CRANFIELD UNIVERSITY

Felipe de Souza Schmitt

Data Sharing on P2P Mobile Networks

SCHOOL OF ENGINEERING

Computational & Software Techniques in Engineering

MSc

Academic Year: 2012 - 2013

Supervisor: Dr. Mark Stillwell

August 2013

CRANFIELD UNIVERSITY

SCHOOL OF ENGINEERING

Computational & Software Techniques in Engineering

MSc

Academic Year: 2012 - 2013

Felipe de Souza Schmitt

Data Sharing on P2P Mobile Networks

Supervisor: Dr. Mark Stillwell

August 2013

This thesis is submitted in partial fulfilment of the requirements for the degree of Master of Science

"Stay committed to your decisions, but stay flexible in your approach."

— Anthony Robbins

Abstract

This research project aims to bring of the concepts of social networking and peer-to-peer networking to the mobile devices. This is successfully achieved through the implementation of an Android service to share information across a network with other peers. By taking advantage of the service discovery capabilities of the mobile devices it allows to keep track of other peers on the network. The implementation enabled a method to keep track of all the application-related data existing on the network in other devices and request the information whenever necessary without any user perception of the actual data sharing process. As the results show, this method is fairly quick and have a low impact on the device's energy consumption.

The result of this research project can be carried on to implement other features regarding social data sharing or to be used as a comparison with other future implementations using different technologies.

Acknowledgements

First of all I would like to thank my parents and brother for giving me support throughout my life, for making me sure that they will always be there for me wherever I am and that our bond is unbreakable. I would not be where I am today without their support.

To all my friends who supported me in times of need and I will always receive them with my arms wide open.

I would also like to thank my supervisor for allowing me to proceed with my own research topic, for giving me great input on all ideas and for brainstorming with me about which approaches to follow whenever a substantial problem appeared.

Contents

List of Figures

List of Tables

Abbreviations

API	**A**pplication **P**rogramming **I**nterface
CRUD	**C**reate **R**ead **U**pdate **D**elete
DB	**D**ata**B**ase
GUI	**G**raphical **U**ser **I**nterface
HP	**H**igh **P**ower
HTML	**H**yperlink **T**ext **M**arkup **L**anguage
HTTP	**H**ypertext **T**ransfer (or **T**ransport) **P**rotocol
IDE	**I**ntegrated **D**evelopment **E**nvironment
IP	**I**nternet **P**rotocol
JSON	**J**ava**S**cript **O**bject **N**otation
LP	**L**ow **P**ower
MVC	**M**odel-**V**iew-**C**ontroller
OS	**O**perating **S**ystem
P2P	**P**eer-**to**-**P**eer
PL	**P**rogramming **L**anguage
REST	**RE**presentational **S**tate **T**ransfer
UI	**U**ser **I**nterface
URI	**U**niform **Resource** **I**dentifier
UX	**U**ser e**X**perience
XML	e**X**tensible **M**arkup **L**anguage
WS	**W**eb **S**ervice

Chapter 1

Introduction

The main goal of this research project is to develop and analyse a framework that combines the following two technological trends and enables the creation of a peer-to-peer (P2P) network without a central infrastructure to allow data sharing among a certain number of mobile devices located within a near location. It will be taken into account problems regarding decentralised searches, resource discovery and indexing, data analysis and other drawbacks that are inherent to a peer-to-peer network in a mobile environment.

The technological evolution has made possible for information to be accessible for millions of people around the world and as it keeps evolving, the human need to acquire new information at any time, increases. The idea of social media have been introduced to society and its acceptance have been increasing rapidly[1]. This reflects the human natural instinct to live in a society and share information among each other. As demonstrated by *The Social Media Report*[1] people have become more passionate about accessing the social media and share information. Moreover they have been using their mobile phones to do so.

Smartphones have started to become more common nowadays, having the number of sales been increasing around the world, as well as their penetration on the market in such a way that over 722 million smartphones were shipped to vendors

in 2012[2]. These devices are used daily to access all kind of social media networks as well as to communicate and share information with other people[1].

1.1 Motivation

This research topic approaches several technologies that have been improving and growing its acceptance, leading to new opportunities and applications by using mobile devices through peer-to-peer (P2P) networks. This technology can be applied to various areas of research and development, therefore becoming a really interesting topic to research. There have been some improvements regarding the hardware provided on mobile devices that have allowed to the development community the possibility to apply existing technologies and architectures on the mobile devices that would not be viable otherwise.

Some of the technologies used are state-of-the-art, as a result some of the features available are still under development and there are still some things to be improved and fixed. On one hand this makes research on this area very important and contribute to a better continuous development, but on the other hand it means that there are some features that do not perform as expected, which can become a challenge for any kind of research in this area.

1.2 Goals

The research project has as a primary goal to analyse the possibility to create an Android[3] information sharing peer-to-peer framework and develop an application to share a specific type of content between several Android devices through a peer-to-peer (P2P) network.

In order to achieve this main goal, the following steps are required to take with the purpose of having a thorough analysis and a steady development process:

- Create an overview of the software architecture;
- Analyse the best techniques to search and retrieve information on a mobile P2P environment;
- Explore the Android's Wi-Fi, Wi-Fi Direct and Bluetooth capability to share information across the network;
- Build a Web Service to provide in a centralised way the information that will be distributed through the P2P network;
- Develop an Android application that takes advantages of its network capabilities and analyse its behaviour in a P2P network to share information.
- Allow Android devices to publish on the network the services it provides;
- Allow Android devices to find other services that are being provided across the network;
- Keep synchronised the information on the devices with the same desired information that are connected to the same network.

1.3 Report Structure

This report have been separated into several chapters, each one of them approaches different concerns that are directly and indirectly related to the research project.

The chapter 2, regarding the literature review describes and analyses research projects that are somehow related to the work done for this project.

Afterwards, on the technology section, the technologies that were used in order to develop this project are described and shown its relation to the project and how they contribute to achieve the goals previously set.

The chapter 3, concerning the implementation describes the implementation specifications and after in chapter 4 dedicated to evaluation that shows the experimental procedures, the results obtained and then analyse the feasibility of such functionality to be used regularly by any smartphone owner.

The chapter 5, discusses some decisions made during the implementation of this project as well as the impact on the results.

At last but not least the chapter 6 regarding the conclusion will give a brief conclusion on the practicability of the technology and some future features that can be extended to other applications.

Chapter 2

Literature Review

2.1 Contextualization and Background

The age of mobile networks is upon us. The number of people with wireless capable devices have grown exponentially on the last years[4, 5], giving the opportunity to innovate and port methodologies and architectures, that before could only be done with fixed computers with a steady network connection, to mobile devices which have a quite good computational power, storage capacity and suitable hardware.

Now, more than ever people are getting more and more social over the network and accepting this new way for communicating with each other, as well as embracing the opportunities it gives to get to know new people[6].

One type of social networking that has been around for some time now are the peer-to-peer networks, where several users spread over a determined area share with each others resources (e.g. media files, computational resources, sensors feedback).

When joined the smartphone's new capabilities with the growing trend to create a social network among friends and even unknowns to share and discover new information, the result can only be interesting and worth researching at a greater depth.

This research project plans to implement and analyse the usage of peer-to-peer mobile network on the smartphone platform Android[3] to share information among users. By creating peer-to-peer mobile networks it is required to take into consideration some concerns such as peer discovery, network stability, network failure recovery, distributed data management, which are also described in this dissertation.

2.2 Previous Work

Due to the mobile different capabilities to discover neighbours across a certain range (Bluetooth, Wi-Fi, Wi-Fi Direct and NFC) there have been already some research regarding the discovery of neighbours. Although inherent to this research there has always been a big concern regarding power management as these radios that are used for searching have different ranges but also different power consumption rates. The task of discovering peers, as shown on previous research[7], can be improved through an hybrid discovery process using high-power (HP) and low-power (LP) radios to find neighbours. It is improved by combining both an having a lower cost to find neighbours that are near the device.

The discovery of new neighbours in this paper only has as a goal to find new devices, not having any knowledge on which services each device provides, which low-power radios do not provide, meaning that in this case it would require the device to connect to another device to acknowledge if the service needed is being executed on the other device, increasing the cost of discovering of new peers that provide the same service.

As this process have a high energy cost while searching for peers on a HP radio, there have been some research on this area where it has been proposed by Bakht[8] the usage of an asynchronous periodic slot-based discovery scheme named Searchlight which uses active and sleep slots to search for new peers instead of being continuously searching for them. The scheme proposed is said to provide better

bounds on discovery latency than any existing protocol when nodes have similar energy requirements[8].

