in order to make it unique. For instance, if the nickname "Mary" already exists, then the peer will be named as "Mary (0)" if there is no one with that nickname.

Once it is not supposed that users know the Lucene syntax (i.e. the language used to query the search engine Lucene-based), the GUI provides two search forms. One (Figure 5.2) allows the search of studies which content contains the words or a regular expression specified by the user, for instance, it is possible to search for all studies which content contains the word "cardiac". Another (Figure 5.3) provides searches for specific attributes, for instance, it is possible to search for all studies performed between 2009-06-10 and 2009-06-15 which patient's name contains the word "Peters".

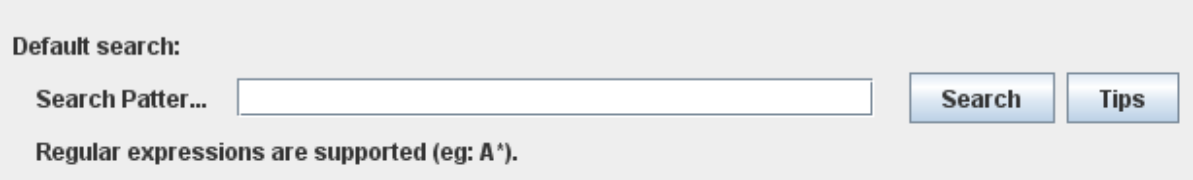

Figure 5.2 - Default search form of the application GUI

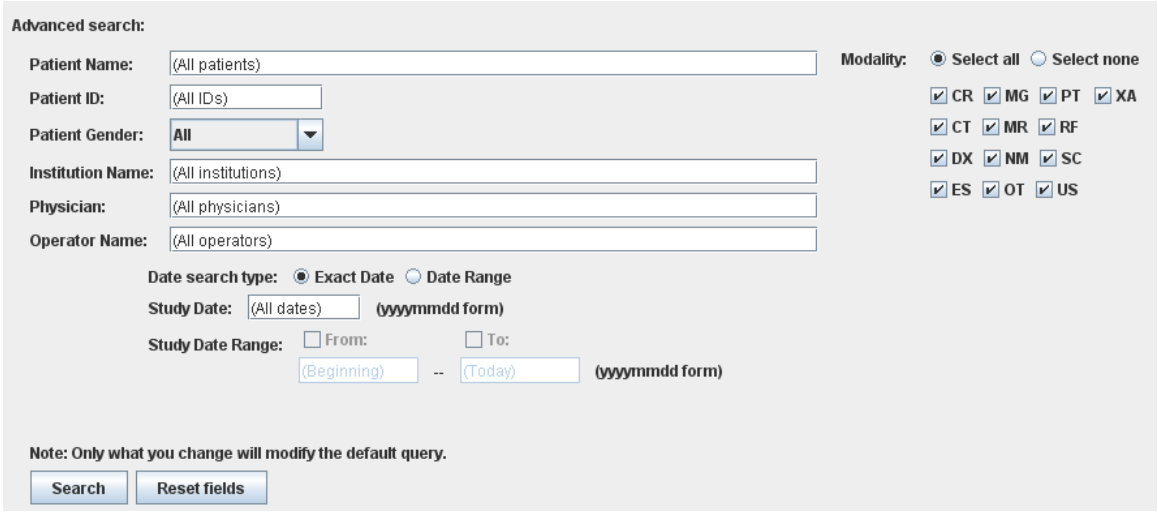

Figure 5.3 - Advanced search form of the application GUI

The results of a search are shown in a tree structure as shown in Figure 5.4. From the top of the tree to leafs, the fields shown are: the string "Search Results", the patient Name, the study date which means the date when the study was made, the modality, the files identification ([file name] : [hash]) and the source of the result.

In the example presented in Figure 5.4, two search results were found and the time that the system took to show the results was 84 milliseconds. The two search results have as patient name "CT BRONCHOSCOPY" (the images had suffered an anonymization process, i.e. a process where the data that refers to the patient are deleted or changed in such a way that would be difficult or impossible to relate the image to its subject). These studies were taken on 19970205 (i.e. 5th February of 1997) and the modality is CT (Computed Tomography). One of the filename is 2.dcm the other is 1.dcm, the code that follows the filename is a hash of the file content. I.e. the content of the file is processed in such a way that a difference in the content would generate a difference in the returned value. In this example the hashes are similar, so it is probable that the content is the same. In the end, "carlosferreira-34411" refers to the origin of the search result. In this case the file is in the peer whose ID is "carlosferreira-34411". As it is shown in Figure 5.4 at the right side there is a thumbnail, this thumbnail is a small representation of the image that is selected (in this case a thumbnail of the image contained in file which name is 2.dcm).