Although this scheme has been proven to improve the latency of peer discovery, it is not implemented on the platform that is being used in this research for peer discovery and energy management since Android has its own scheme. Some of the constrains of this existing scheme are going to be detailed across this thesis.

Another research area that is of extreme interest to this research project is service discovery across a network. Over the last years there have been some research on this area regarding service broadcasting and discovery. These researches led to a methodology called ZeroConf that stands for zero configuration networking. ZeroConf's vision is to search and use services and not about its configuration[9]. This methodology provides an easy framework to find and analyse all the services available[9].

Through research and development there has been created an implementation of a DNS-based service discovery (DNS-SD) called Bonjour provided by Apple Inc.[10, 11] to accelerate the process of service discovery. The Bonjour implementation uses a multicast DNS[12] (mDNS) which is a protocol similar to DNS but is implemented over a multicast protocol.

Using this idea and concept, Arthur van Hoff started in 2002[13] a pure java implementation of multicast DNS called JmDNS that implements most of the Bonjour functionalities while still remaining fully compatible with it.

Since Android uses Java as the programming language for development and the library works with the Dalvik Virtual Machine, which is the process virtual machine for Android instead of the normal JVM, this library can be used in this research project in order to achieve its goals.

2.3 Research Novelty

The main objective of this thesis is to find out the main constrains and how feasible it is to create a peer-to-peer network on a mobile environment in order to share data that would be beneficial to all the users involved.

Since the mobile industry have been improving and innovating in this area, it is needed to see the current status of this technologies and how far can we push them to provide a better service to the users.

This thesis will take into account the current standards that have been implemented on the Android operating system (OS) for peer discovery using its default API for wireless network and peer discovery.

2.4 Technologies

2.4.1 Peer-to-Peer Networks

In this section there will be an overview about how a peer-to-peer network works, as well as the concerns that need to be taken into account on a decentralised networks structure. It will also address the main topics regarding peer-to-peer networks such as the performance of several search methods, indexing and discovering new resources within the network, data integrity, network security and distributed data management.

A peer-to-peer (P2P) network, where the "peers" are computers connected to each other through an access point (internet, intranet or others), allowing shared access to some resources, such as data, sensors, processors and do not require a central server[14]. This meaning that if you retrieve any "peer" from a $N+3$ peers network, the service must remain active.

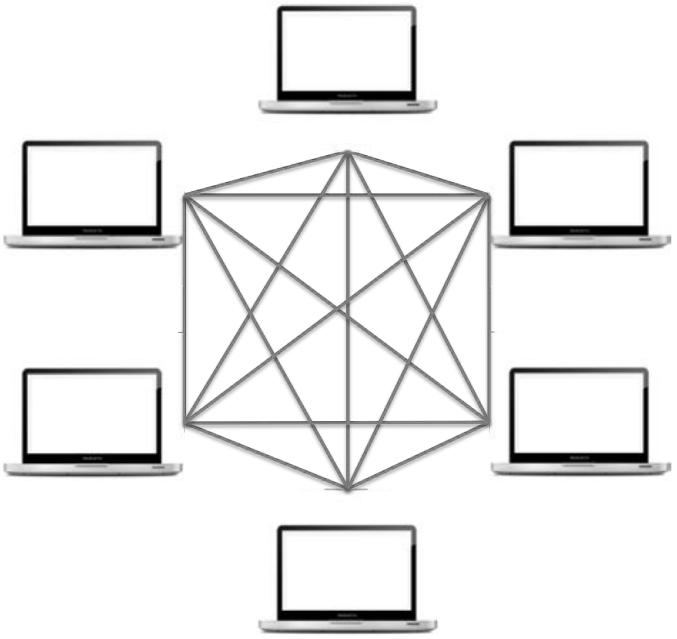

FIGURE 2.1: Representation of a peer-to-peer network.

Regarding a network architecture like the one presented above on the figure 2.1 there are several concerns that need to be taken into account. For example the ones described in the next sections.

2.4.1.1 Search

In order to have an efficient search mechanism it must be possible to locate the data required in an efficient way provided its resources. Although in a peer-to-peer network the individuality of each peer turns the resources unstable and unreliable.

For these reasons when implementing a search mechanism for key lookups on a peer-to-peer network, the developer must be aware of the autonomous peers and the probability of redundancy within all the resources available.

2.4.1.2 Security

On peer-to-peer networks security is very important and while implementing such architecture it is necessary to guarantee access control, data analysis and anonymity.

Any other peer across the network should not have access to any other resource than the ones specific to the architecture, allowing only access to the resources needed for such interaction.

In terms of data analysis, the data that is being transferred across the network must always be analysed in order to check if the data being received is the one expected.

The anonymity provides the peers to only disclosure information about their identity that is necessary in order to proceed with the interactions happening across the peer-to-peer network.

2.4.2 Mobile Devices

On the last few years the number of smartphones sales have been increasing very rapidly[2] and their computational specifications have been improving in a way that it is possible to obtain a mobile phone that has more computational power then a computer a few years ago. This opens up new horizons in the field of mobile computing and networking, which still is a broad unexplored research area.

2.4.2.1 Android Platform

The Android platform is based on the Linux operating system but applied to mobile devices, where it uses its own dedicated libraries, APIs and frameworks. Android's development is led by Google that uses Dalvik virtual machine(VM), which is a fast just-in-time compiled and optimised bytecode VM. Being the code developed in Java, its bytecode is translated into Dalvik bytecode in order to be possible to run it on the Dalvik VM[15].

The development of the idea proposed on this research project will be put into practice by using the operating system developed by Google[16]. This operating system was chosen because it has a strong development support given by a reliable company such as Google and besides providing several technologies that will

help the implementation of a peer-to-peer network it is also the world's leading smartphone platform with around 70% of the market share[2]. Therefore in the eventuality that this research project reaches the deployment stage it would reach a broaden range of users. By using the Android platform it is also possible to develop a cost free prototype (unlike some of its competitors) where it could be done a more extensive testing by deploying it to a selected group of testers.

2.4.2.2 Android SDK

The Android Software Development Kit (SDK) provides the API libraries and developer tools that allows the user to build, test and debug applications for the Android Platform[17].

The API libraries available on the Android SDK allow the programmer to access some of the hardware capabilities of the mobile devices, in order to build the prototype it will be used some of the main developer APIs and specifically it will be analysed the possibility to use some that would allow access to the network hardware such as Wi-Fi and Bluetooth.

2.4.2.3 Ad-Hoc Network

In order to create a peer-to-peer network it is required that the devices are able to connect with each other without having a centralised structure, therefore using ad-hoc connections between the devices.

With the purpose of creating ad-hoc connections between the devices to share information among them, in this research project it will be analysed two different technologies that allows that. These technologies are Bluetooth and Wi-Fi, each of them have its constrains, so it will be selected the one that best fits this project.

2.4.2.4 Bluetooth

This is a wireless technology for short-range communications that provides a simple, secure and a low cost solution to realise communications between two devices. This type of connections is called pairing, which allows the two devices to securely send data to each other[18].

Bluetooth technology was originally invented in 1994 by engineers at Ericsson. In 1998, a group of companies agreed to work together using Bluetooth technology as a way to connect their products. These companies formed the Bluetooth Special Interest Group (SIG). This organisation is devoted so maintain the technology and control the technical aspects of it[19].

2.4.2.5 Wi-Fi Direct

Wi-Fi Direct is a wireless technology that enables devices with Wi-Fi capability to connect with each other without the need of an access point or internet[20]. This technology takes all the advantages of an Wi-Fi connection, such as medium-range communications and large bandwidth connections that allows the devices to send data using the Wi-Fi standards at a fast speed[20].

The devices that support Wi-Fi Direct can create a connection between them anywhere, as long as both devices are within Wi-Fi range. When this happens the device will signal other devices within the area range and this way view the available devices and afterwards send a request to connect to one of them, or receive an invitation from any other device within range.

After two devices are connected they form a Wi-Fi Direct Group and afterwards any device can ask to join that group, forming this way a peer-to-peer network with several devices[20].

2.4.3 Web Services

In order to control the data that is going to be shared within the testing framework application it will be created a simple RESTful Web Service (WS) that will provide with daily news retrieved from other sources (e.g. RSS feeds). This information provided by the WS is going to be used as reliable and secure data that will be transferred among the devices through a peer-to-peer network without the access to the internet after at least one of them have downloaded the information. The Web Service will be implemented through the programming language Ruby using the framework Rails. The information will be transferred using the open standard JSON since it is a lightweight data-interchange format when compared to XML [21].

2.4.3.1 RESTful Web Service

REST defines a set of architectural principles by which you can design Web services that focus on a system's resources, including how resource states are addressed and transferred over HTTP[22].

This architectural style follows four basic design principles [22]:

- Use HTTP methods;
- Stateless;
- Use URIs as structured directories;
- Transfer messages using XML or JSON.

A RESTful service uses HTTP methods such as GET, POST, PUT and DELETE to accomplish CRUD (Create, Read, Update, Delete) operations on a data storage. All the requests sent through HTTP on a RESTful service are completely stateless, as the server should not maintain any session from the client, allowing it to scale and meet high demands [22]. In order to access different resources in a RESTful

service there are specific and intuitive URIs that indicate the resources that are being accessed. In this kind of service there is the need to transfer messages with resources representation and they could be either in XML or JSON.

2.4.3.2 Ruby on Rails

2.4.3.3 Ruby

Ruby is a dynamic, open source programming language with a focus on simplicity and productivity. It has an elegant syntax that is natural to read and easy to write[23].

The programming language (PL) Ruby has several strong arguments on its side that makes it a perfect PL to use for prototyping[24] (and further related to Web Development):

- Simple programming language;
- Easily scalable;
- Small learning curve;
- Ideally suited to rapid application development.

As well as Python, it is a programming language that is easy to learn and allows fast prototyping and helps you improve the productivity by maintaining the simplicity.

2.4.3.4 Rails

Rails is an open source web application framework developed for the Ruby programming language, which denominates Ruby on Rails. This framework emphasizes the use of well-known software engineering patterns, such as model-view-controller, do not repeat yourself, among others.

The Ruby on Rails framework allows the developer to easily create a Web Service, maintain a database record and without difficulty manage the controller between the requests and the retrieval of information. As well as Ruby, the framework have as a positive aspect the easy scalability of its applications[25].

2.4.3.5 JSON

The lightweight data-interchange format JSON (JavaScript Object Notation) is based on a subset of the JavaScript Programming Language[21], which is easy to read and write for humans and easy as well to parse and generate through code.

The JSON has almost no learning curve for developers that are already familiar with JavaScript or other programming language with similar support for rich literal notation such as Python and Ruby [26].

The structure of a JSON Object can be of two types:

- A collection of name-value pairs;
- An ordered list of values.

The JSON Object is a unordered set of name-value pair that starts with a left brace ({) and ends with a right brace (}). Within the braces the name-value pairs are separated by a coma (,). Having each name followed by a colon (:) and then the value related to that name. On the other hand the JSON Object Array is set by starting with a left square bracket ([), followed by a list of values (or JSON Objects) split by a coma (,), ending with a right square bracket(]) [21].

An example:

```
{
    "title" : "NASA Award Grant To Cranfield University",
    "date" : "18th Mar 2013",
    "keywords" : [
        "nasa",
        "cranfield",
        "uk",
        "award"
    ]
}
```

The example above shows the structure of a JSON Object and demonstrates how easy it is to read and write using the JSON standards.

Chapter 3

Methods

The main goal of this research is to evaluate the feasibility and constrains that currently exist on sharing social data through peer-to-peer networks taking advantage of the mobility that mobile devices allow. In order to achieve this it is necessary to develop an application that would take advantage of these characteristics that could reach the most people as possible. Due to this it was chosen to implement it to the Android OS, due to its current enormous market share of all the mobile devices sold around the world [27].

This chapter will be divided into the architectural design of the project and how each of the two main parts (Server and Client) were implemented. Furthermore it explains how such experiment could be replicated and also how does the peer discovery and the data management are implemented in order to have an easy, non-interactive and smooth transition of information between devices.

3.1 Development Techniques

With the purpose of achieving the goals initially set for this project it was created a project plan with several milestones within several iterations of the process called sprints. Each sprint had its own features and goals, which could provide an easy way to check if the project was on schedule or not. These sprints were useful also to keep track and report to my supervisor of my progress and difficulties found along the way that are also described ahead in this report.

In addition this project was developed on the operating system Mac OS X, using the text editor Sublime 2 in order to write the Ruby code for the web service and the new Android development environment provided by Google called Android Studio. This integrated development environment (IDE) is an useful tool to develop applications for Android as it allows the developer to deploy the application to a physical device and run it directly on an Android smartphone. It also allows the developer to have realtime access to the logs being produced on runtime with debug information. This helps to debug the application and find the problems that occur on runtime, which can be difficult to figure it out otherwise, if we take in to consideration the complex and unpredictable environment that mobile and network development normally are.

Also in order to have an attractive design that would engage the end user and at the same time would be intuitive and easy to use, it was used the image manipulation software Adobe Photoshop CS6 for mac. This allowed the creation of a user interface (UI) that could provide a good user experience (UX) and separate completely the end user from all the background procedures that allow the social information sharing to happen.

Furthermore in the development process of this project it was used a git repository to register all the changes made and trace all the work done, from design reports to source code for both parts. In order to maintain the privacy of this project it was used a private server to store all the information regarding the development

process. By using this technology it allows a more agile and flexible approach to the development process and keep track of the progress done across the project.

3.2 Architectural Design

This section will demonstrate how each one of the main components are designed and show its elements and their expected behaviour that will have an impact in achieving the goals previously set.

The first step, when creating a distributed system, should be to analyse all the different main components that such system include and study their interaction with each other.

Another important step is for each one of these main components, explore its architecture and the elements that are needed in order to achieve its desire functionality in the global environment of the distributed system.

The architectural design of this distributed system can be divided into two main different parts, the server-side (web service) and a client-side (mobile device application) in this section each one of these parts are explained in depth.

3.2.1 Server - Web Service

One of the main components of this general architecture is the web service that provides an easy access to collected and stored information. This component will be a very important part of the system as it provides all the information that is shared within this project environment.

The main reason for having this component on this environment is to be able to control and analyse all the data that is in the system which is transferred to mobile devices and then shared among them.

The service being provided allows that all the information is gathered from several sources and afterward they can be formatted into a single standard, keeping all the information formatted the same way so they can be sent to mobile devices as if they were all from the same source .

This part of the architectural design of the web service can be described by the following figure:

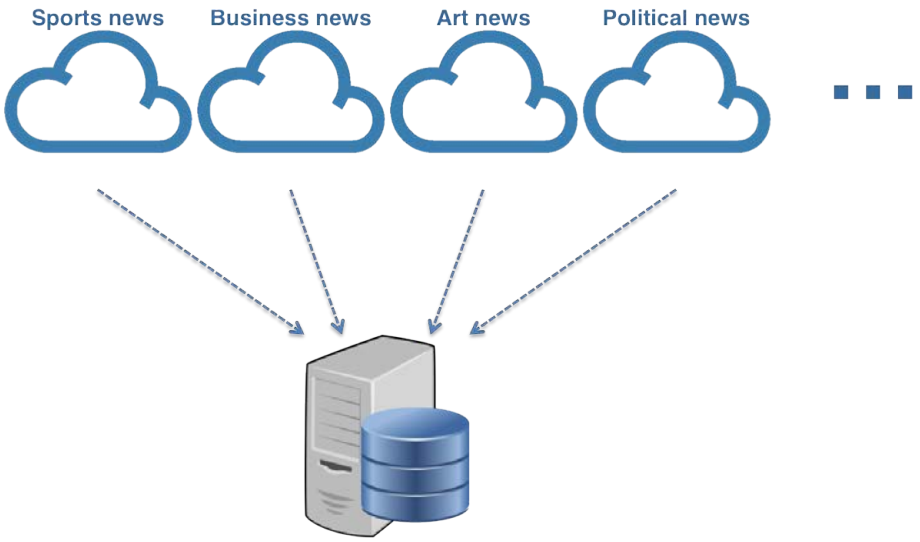

FIGURE 3.1: Feed collection to the web service.

The figure 3.1 shows that the server, which has its own database (DB), fetches the news from several sources periodically and afterward this information is parsed, so it can be stored, categorised and inserted them into the its corresponding table on the database.

In order to keep a clean and separated categorisation of the news collected from the several feeds, each category is considered as separate modules. This consideration enables the addition of future source feeds for specific categories and handle them to integrate the database and merge with all the previous source feeds.