Figure 5.4 - Representation of the results in the GUI

The GUI also has a Download button that can be pressed in order to request the transfer of the studies, series and images selected. An external DICOM viewer can be configured to open those transferred images.

5.2 Performance Evaluation

In order to be able to evaluate the application performance, some tests were designed and performed.

5.2.1 Retrieve of Search Results

The first set of tests was made to have an idea of how much time the application takes to retrieve the results that match a query. It was tested under three different scenarios: (1) the search made was local, the peer only checks its local index (Local); (2) all peers queried were in the same institutional network (LAN); (3) all peers queried were somewhere in Internet (WAN).

In this set of tests, each peer was firstly equipped with an information system index that contained data about 41056 images. The queries were chosen, in order to test the network with different numbers of search results. Besides, the network was tested with different number of peers to have an idea of how network responds to the number of peers increase.

The taken times are the amount of time between the moment when user presses the search button and the moment when all the results were shown. In order to minimize accidental errors in the measurements, each test was made ten times and then the average value was calculated.

5.2.1.1 Searching in the Local Index

In order, to have an idea of the time that the local index takes to answer a query, we tested firstly the local index. The aim of this test was to evaluate the search engine performance with two

distinct variables: the number of results retrieved and the machine capabilities. To vary the number of results retrieved we pre-defined a set of queries that retrieved a specific and distinct amount of results. Moreover, to evaluate the effect of the machine capabilities in the searching process, the same set of searches were made in two different machines: the fastest machine was an Intel Core Duo E8400, with 3,0 GHz CPUs and 4GB of memory, the slowest machine was a Pentium4, with a 2,6 GHz processor and 512MB RAM. The information system index was equal in both machines and test results obtained are presented graphically in Figure 5.5.

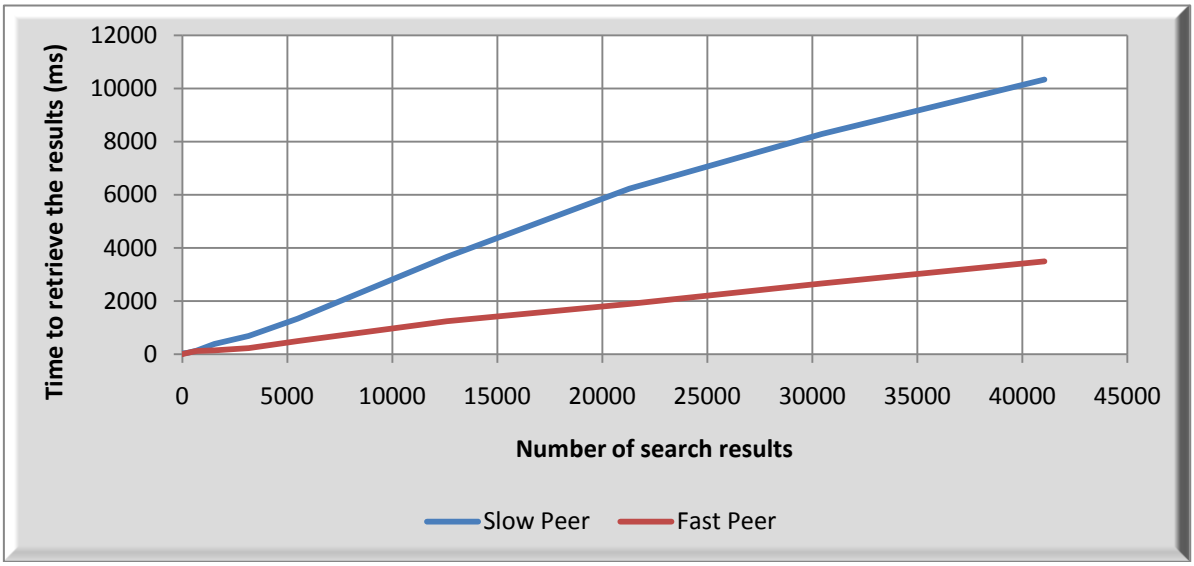

Figure 5.5 - Graph of the time that the local index takes to retrieve search results

From the graph analysis, it is clearly perceptible that the machine where the search engine is hosted has a big effect in the search performance. The fastest machine took about 3,5 seconds to retrieve 41056 results, whereas the slowest machine took about 10,3 seconds to retrieve the same number of query results.

5.2.1.2 Searching in LAN

The searching mechanism in LAN was tested in two perspectives: number of peers and number of results retrieved. It was used a 100 Mb LAN and four desktop computers with different characteristics, so we could test the behavior of the network with heterogeneous peers. To evaluate the influence of the number of peers in the searching performance, this test was made in three phases:

- Two peers – The searches were made in a LAN with two peers active. One of the peers was the query sender and the other the respondent.
- Three peers – With three peers connected, one of the peers made the queries and sent them to the other two peers.
- Four peers – The searches were made by one peer and answered by the other three peers.