Since each category is considered a different module, the database schema for this project contain a table for each category which allows the implementation of specific attributes for some categories in the future, such as tags, teams involved in a sports news, economical information drawn from data mining of each news and others.

This technique can provide a better metadata of the news for a specific category which can change the way the news are presented to the users afterward.

An example of a table schema being used for the database in the web service can be described by the following table:

Column	**Type**	**Modifiers**
id	integer	not null default nextval([category_seq])
name	varchar	not null
summary	text	
url	varchar	not null
published_at	timestamp	not null
guid	varchar	not null
created_at	timestamp	not null
updated_at	timestamp	not null

TABLE 3.1: Database schema

As it is possible to see by analysing the table 3.1 each entry on the database have some metadata that helps further on with some tasks to provide a better UX, for example: sort the news by date, provide a direct link for the news source's website, check the *guid* to see if the analysing news already exist on the environment database.

How all these steps were done, are explained when the web service implementation is described further in the subsection 3.3.3.

After collecting all the information it is required for the web service to set available all this information to the mobile devices so they can fetch all the news for a specific category instead of for a specific source.

As the web service main purpose is to feed information to the Android application for development and testing environments, it is required to have a REST service to provide an application programming interface (API). This API must provide an interface for the mobile application to access the information stored on the web

service database, this interface should allow access through normal GET request, for example using the following URI:

`http://newsfeeder.herokuapp.com/`<category name>?format=json

The response for this request is in a JSON format, with information of all the news from the category specified in the URI. Being this an array of news where each news it is available as a JSON object with some attributes such as the title of the news, its summary, the publish, update and creation date and a direct URL to the source's website. Following the JSON format an example of a news object from such response containing those information should be as following:

```
{
"created_at":"2013-08-01T23:25:18Z",
"guid":"feedzilla.com:323367660","id":2503,
"name":"Hulu, HBO Working with Google to add content to
Chromecast (Mercury News)",
"published_at":"2013-08-01T23:06:00Z",
"summary":"Hulu, the TV streaming company owned by major media
companies, and Time Warner Inc.'s HBO said they're working with
Google to add their paid-subscription services to the company's
new Chromecast device.",
"updated_at":"2013-08-01T23:25:18Z",
"url":"http://news.feedzilla.com/en_us/stories/technology/323367660"
}
```

Given such representation, it can be interpreted and stored in the mobile device's database so it can be displayed properly to the user afterward. The choice of using JSON as the data representation type is based on the fact that it is simple, light and easy to read either literally or programmatically.

These are the main design concepts behind the architecture of the web service that provide the two main features that will be provide for the distributed system

environment: collect data from several sources and present them to the mobile devices as a single source in a light and simple way.

3.2.2 Client - Mobile Device Application

The other main component in this system is the client side component which is a mobile device application that any user can install on theirs mobile device and have access to all the information either from downloading it through the web service or by peer-to-peer network with other devices using the same application.

On the client side, the Android application have two different modules within its own architecture. The application that runs on the foreground which displays the information and is managed by the user interaction and the service that runs in the background and manages all the discoveries and the synchronisation of information between either other devices or the web service.

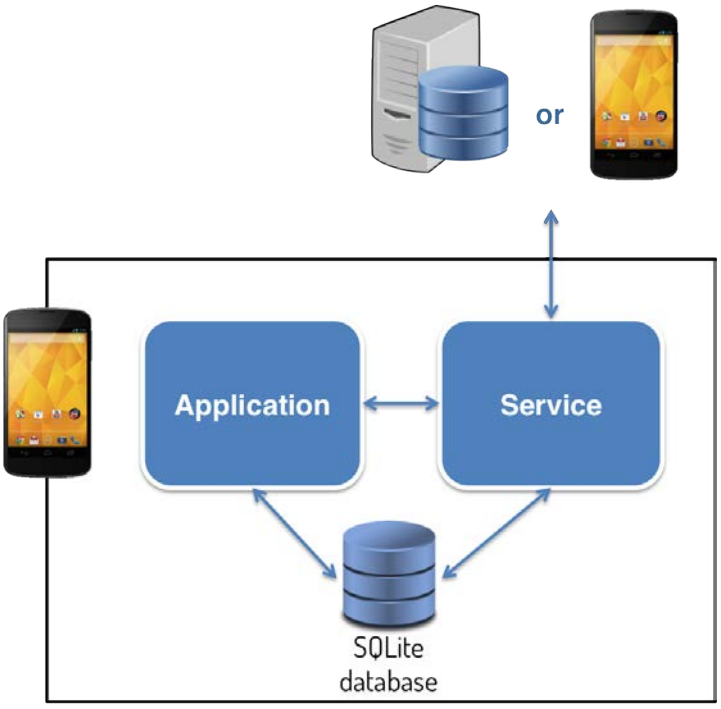

FIGURE 3.2: Client-side architecture.

The application module that is shown in the figure 3.2 is responsible for the user interaction and the display of information to the user when the Android application is open on the foreground. This module can also call some functionalities that the

Service module provide that will update the current information stored on the device by downloading it through the web service if it has an internet connection.

The service module on the other hand is running on the background of the device as a service and is responsible for continuously keep track of the discovered peers, discover new ones and synchronise the required information between them. The Service will download new information from the web service when an internet connection is available as well as it will manage all the requests (sent and received) from other devices within the network and also manages its corresponding data transition (lookup, send and receive).

The application and the service module have direct access to the device *SQLite* database which allows both modules to execute queries to display the news to the user as well as to lookup for information when a request is received, which allows the device to send the information to the request sender if the data is found.

Both of these modules are described further in this chapter, more specific on the section regarding the implementation of each system that will cover every functionality for each module and explain how they all are connected.

3.3 Web Service

For this research project it was required to exist a web service where all the data could be provided as a single source and all with the same structure so that the information could be fetched by the client side (mobile device application). This allows the addition of new sources of information without any effort from the user, which enables the service to expand to new categories and sources without the user having to interact with the system.

3.3.1 Framework

In order to do so, it was chosen to implement the web service using the framework Rails[25] which runs on Ruby programming language, due to its flexible and scalable structure that can be easily changed to support new categories of information and from different sources.

This framework also allows the creation of a *MySQL*, *PostgreSQL* or *SQLite* database[28] to store all the information. This shows the flexibility that this framework have and also have the possibility to change the database technology between development, production and testing environments. This flexibility can be seen as the project was developed using *SQLite* for *development* and *testing* and in order to be compatible with the cloud service that was used to deploy the web service (see subsection 3.3.2), it was changed to *PostgreSQL* for the *production* environment.

One of the greatest advantages of using Rails is that it also provides a structure to build a REST service by allowing several types of response to the same GET request only needing one extra parameter to determine the response type which could be for example HTML, JSON or XML (see section 3.3.3.4).

3.3.2 Cloud Service

With the purpose of having a service that would be available at all time and could handle several requests from a large number of users without having any constrain the web service was deployed to a cloud service called Heroku[29]. It provides a free service when using one virtual machine to handle the requests, this number can be increased as required by altering a parameter but it has some extra costs.

In order to provide this service for free to several developers it has some constrains, for example there are two main server cycles, active and sleep. When a service does not have a request for a certain period of time it enters on a sleep cycle where there is a front service provided by Heroku that will wake up the service when a new request is received. This means that after the service has been inactive for a while it will take longer to respond to the first request made after that inactive period.

The constrain stated above was solved by adding to the service a plug-in called New Relic[30] that provides an in-depth analysis of a service by sending a ping to the service from time to time and check the response time and availability. Taking advantage of this plug-in, it was set to verify the performance of the service every half an hour, allowing the service to be in the active cycle always, therefore having a better response time even when there are no requests in a long period of time.

Another feature this cloud service provides is a git repository where all the changes can be tracked and as soon as a *push* (upload of the changes made) is done, it automatically restart the server with the new changes. This provides an easy way to deploy all the new functionalities, after they have been tested locally, and have a realtime update on the information passed by to the mobile application.

3.3.3 Implementation

This section will approach several topics regarding the web service implementation which will describe how each feature was implemented and its role in the system.

Regarding the web service, the components explained in this section are the database where all the information is stored, the service that keeps the information up to date and the application programming interface that uses REST to distribute the information to the mobile devices through requests.

3.3.3.1 Database schemas

As stated earlier, with the purpose of allowing the customisation of each set of information to each news category, the database used to storage all the information gathered through all the feeds have a different table for each category.

By using the cloud service Heroku (see section 3.3.2) it was required that the database for the production environment to be a *PostgreSQL* database and Ruby on Rails allows us to do this by declaring on the Gemfile (configuration file) which "gems" (standard Ruby libraries) to use. For example, for this project it was used an *SQLite* database for development and testing and a *PostgreSQL* database for production. This was set by inserting the following declaration to the Gemfile:

```
group :development, :test do
  gem 'sqlite3'
end

group :production do
  gem 'pg'
end
```

As we can see the declaration can be personalised to several different environments for development, testing and production which allows a great flexibility for a distributed system.

In Ruby on Rails the database can be created through the usage of table schemas written into a file and then run the command:

```
rake db:migrate
```

This command checks for changes on the database schemas and execute them to change the database structure.

In order to create a table with the necessary attributes it should have enough metadata and information regarding the news so it can be presented the best way possible to the user.

It has been decided that to analyse the possibility of using peer-to-peer networking capabilities for social interactions among mobile devices the following attributes were enough: name, summary, url, published_at and guid.

With the purpose of creating a table with those specific attributes, the following schema allowed that:

```
class CreateTechnologies < ActiveRecord::Migration
  def change
    create_table :technologies do |t|
      t.string :name
      t.text :summary
      t.string :url
      t.datetime :published_at
      t.string :guid

      t.timestamps
    end
  end
end
```

This schema is an example for the creation of the table regarding the technology category of news. As for this research project all the other category schemas are the same, having the possibility to change these attributes for extra metadata on some categories for a more personalised information on some categories.

3.3.3.2 Database access

The Model-View-Controller (MVC) design pattern that Ruby on Rails integrates allows a clear division between the data structure (Model), the user interaction (View) and the data process between database access through the Model so it can be displayed at the View.

The usage of the data structure (Model) to access the database enables an easy access to the information without having to execute complex queries to obtain the data required.

For example, in order to access all the news of one specific category (e.g. Sports) it is possible to use the Sport object from the Model declaration and use the following method:

```
@sports = Sport.all(:order => '"published_at" DESC')
```

This simple method returns an array of all the news about sport in a descending order of its published date. The Sport data structure (or any other from any category) have some methods implemented that allow a quick and easy access to the information stored at the database.

In this project in order to keep a simple web management tool that enables the control of the data on the server some methods were implemented that can create, read, update and delete (CRUD) information stored in the database. This methods are the same that can be accessed through the REST API.

All these methods implementation are explained at section 3.3.3.5, regarding the REST implementation.

3.3.3.3 Information feed

Since the web service provides news information, it is required that the information should be always up to date. So the information feeds should be regularly checked for new data so it can have access to the newest information as soon as possible.

Ruby has a gem library that allows the schedule of tasks at every x minutes, hours or days called *rufus-scheduler*[31]. The usage of this gem library enabled a way to update the database regularly by checking the feed sources for new information at every 15 minutes.

This can be achieved by creating a *task_scheduler.rb* file at the *initializers* folder and insert all the tasks that are required to be executed within intervals of time.

In order to initialise the *rufus-scheduler* so that a thread can be created to proceed with the tasks that have been allocated, the library must be initialised by inserting the following at the top of the *task_scheduler.rb* file:

```
scheduler = Rufus::Scheduler.start_new
```

For example, in order to update the news for the Art category every 15 minutes from a RSS source using the *rufus-scheduler* library, the following piece of code should be inserted on the *task_scheduler.rb* file:

```
scheduler.every("15m") do
   Art.update_from_feed(
         "http://api.feedzilla.com/v1/categories/13/articles.rss")
   Art.count

   puts 'feed updated.'
end
```

Within this task that occurs every 15 minutes all the different categories are updated using different source feeds by calling the *update_from_feed* method that has

been implemented at the Model data structure of every category. This method is implemented so it can fetch the news through the RSS feeds by using another gem library called *Feedzirra*[32].

```
def self.update_from_feed(feed_url)
    feed = Feedzirra::Feed.fetch_and_parse(feed_url)
    add_entries(feed.entries)
end
```

The method *update_from_feed* uses the *Feedzirra* library which allows an easy technique to parse RSS feeds, which can be afterward inserted into the database through the Model method for adding entries.

```
def self.add_entries(entries)
    entries.each do |entry|
      unless exists? :guid => entry.id
       entry.summary = entry.summary[0, entry.summary.index('<br')]
        create!(
          :name         => entry.title,
          :summary      => entry.summary,
          :url          => entry.url,
          :published_at => entry.published,
          :guid         => entry.id
        )
      end
    end
end
```

The *add_entries* method, which receives as an argument an array of new entries, checks if the *guid* is unique and if it already exists, in order to remove advertising nodes that sometimes are inserted within RSS feeds. Afterward the method strips the string in the summary node so it only contains plain text.

After the feed has been parsed, the information have been treated and it is ready to be stored into the database, the default method for creating a new entry inserts the information on the Model's respective table using the *PostgreSQL* standards.

These are the mains steps from gathering the information from the source feeds, this is done periodically so it can keep the information up date and store them into the database after it have been through the data analysis process.

3.3.3.4 Response type

In order to provide the service that the server enables to other systems it was required the creation of a REST API. Its main purpose is to allow the transfer of representations of resources existing on the server. This way the mobile device application can access the resources that exist on the database of the server to download data to its own database to display to the user afterward and even share among other devices.

As stated previously, JSON is the standard used in this project to represent data structures on the transactions of information between the server and the mobile device. This standard is also used for the transitions between mobile devices for sharing information as it can be seen on section 3.4.2.7.

The usage of this standard is a perfect match with Ruby on Rails as it has a gem library that turns parsing and construct JSON objects trivial, this gem can be used by simply adding on the Gemfile the following line:

```
gem 'json'
```

This library allows the construction of a JSON object from any Ruby object or array, which allows an easy data process in order to format the information and respond to requests using the JSON standards.

For example, in order to format a single news information to a JSON format can be done through the following method:

```
render :json => @single_news
```

The return of this method call is the following JSON object, which represents one single news from the category Art:

```
{
    "guid":"feedzilla.com:324590381",
    "name":"Laura Benanti, Norm Lewis & More to Lead The Public
        Theater's Musical Adaptation of THE TEMPEST at Delacorte
        Theater, 9/6-8 (Broadway world)",
    "published_at":"2013-08-07T16:54:00Z",
    "summary":"The Public Theater announced today that Todd Almond
        Ariel, Laura Benanti Goddess, Carson Elrod Caliban, Jeff Hiller
        Trinculo, Norm Lewis Prospero, and Jacob Ming-Trent Stephano
        have been cast in the free, original musical adaptation of
        THE TEMPEST, part of The Public's groundbreaking new
        initiative for community-based theater, PUBLIC WORKS.
        Directed by Public Works Director Lear deBessonet, with
        music, lyrics and book by Almond, and choreography by Chase Brock,
        THE TEMPEST will run September 6 through September 8 at 800 p.m. at
        the Delacorte Theater in Central Park.",
    "url":"http://news.feedzilla.com/en_us/stories/art/324590381?
        client_source=api&format=rss"
}
```

As it is possible to conclude from this example the JSON structure can be easily readable either by a human or by a parsing method. Since all the request responses are in JSON this method is called on every API request that is sent to the server.

3.3.3.5 REST API

With the purpose of providing access to the mobile device application to representations of some of the web service resources it was necessary the creation of a

REST API that could facilitate the access to the information stored in the web service database.

Since the web service provides a REST API, it implements the acronym CRUD that stands for Create, Read, Update and Delete. The CRUD refers to the main methods that are implemented on any relational database application that can be translated into HTTP methods such as: POST, GET, PUT and DELETE (corresponding to Create, Read, Update and Delete).

The access to the information stored in the web service database must meet the mobile device application's needs, to achieve such access the following methods were implemented:

- Request all news stored of one category;
- Request one single news given its id number;
- Add a recent news (web management);
- Edit an existing news (web management);
- Delete an existing news (web management);
- Request all news stored of one category after a given time.

Because managing the data that is being inserted into the environment of this research project it is important, some of the previous stated methods exist with the goal of providing a better management tool.

The API request to get all the news stored in the web service database of one category can be called by using an HTTP GET request through the following URI:

```
http://newsfeeder.herokuapp.com/<category>?format=<json or html>
```

This request will invoke the corresponding format type method, which will respond with either an HTML page providing all the news of that category or return a JSON array with all the news of that category.