Each one of these phases consisted on a set of queries, each one of those, specifically chosen to test the performance with a determined number of search results.

In Figure 5.6, it is graphically presented the test results of the search retrieval.

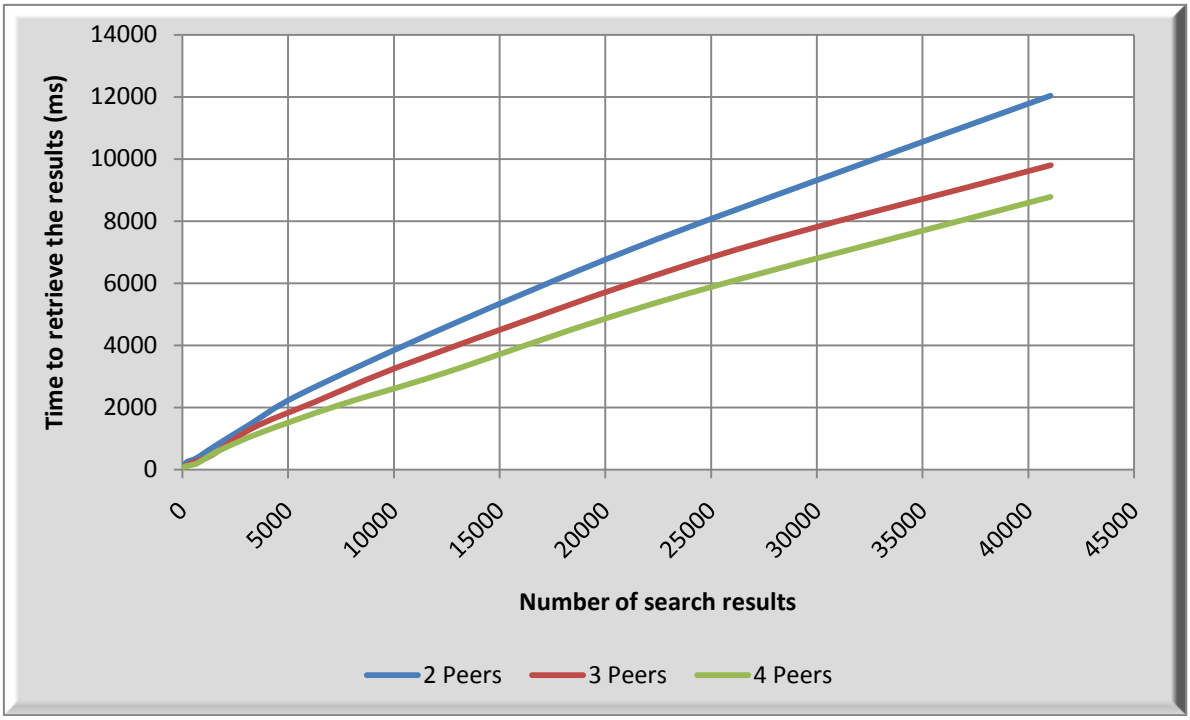

Figure 5.6 - Graph of the time that LAN takes to retrieve search results

As the graph shows, when the network has two peers (one query sender and one responder) it takes about twelve seconds from the moment that an user presses the search button until the moment he is able to visualize all the 41056 results. It is important to refer that the search results are sent in messages that contains up to 500 search results. Therefore, the user gradually receives the results, turning the waiting time until the reception of all the results more pleasant.

It is also perceptible that the performance of the search process is better with more peers. For instance, a search that retrieves 41056 results takes 8,8 seconds in a network with 4 peers and about 12 seconds in a network with 2 peers. This happened because each peer had the same index, so each peer answered the query with the same number of results. Therefore the computational effort the local index, building the XML messages and sending them is divided into two peers.

5.2.1.3 Searching in WAN

The searching mechanism in WAN was tested in a similar way as the LAN was tested as described in 5.2.1.2. We used a free account of Google App Engine to deploy the relay service and the same 4 machines and the same set of queries we used in the LAN tests. In Figure 5.7 it is presented the test results.

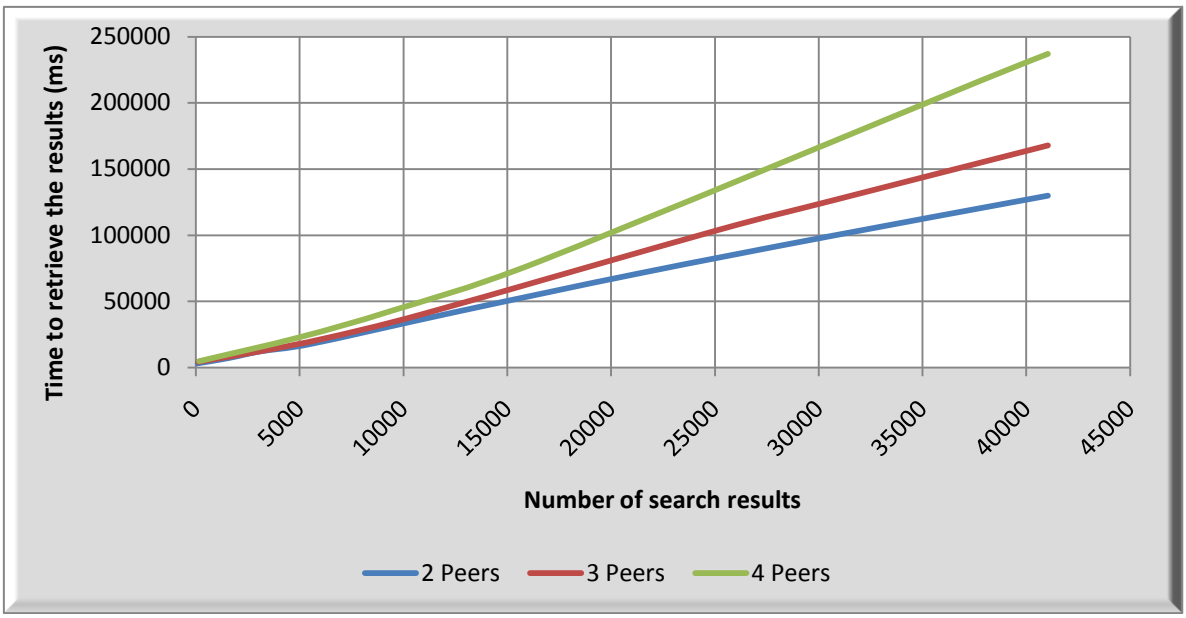

Figure 5.7 - Graph of the time that WAN takes to retrieve search results

As we can see in Figure 5.7, the searching time has a natural tendency to take longer with more peers, in opposition to the LAN reality. For instance, a search that retrieves about 41000 results in a 4-peers network takes about 237,4 seconds, whereas a search that retrieves approximately the same number of results in a 2-peers network only takes about 125,2 seconds. This tendency to worsen the searching performance with the increase of the number of peers is associated with the restrictions imposed by the GAE in free accounts. These restrictions define a set of maximum resources usage attributed for each application. Once we are using a polling strategy, with more active peers GAE will have more requests to respond. If it has more requests to respond and the same resources it will take more time to respond each one of them.

We believe that these results would be by far better if it was used a paid Google App Engine account. Since we would possibly have more processing power which would allow the application to send do a more intensive polling without an additional increase of communication latency.

5.2.1.4 Balance Between Local, LAN and WAN Searches

To compare all the three search domains: WAN, LAN and local search engine, we gathered the test results into one single graph (see Figure 5.8) that contains the times that one peer (Peer A) takes to retrieve the search results locally, the time that one peer takes to get all the results from Peer A if it is in the same LAN, or if they communicate through the relay service (WAN).

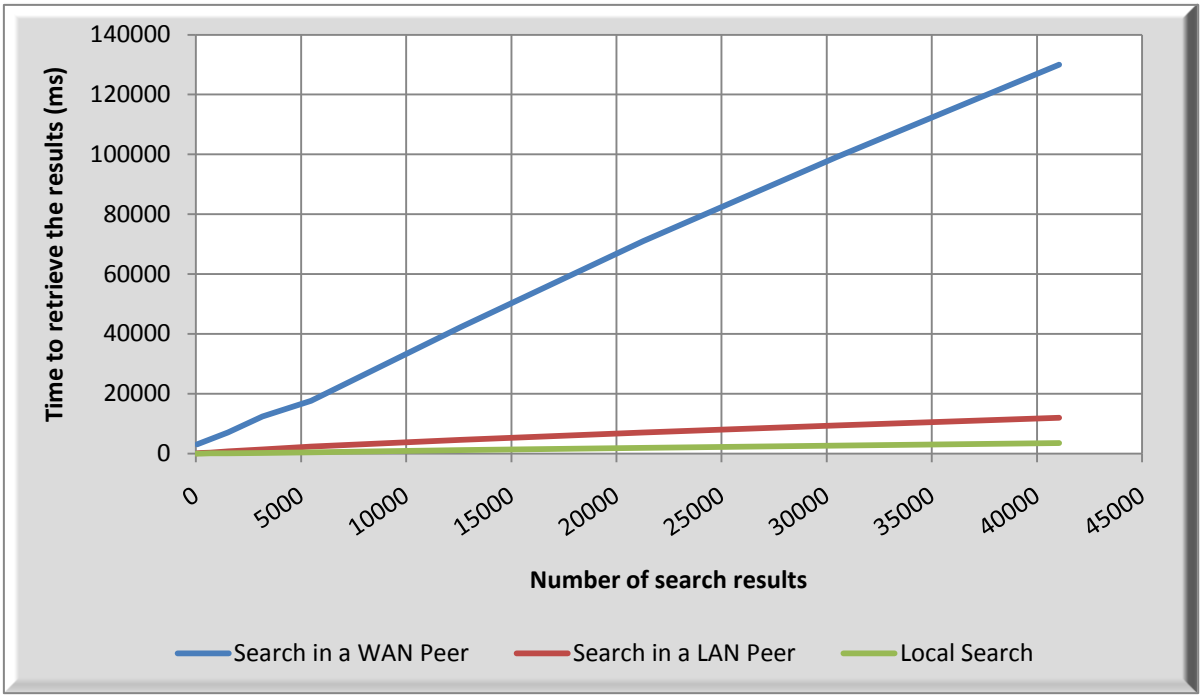

Figure 5.8 - Comparison graph of the search results retrieval times of WAN, LAN and local index

As the graph shows, the search in WAN is the slowest kind of search, in second is the search in LAN and the fastest is the local search.

A noteworthy result is that we queried locally the indexer with the same queries as we queried the network and multiplied the number of images by the number of responder peers, obtaining the number of results that a search must retrieve. With this knowledge and analyzing the search results we could conclude that, as far as the tests went, all the resources available in the network were found.

5.2.2 Multi-Threading File Transfer in WAN

As described in 4.3.2.2, the file transfer was not efficient with single-threaded senders and receivers. Therefore, multi-threaded senders and receivers were implemented.

In order to prove that multi-threaded senders and receivers make the communications faster some tests have been made. We chose a file which size was about 18 megabytes and transferred it, using different number of threads on the receiver and on the sender. The times were measured in the following way:

- Upload time – is the amount of time between the reception of the File Request Message and the moment the sending of the last file fragment is finished.
- Download time – is the amount of time between the moment when the user requests the File download and the moment when the last file fragment was received.

The results are presented in Figure 5.9, where each result is actually an average of ten measures done, once each test was performed ten times.

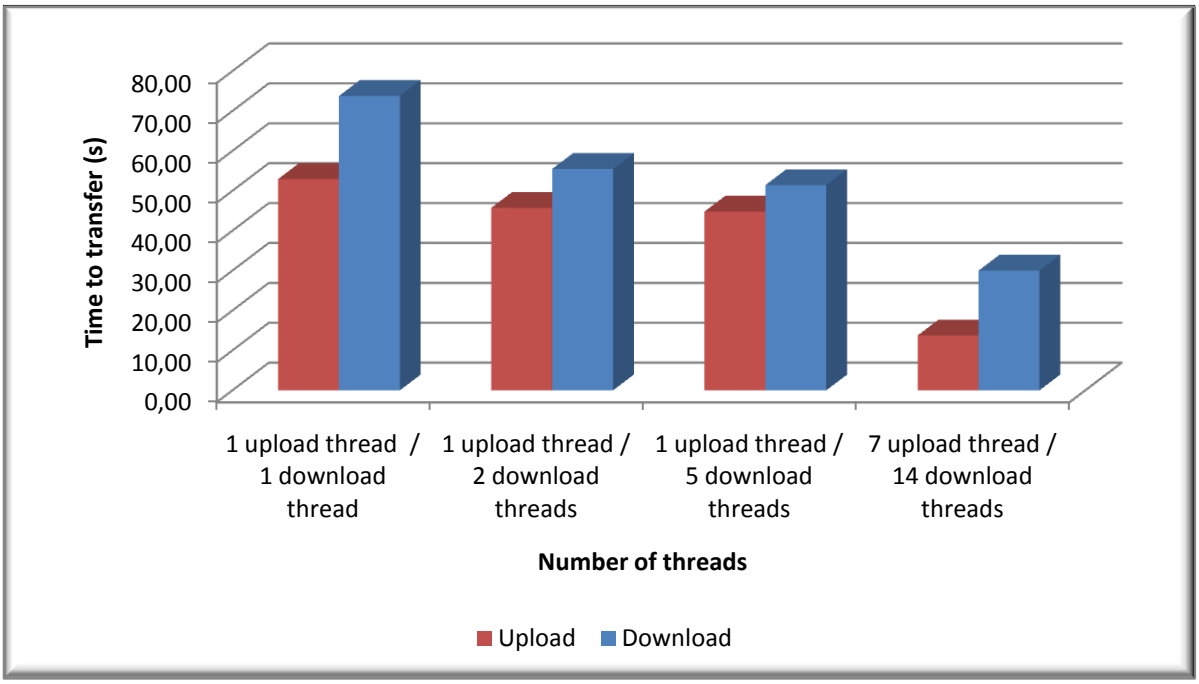

Figure 5.9 - Graph of upload and download times of eighteen megabytes of image data

As the graph shows, it is clearly faster the file transfer with a 7-threaded sender and a 14-threaded receiver than with a 1-threaded sender and a 1-threaded receiver. The need of more threads in the receiver than in the sender is due to the fact that GAE takes more time to perform a query than storing a new entity.

6 Conclusion

Digital medical imaging is a research area that has been growing in importance and proliferation. However, such a growth brings problems of interoperability between systems. To solve these problems, an effort has been made to standardize processes, appearing international standards as DICOM.

During the last years, DICOM standard has been implemented in almost all medical equipments, therefore much medical data has been produced under the DICOM format [7]. All this data is stored in multiple repositories in multiple institutions what brings us the problem of how to share the information between the institutions. Solutions for this problem were implemented such as: email, traditional mail, VPN, among others, but none of them can truly meet the needs of the medical environment.

This thesis proposes the use of Peer-to-Peer technology to provide a sharing platform, so that it can be possible to access medical data when and where it is needed. This technology is widely used, once it is very flexible and can be adapted to the needs.

The Peer-to-Peer architecture suggested and implemented is a JAVA open source project, compatible with multiple platforms, that combines reliable multicast, cloud computing and indexing technologies. On the one hand, with reliable multicast it is possible to easily propagate the queries in an intra-institutional network guaranteeing, at the same time, that all active peers receive the queries. On the other hand, cloud computing is a relatively stable way to serve as a bridge of communications between institutions, solving communications obstacles such as firewalls and routers. Finally, the indexing technology is the responsible for the content-based search feature, providing to users a Google-like search experience.

Therefore, it was achieved a reliable, scalable and resilient solution to share medical imaging data among the medical environment.

In the end, the network proposed was adopted by the project Dicoogle [59] as communication platform. Besides, the thesis work was the object of two scientific papers:

- Carlos Costa, Carlos Ferreira, Luís Bastião, Luís Ribeiro, Augusto Silva and José Luis Oliveira. *Dicoogle - An Open Source Peer-to-Peer PACS.* Submitted to "Journal Digital Imaging".
- Carlos Ferreira, Luís Ribeiro, João Cândido Santos, Carlos Costa and José Luís Oliveira. *A Peer-to-Peer architecture for medical imaging networks.* Submitted to "1st ACM International Health Informatics Symposium".

6.1 Future Work

At this moment, the network implemented provides a stable service and easy of installing solution. However, to achieve the full potential of the Peer-to-Peer architecture proposed, the implementation should be improved in some items.

The implementation of the relay service above the Google App Engine (GAE) brought some problems. The impossibility of sending a notification from GAE to one peer, when there is a new message in the relay, brought some performance issues. Commonly, the messages take longer to reach to its destination with a polling strategy than if they were simply forwarded by the relay as the relay implemented in a server (see 4.3.1). Therefore, in the future, other cloud computing services can be studied, in order to determine which one fits the best with this usage scenario.

Security measures shall be implemented above the network, more critically in WAN. For this matter, security measures were thought. However, some were not implemented as, for instance:

- The authenticity can be improved with a login system on the relay service, using either Google Accounts or a login system created from the scratch and deployed into GAE.
- At the privacy level, something was implemented with a cryptographic algorithm that uses the same key to encrypt and decrypt, it is needed that both sender and receiver have the same key to allow the communications between them. Anyway, this can be improved with the use of HTTPS messages for WAN communications, instead of the current HTTP traffic. Both GAE and HTTP-core are able to receive and send HTTPS messages. Therefore, the change shall be easily implemented.
- For what the integrity is concerned, in WAN the HTTP traffic is exchanged above TCP sessions, at the relay service the messages waiting to be forward are stored in the datastore service that assures the integrity of the data stored. On the other hand, in LAN the traffic is exchanged through JGroups, that guarantees that the messages are always received error-freed and in the right order. Therefore, it is possible to guarantees that the implementation already assures some level of integrity into the messages exchanged.

In a LAN, some routers that blocks multicast traffic may exist, so the current LAN implementation does not support this kind of obstacles. However, JGroups API has a tunneling feature that can be used to surpass the multicast-blocking routers.

7 Bibliography

1. Ishikawa, H., et al., *Three-Dimensional Optical Coherence Tomography (3D-OCT) Image Enhancement with Segmentation-Free Contour Modeling C-Mode.* Investigative Ophthalmology & Visual Science, 2009. **50**(3): p. 1344-1349.
2. Azevedo-Marques, P.M.d. and S.C. Salomão, *PACS: Sistemas de arquivamento e distribuição de imagens*, in *Revista Brasileira de Física Médica*. 2009, Associação Brasileira de Física Médica.
3. Huang, H.K., *PACS and imaging informatics : basic principles and applications*. 2nd ed. 2004, Hoboken, N.J.: Wiley-Liss. li, 649 p., 8 p. of plates.
4. Ribeiro, L.M.S., C. Costa, and J.L. Oliveira, *A Proxy of DICOM services.*
5. Pereira, M.F.N.S.A., *Tecnologia Peer-To-Peer para bibliotecas digitais*, in *Departamento de Electrónica, Telecomunicações e Informática*. 2008, Universidade de Aveiro: Aveiro. p. 101.
6. Zhu, H., *Medical Image Processing*, in *Summer School Program - Introduction to Mathematical Medicine*. 2003, Fields Institute.
7. Costa, C., et al., *Indexing and retrieving DICOM data in disperse and unstructured archives.* Int J Comput Assist Radiol Surg, 2009. **4**(1): p. 71-7.
8. National Electrical Manufacturers Association. and American College of Radiology., *Digital imaging and communications in medicine (DICOM)*. 2009, Washington, D.C.: National Electrical Manufacturers Association.
9. Pianykh, O.S., *Digital imaging and communications in medicine (DICOM) : a practical introduction and survival guide*. 2008, Berlin: Springer. xx, 383 p.
10. Instrumentation, A.f.t.A.o.M. *AAMI - glossary: IT World*. 2009; Available from: http://www.aami.org/resources/glossaries/itworld/Glossary-I.html.
11. National Electrical Manufacturers Association. and American College of Radiology., *Digital imaging and communications in medicine (DICOM) Part 18: Web Access to DICOM Persistent Objects (WADO)*. 2009, Washington, D.C.: National Electrical Manufacturers Association.
12. Mustra, M., K. Delac, and M. Grgic, *Overview of the DICOM Standard.* Proceedings Elmar-2008, Vols 1 and 2, 2008: p. 39-44.
13. Medting. *MEDTING - MedicalTube, medical meeting; exchange video and image*. 2008; Available from: http://medting.com/.
14. Rüdiger Schollmeier, *A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications* in *First International Conference on Peer-to-Peer Computing (P2P'01)*. 2002.
15. Singh, M.P., *Peering at peer-to-peer computing.* Ieee Internet Computing, 2001. **5**(1): p. 4-5.
16. Taylor, I.J., *From P2P to Web services and grids : peers in a client/server world*. Computer communications and networks,. 2005, London: Springer. xix, 275 p.
17. Shirky, C. *What is P2P...and what isn't*. 2000 [cited 2010; Available from: http://openp2p.com/lpt/a//p2p/2000/11/24/shirky1-whatisp2p.html.

18. H.Schulze and K. Mochalski, *The Impact of P2P File Sharing, Voice over IP, Skype, Joost, Instant Messaging, One-Click Hosting and Media Streaming such as YouTube on the Internet.* Tech Report, 2007.
19. [Anon], *SETI@Home hits milestone.* Dr Dobbs Journal, 2002. **27**(8): p. 14-14.
20. Anderson, D.P., et al., *SETI@home - An experiment in public-resource computing.* Communications of the Acm, 2002. **45**(11): p. 56-61.
21. Demorest, P., et al., *Serendipitous detection of radio pulses from evaporating black holes, GRBs and extragalactic supernova using SETI@home.* Astronomy, Cosmology and Fundamental Physics, Proceedings, 2003: p. 436-437.
22. Douglas, K.A., et al., *Spinoff successes of the SETI@home project.* Astrobiology, 2007. **7**(3): p. 510-510.
23. Korpela, E., et al., *SETI@home - Massively distributed computing for SETI.* Computing in Science & Engineering, 2001. **3**(1): p. 78-83.
24. Murray, T. *Overall project stats for Seti@Home*. 2007; Available from: http://stats.kwsn.net/project.php?proj=sah.
25. Werthimer, D., et al., *The Berkeley radio and optical SETI program: SETI@home, SERENDIP, and SEVENDIP.* Search for Extraterrestrial Intelligence (Seti) in the Optical Spectrum Iii, 2001. **4273**: p. 104-109.
26. Wehrle, K. and R. Steinmetz, *Peer-to-Peer systems and applications*. 2005: Springer.
27. Karl Aberer, M.H., *An Overview on Peer-to-Peer information systems.* 2002.
28. Fox, G., *Peer-to-peer networks.* Computing in Science & Engineering, 2001. **3**(3): p. 75-77.
29. Li, D.S., X. Nong, and X.C. Lu, *Topology and resource discovery in peer-to-peer overlay networks.* Grid and Cooperative Computing Gcc 2004 Workshops, Proceedings, 2004. **3252**: p. 221-228.
30. Portmann, M., et al., *The cost of peer discovery and searching in the Gnutella peer-to-peer file sharing protocol.* Ninth Ieee International Conference on Networks, Proceedings, 2001: p. 263-268.
31. Ripeanu, M., *Peer-to-peer architecture case study: Gnutella network.* First International Conference on Peer-to-Peer Computing, 2002.
32. *The Gnutella Protocol Specification v0.4*. Available from: http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.
33. Berkes, J.E., *Decentralized Peer-to-Peer Network Architecture: Gnutella and Freenet.* 2003.
34. Galante, J., *Skype Names Former Sony Ericsson Mobile President as Chairman.* Bloomberg, 2010.
35. Baset, S.A. and H.G. Schulzrinne, *An analysis of the Skype peer-to-peer Internet telephony protocol.* 25th Ieee International Conference on Computer Communications, Vols 1-7, Proceedings Ieee Infocom 2006, 2006: p. 2695-2705.
36. Brunner, R., *A performance evaluation of the Kad-protocol*, in *Corporate Communications Department*. 2006, Institut Eurécom.
37. Wang, P., et al., *Attacking the Kad Network*, in *SecureComm 2008*. 2008, ACM: Istanbul, Turkey.
38. Blanquer, I., V. Hernández, and F. Mas, *A P2P Platform for sharing radiological images and dignoses*. 2004.
39. Chang, I.C., et al., *Factors affecting cross-hospital exchange of Electronic Medical Records.* Information & Management, 2009. **46**(2): p. 109-115.
40. Prasad, A., *Chaos! Thy name is Internet*, in *DRTC Workshop on Information Management*. 1999.
41. *Apache Lucene*. Available from: http://lucene.apache.org/java/docs/index.html.
42. Ban, B. *JGroups - The JGroups Project*. 2010; Available from: http://www.jgroups.org.
43. *jxta: JXTA™ Community Projects*. Available from: https://jxta.dev.java.net/.

44. *Xeerkat - Project Hosting on Google Code*. 2010; Available from: http://code.google.com/p/xeerkat/.
45. Foundation, X.S. *XMPP Standards Foundation*. 2010; Available from: http://xmpp.org/.
46. *jxta-jxse: View announcement*. Available from: https://jxta-jxse.dev.java.net/servlets/NewsItemView?newsItemID=5464.
47. Ban, B., *Reliable Multicasting with the JGroups Toolkit.* 2009.
48. Ribeiro, L., *Peer-to-Peer DICOM Storage Facility*. 2010.
49. Abdellatif, T., E. Cecchet, and R. Lachaize, *JGroups evaluation in J2EE cluster environments*. p. 24.
50. Ban, B. *JGroups - Success Stories - Projects Using JGroups*. 2010; Available from: http://www.jgroups.org/success.html.
51. Steve Bennett, M.B., Robert Covington, *Oracle White Paper in Enterprise Architecture - Architectural Strategies for Cloud Computing.* 2009.
52. LLC, A.W.S. *Amazon Elastic Compute Cloud (Amazon EC2)*. 2010; Available from: http://aws.amazon.com/ec2/.
53. Rackspace US, I. *Cloud Computing, Cloud Hosting & Online Storage by Racksapce Hosting, Moso is now the Rackspace Cloud.* 2010; Available from: http://www.rackspacecloud.com/.
54. Google. *What is Google App Engine?* 2010 [cited 2010; Available from: http://code.google.com/intl/pt-PT/appengine/docs/whatisgoogleappengine.html.
55. Microsoft. *Windows Azure Platform | Cloud Computing | Online services | Data Storage*. 2010; Available from: http://www.microsoft.com/windowsazure/.
56. Zahariev, A., *Google App Engine.* 2009.
57. Rimal, B.P., E. Choi, and I. Lumb, *A Taxonomy and Survey of Cloud Computing Systems*, in *Fifth International Joint Conference on INC, IMS and IDC*. 2009.
58. Irani, R.K., *Google App Engine Java Experiments*, R.K. Irani and J. Bâton, Editors. 2010.
59. IEETA. *Dicoogle :: Home*. 2010; Available from: http://www.dicoogle.com/.

Appendix A **Libraries**

During the implementation of the P2P network for medical imaging, there were needed some libraries to deal with particular tasks, such as:

- JGroups: Open source toolkit, responsible for the communication in LAN of this project. This library allows each peer to send and receive messages, to know about all peers connected and it guarantees that all sent messages are received correctly and orderly.
- Dcm4che: Open source clinical image and object management. This library is used for the handling of the DICOM files (the resources shared in the network developed).
- Dom4j: Open source library for working with XML. This library is the responsible for the construction and handling of the XML messages (Query Request, Query Response and File Request).
- Lucene: Open source project, responsible for the search engine. In the implementation of the P2P network, this library is the responsible for the index of DICOM files and local search.
- HttpCore: Open source library responsible for the construction, sending and receiving of HTTP traffic. It is the library responsible for the communication to the WAN.