This request is handled by the corresponding category's controller (within the MVC design pattern) that is responsible to invoke the database request and format the response according to the request.

An example of this procedure is the following:

```
# GET /arts
# GET /arts.json
def index
    @arts = Art.all(:order => "published_at desc")

    respond_to do |format|
        format.html # index.html.erb
        format.json { render :json => @arts }
    end
end
```

Comparing with this previous example, the single news request is exactly the same except for the database query where the SELECT request specifies the ID number, returning only that specific news.

In order to provide an easy management interface it was created three management methods that allow to create, edit and delete any news. These methods take advantage of the Model data structure already implemented methods, which can be seen in appendix A.

The method that requests all the news from a specific category after a certain time is the most important request as it is the one that will provide the mobile devices to keep updating their database when they have an internet connection.

This method has a specific handler that takes as an argument an *integer* that corresponds to the id number of the category and a *string* with a time and date that must be previous than all the news of the response.

An example of a HTTP request of news from the category Art that are more recent than the 7th of August is the following:

```
http://newsfeeder.herokuapp.com/request/1/'2013-08-07T17:43:02Z'.json
```

It will return an JSON response with all the news published (that have been stored in the web service) after the 7th of August at 17:43:02.

If the request sends as an argument a category id that does not exist the JSON response provides the list of existing categories and its id numbers. This can be useful if a category is added or removed the mobile device application can be implemented to handle this type of changes and readjust (see section 3.4.2.1).

All of these REST methods implemented enable to the mobile device application a controlled access to the information stored in the web service database, this way providing an unique source of information with data from several different sources as it was the main goal of the web service.

3.4 Mobile Device Application

The mobile application can be considered the main focus of this research project as it will take advantage of the web service information to investigate the aims of this project.

The main topics that are investigated on this implementation are the network capabilities of Android to share information over a peer-to-peer network, to publish and discover services across a network to be aware of surrounding peers and to execute this process without the least user interaction as possible.

3.4.1 Framework

The Android OS have a powerful API that allows developers to use the mobile device's hardware capabilities and truly explore the features the mobile device has to offer and innovate by combining existing technologies to achieve new goals, making it perfect match for this research project.

Some of the API methods available allow the usage of some of the network ability that these mobile devices enable. In this research project there are two main API topics that are within the research interest, those topics are the WiFi and the Network Service Discovery API.

The Network Service Discovery API that is available for development in the Android Developers API allows the usage of some methods that enables a new area of research for service discovery within a network, which will be taken into consideration on the implementation of the mobile application for this research project.

Also the Android Developers API have certain methods that allow the creation of listeners on some of the default network methods such as network discovery, connection status, WiFi Direct methods and the usage of sockets to transfer data over a network.

The usage of the Android framework enabled the implementation of this mobile application that allowed the achievement of the research project goals.

3.4.2 Implementation

The implementation of the mobile application can be divided into two different modules, the application and the service module, as it was previously presented in section 3.2.2.

These two modules have different purposes, although there are some similar actions they both take such for example accessing the mobile device database for data processing (display, store or transfer).

This section firstly describes the database schema and how it is implemented its access, after it describes the connection between the mobile application and the web service to feed information as a single source to the device, then it describes the process of discovering a new service, analysing those discoveries, keeping track of services available and afterward the process of sharing the data available between the devices that require it.

3.4.2.1 Database schemas

The Android framework allows the creation of *SQLite* databases, which makes possible the creation of a database on the mobile device to store all the information that can be used afterward, such as news informations and user's preferences.

The *SQLite* database management classes that Android provide can be used to manage the application own private database, which is how the database for this application was implemented.

Through the usage of the *OpenHelper* java class that extends the *SQLiteOpenHelper* it is possible to create a local private database by using its default constructor with the application context, database name (*p2pnews.db*) and the database version.

The database version allows further upgrades on the database to add or remove new tables and enable the adaptability stated in section 3.3.3.5. This upgrade and adaptability can be implemented in the following method:

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    for(String cat : Categories.db){
        db.execSQL("DROP TABLE IF EXISTS " + cat);
    }
    onCreate(db);
}
```

The override of the *onCreate* method of this class make possible the execution of an SQL schema to create the tables for each category. Each table has the same attributes as the ones that can be found in the web service's response: title, summary, url, published date and guid.

The schema that is executed to create the table of each news category can be seen on the following source code:

```
@Override
public void onCreate(SQLiteDatabase db) {
    for(String cat : Categories.db){
        Log.d(TAG, "Creating table: " + cat);
        db.execSQL("CREATE TABLE " + cat +
            "(title TEXT, " +
            "summary TEXT," +
            "url TEXT," +
            "published TEXT," +
            "guid TEXT PRIMARY KEY" +
            ")");
    }
}
```

The implementation of these methods allow the creation of a local private database that can store the information gathered from the web service regarding several news categories and still keep separated a table for each category in order to permit a future implementation of extra attributes more specific to each category.

3.4.2.2 Database access

Database access is an important component of the mobile application, as it is used by both modules (application and service).

The main methods that the database access class have implemented are:

- Insert information about a single news;
- Retrieve a specific news information;
- Update information about a single news;
- Delete a single news from the database;
- Delete all the news from a category;
- Retrieve all the news from a category;
- Retrieve all news from a specific category after a given time;
- Retrieve the most recent news of a specific category.

The first four methods stated on the previous listing are required to enable the database maintenance (CRUD) for operations like create a new entry, read the information of a single entry, update any entry of the database and delete entries when they are no longer needed or by user's request.

Delete all the news from a category — On the preferences menu of the application there is the option to delete every data that was previously stored in the application database, for that option it was implemented a method that deletes all the content stored in one or every category's table in the database.

```
public void deleteAll(String table_name) {
    this.db.delete(table_name, null, null);
}

public void deleteAll() {
    for(String cat : Categories.db){
        this.db.delete(cat,null,null);
    }
}
```

Retrieve all the news from a category — At the home screen of the application the user can choose one from its favourite category to see all the news that are currently stored in the device as a list, in order to display that list it is required to retrieve from the database all the news data for a single category.

This can be done by executing the following SQL query and then iterate for each entry to get all the column values:

```
Cursor cursor = this.db.query(table_name, null,
                null, null, null, null, "published DESC");
if (cursor.moveToFirst()) {
    do {
        ArrayList<String> info = new ArrayList<String>();
        for(int i = 0; i < cursor.getColumnNames().length; i++){
            info.add(cursor.getString(i));
        }
        list.add(info);
    } while (cursor.moveToNext());
}
```

Retrieve all news from a specific category after a given time — It retrieves from the database all the news from a specific category that were published after a

certain date and time. This feature is extremely helpful further in the data sharing process as it can allow to only send the news that are more recent than the date in the request that was received by the device.

The difference between this last feature with the previous one is that it should only return the entries that have a *published* date more recent than the date passed as argument. This can be achieved by changing the SQL query to the following:

```
this.db.query(table_name, null, "published > ?",
    new String[] {String.valueOf(date)}, null, null, "published DESC");
```

Retrieve the most recent news of a specific category — This method returns the information on the most recent news of a specific category. In order to the mobile device send a request either to the web service or to another mobile device working as a peer on the network it requires the date of the newest news within the category it is searching for, so it can receive only the news that are more recent than the ones it already has in the database.

In order to get the latest news from a category it is required that when executing the SQL query the results should be in a descending order by its published date and only the first result entry is needed. This can be achieved by the following method invocation:

```
this.db.query(table_name, null,
                null, null, null, null, "published DESC", "1");
```

It will only return one entry as stated on the last argument, afterward for metadata purposes the date of that entry can be verified by analysing the corresponding entry's column.

All these methods are implemented in order to facilitate the mobile application's access to its own database and provide an easier way to acquire the information needed to display, request and send information between the mobile devices or even the web service (request only).

3.4.2.3 Information gathering

The mobile application needs to gather information in order to display to the user the news from the categories that have been previously set as interesting to him.

As stated previously there are two completely different ways on how the mobile application can receive new information about the new categories that the user defined as interesting. This means that there are two different implementations when it comes to information gathering.

These two different ways to transfer the information to the mobile device can be illustrated in the following figure:

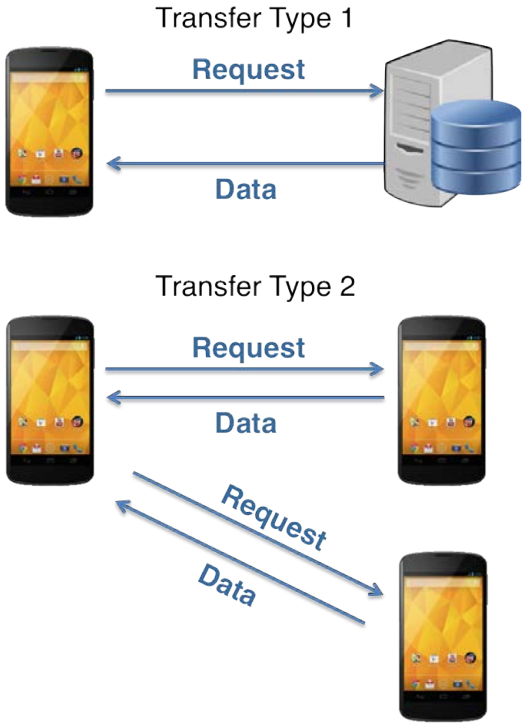

FIGURE 3.3: The two different types of information gathering.

As it is possible to see by analysing the figure 3.3, the mobile application can gather the information from the web service (see section 3.3) or from another mobile devices that are within the local network and have the information it requires.

Web Service Transfer — The more trivial one is the transfer type 1 that can be seen in the figure 3.3, which requires an internet connection to make HTTP

requests to the web service in order to download the news (from the chosen categories) from the server.

This type of requests are done by the service module (see figure 3.2) as soon as the user has an internet connection, downloading the news as soon as possible. In order to achieve this it was implemented by creating a listener called *receiver* that listens to the mobile device connectivity changes. It is possible to declare this listener on the Android's manifest file (configuration file), this way the user when installing also can have knowledge of what type of features are being used to give some sense of security. The declaration in the manifest file is the following:

```
<receiver android:name="com.android.sync.NetworkStateReceiver">
    <intent-filter>
        <action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
    </intent-filter>
</receiver>
```

By analysing the changes of connectivity in order to find if the device has gain access to an internet connection, it is possible to filter some types of connectivity such as bluetooth, near field controller (NFC) and standard cell network.

Afterward, whenever the device gains internet connection, the device invokes the service method called *NewsRetriever* which runs as an *AsyncTask*, allowing the method to run on background without the user even noticing. This method can be customised through the user preferences menu, where the user can decide the interval period between server synchronisations and how further back the user wants to acquire news information (one week, two weeks, one month or since ever).

These preferences make the *NewsRetriever* method act differently in order to proceed with the most appropriate request type. If the user prefers to acquire all the information about the news of its favourite categories that exist on the server the method will create a HTTP GET request to the following URI:

```
URL uri = new URL("http://newsfeeder.herokuapp.com/"+
                Categories.codeNames.get(category_index)+".json");
```

On the other hand if the user only wants the news after a certain period of time, it gets the date of the latest news in the database, if that date is previous than the user's preference then it uses a method that was implemented called *weekAGo* that calculates the date of x weeks ago, where the x is the user's preference and is passed as an argument to the method.

```
String latest = sdf.format(weekAGo(week)).toString();
URL uri = new URL("http://newsfeeder.herokuapp.com/request/"+
                (category_index)+"/'"+ latest +"'.json");
```

After the HTTP GET request has been done the web service will reply with a JSON response (see section 3.3.3.4). This response is parsed by getting the JSON Object *response* that contains an JSON Array with all the news requested. Afterward the JSON Array is parsed into an *ArrayList* that contains all the news information which are inserted into the database as a transaction which keeps the connection to the database open when inserting the news, by inserting as a transaction it is possible to reduce significantly the amount of time spent on this task as it does not lose time on creating and destroying connections to the database.

This way all the information requested is transferred from the web service to the mobile device and stored in the local database, completing the transfer type 1 (see figure 3.3).

Peer-to-Peer Transfer — This type of transfer is done by sharing information between mobile devices that are located within the same network.

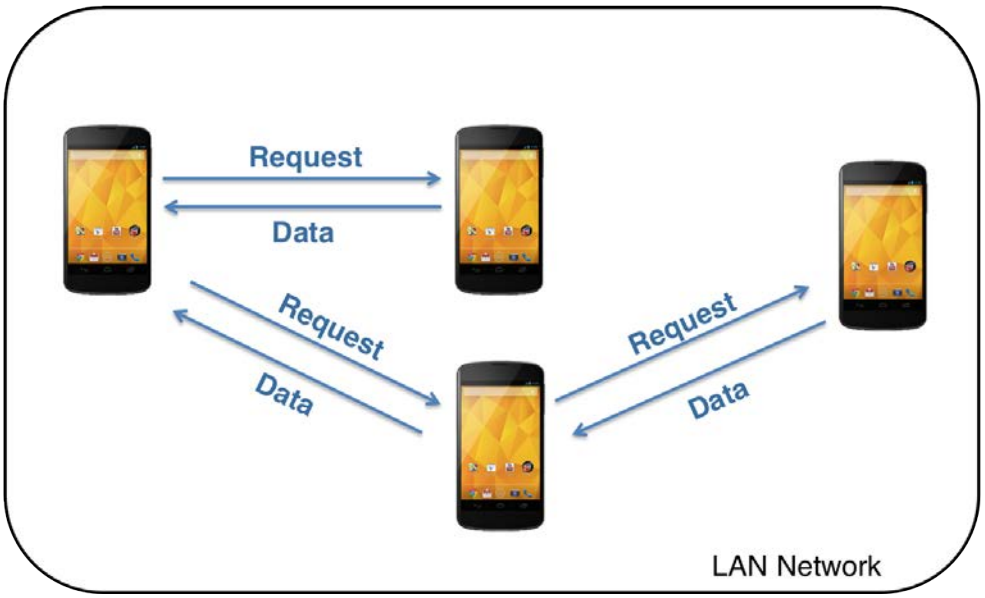

FIGURE 3.4: Mobile devices sharing within a LAN Network.

In figure 3.4 it is possible to observe that sharing does not occur among all the devices, this is because the devices can only share their available information, so transfers can only occur with devices that have matching chosen news categories.

Although, the devices can only send requests and data after they gain awareness that other devices that provide the same service are in the same network. This can be achieved through the process publishing a service over the network and discovery the published services that provide the same service. The implementation of the service publish and discovery are described in the next section.

Due to the complexity of this type of transfer, it will be described more thoroughly over the next four sections.

3.4.2.4 Service discovery

The service discovery is one of the most important steps on mobile peer-to-peer networking, because in order to initiate any interaction among peers it is required that one peer acknowledge the existence of the other.

A service in order to be discovered, first needs to be register its own service across the network so other devices may discover it. This will allow other devices to verify

that there are other services on the network and after resolving their metadata it can be analysed and check what kind of service is being provided.

In the Android application it takes advantage of the same listener to connectivity changes so the service module (see figure 3.2) can start running on the background whenever the mobile device connects to a network.

The service module after starting, it registers a service across the network with all the metadata of the *P2PNews* service which will turn possible the discovery from other mobile devices that such service is being provided.

The registration of the service across the network can be customised by changing some metadata such as name, protocol, transport layer, port and register a listener to handle possible outcomes such as registration or unregistration success or failure.

An example of the configuration of a service before its registration across the network can be given by the following code snippet:

```
public void registerService(int port) {
    NsdServiceInfo serviceInfo  = new NsdServiceInfo();
    serviceInfo.setPort(port);
    serviceInfo.setServiceName("P2PNews-Service");
    serviceInfo.setServiceType("_http._tcp.");
    mNsdManager.registerService(
        serviceInfo, NsdManager.PROTOCOL_DNS_SD, mRegistrationListener);
}
```

The previous code registers a service called *P2PNews-Service* that uses an HTTP protocol over a TCP transport layer and registers using the DNS service discovery protocol.

After registering the service across the network it becomes available to every device that searches for services that have been register on the network. Therefore the next step is to create the service discovery which will allow the awareness of other peers within the network.

In order to start a service discovery it is necessary to implement listeners and then to invoke the network service discovery (NSD) Android's API. This way the listeners that have been implemented can receive updates on the Android's service findings.

The two main methods that are implemented on the Android's network service discovery listener are the *onServiceFound* and *onServiceLost*. These two methods are invoked every time the Android's service discovery framework finds and loses a service on the network. Therefore it is needed to keep track of all the services that have been currently found on the network and have not been lost yet. This process is done as the following method:

```
@Override
public void onServiceFound(NsdServiceInfo service) {
    Log.d(TAG, "Service discovery success" + service);
    if (!service.getServiceType().equals(SERVICE_TYPE)) {
        Log.d(TAG, "Unknown Service Type: " + service.getServiceType());
    }
    else if (!service.getServiceName().contains("P2PNews")) {
        Log.d(TAG, "Not the service I am looking for:" + mServiceName);
    }
    else{
        Log.d(TAG,"Trying to resolve service: " + service.getServiceName());
        mNsdManager.resolveService(service, mResolveListener);
    }
}
```

As we can see by analysing the code above, after finding a service with the characteristics we were searching for it is required to resolve the service in order to get a valid ip address so it can be connected afterward. For a further in depth analysis on how this was implemented see section 3.4.2.5.

On the other hand whenever a service is lost the service must be aware and remove that service from the list of services available on the network. A service lost can be invoked if the other device has disconnected from the network, unregistered its service or even because of some Android internal errors that causes unexpected service loss.

Therefore the listener for the service lost method can be implemented as the following:

```
@Override
public void onServiceLost(NsdServiceInfo service) {
    Log.e(TAG, "service lost" + service);
    Log.d(TAG, "Removing the service: " + service.getServiceName());
    for(int i=0; i<mServiceList.size(); i++){
        if(mServiceList.get(i).getServiceName().
                    compareToIgnoreCase(service.getServiceName())==0){
            Log.d(TAG, "Removing service: " + mServiceList.get(i));
            mServiceList.remove(i);
            break;
        }
    }
}
```

This implementation removes the service from the available services list after it has been lost, which will prevent in the future to send requests when there are no services available in the network.

By keeping track of the services available on the current network it is important as it keeps the device from making request unnecessarily and prevent devices from flooding the network with requests that are never going to be answer.

With this it is possible to keep a good awareness of the mobile devices within the network that provide the same service and afterward initialise interaction between

them, but before it is required that the Android framework resolve the services so it can get a valid IP address that can be used for further interaction.

3.4.2.5 Service Analysis

Once a network service has been discovered it is necessary to wait for the Android system to resolve the service's IP address in order to allow future connections to the device broadcasting that specific service.

Afterward it is required to check if the service that is being providing the type of service it is expected and then check if the service found is not an echo of our own published service.

This can be done by analysing the IP address of the service, after it has been resolved, and check if it is the same as the device that is searching. Furthermore it is also required to check if the service has already been registered in the device's system and if it is an echo from another device that has been previously found.

Both of these checks can be verified by the following source code on the *onServiceResolved* listener:

```
Log.e(TAG, "Resolve Succeeded. " + serviceInfo);
if(serviceInfo.getHost().getHostAddress()
                .compareToIgnoreCase(Protocol.getIPAddress(true))==0){
    Log.d(TAG, "This is my own service.");
    return;
}

for(NsdServiceInfo nsdInfo : mServiceList){
    if(serviceInfo.getHost().getHostAddress()
            .compareToIgnoreCase(Protocol.getIPAddress(true))==0){
        Log.d(TAG,"Already on the list.");
        return;
```

```
        }
    }

/**
* New service has been found
*/
Log.d(TAG, "Adding the service: " + serviceInfo.getHost().getHostAddress());
mServiceList.add(serviceInfo);
```

After the service has been added to the list that keeps track of all the services that provide the expected service over the network, it is required to send a request to that service's device in order to discover if it has any data that would be useful to the sender.

These requests are the method to keep track of the information that exist over the network. This way devices get awareness of not only the devices that exist over the network that are providing the same service but also all the information that the other devices have.

3.4.2.6 Data requests

As a way to gain awareness of the information that other devices over the network have, each device can send data requests through a control channel using a *Multi-Cast Socket* that every peer that provide the service starts listening as soon as it is aware of other devices on the network.

This approach avoid one device having to make multiple requests if there are several peers on the network and it requires new information.

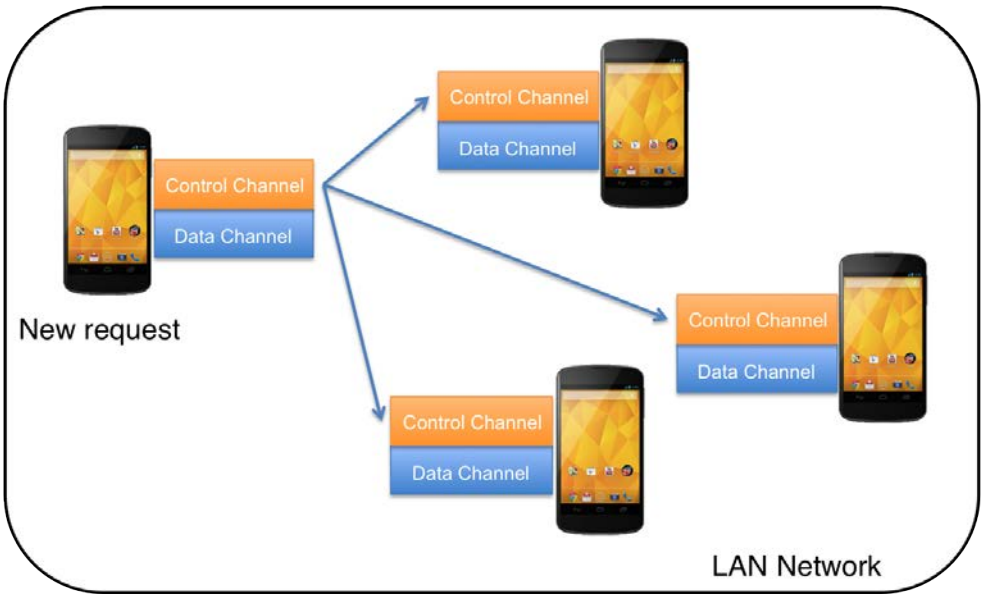

FIGURE 3.5: Example of a data request being sent.

The figure 3.5 shows that the request sent by the device (New request) is received by all the other devices within the network that are listening to the *MultiCast Socket* as the control channel.

Each request sent through the control channel has some metadata that allows the receiving devices to check for the information requested and if it exists, sends the data to the request sender.

In order to keep track of the requests that are sent over the network, each request has its own unique identification string that is created by the combination of the Android ID and the number of milliseconds since the 1st January 1970.

Every request have the following structure:

```
Search <unique_id> <num_categories> <category> <last_date> ...
<ip_address> <port>
```

As it is possible to see the requests contain the list of categories and the date of the last information gathered on it, so it is not necessary to send multiple requests if the device has several categories.

```
Search cff7efaca91dccaa-1376151971277 6 Art 2013-08-10T16:14:58Z
```

```
Business 2013-08-08T12:50:52Z Entertainment 2013-08-10T16:15:01Z
Politic 2013-08-08T12:50:55Z Technology 2013-08-08T12:50:57Z
Travel 2013-08-08T13:05:53Z 192.168.1.5 8000
```

After other peers connected to the network receive a request, the devices will send to the request sender the information they have that is useful for such request.

3.4.2.7 Data sharing

Upon receiving a new data request the device promptly accesses the database by invoking a method to check the date of the latest information on each category (see section 3.4.2.2) that appears on the request.

Once the device compared all the database dates with the dates on the request it creates a queue with all the categories that are required to send and the dates of the oldest information that is necessary.

The device executes a background process by using an *AsyncTask* for each category in the queue and sends over a socket (data channel) a JSON object containing the data request for that specific category.

This process includes accessing the database to get all the news that are prior to the date sent on the request:

```
ArrayList<ArrayList<String>> news = dh.selectAllNewer(category,latest);
```

And afterward it is required to create a JSON object that contains a JSON array with all the news entries that are returned by the method above. In order to create this JSON object, it is used the *org.json* library that allows the construction of a JSON object by invoking the following methods:

```
JSONObject sendJSON = new JSONObject();
sendJSON.put("error","none.");
```

```
sendJSON.put("database",category);
JSONArray arr = new JSONArray();
int index = 0;
for(ArrayList<String> obj : news){
    JSONObject each = new JSONObject();
    each.put("name",obj.get(0));
    each.put("summary",obj.get(1));
    each.put("url",obj.get(2));
    each.put("published_at",obj.get(3));
    each.put("updated_at",obj.get(3));
    each.put("guid",obj.get(4));
    arr.put(each);
}
sendJSON.put("response",arr);
```

At the end of having the JSON object with all the information regarding a specific category the mobile device creates a socket connecting to the ip and port of the data request and sends the information.

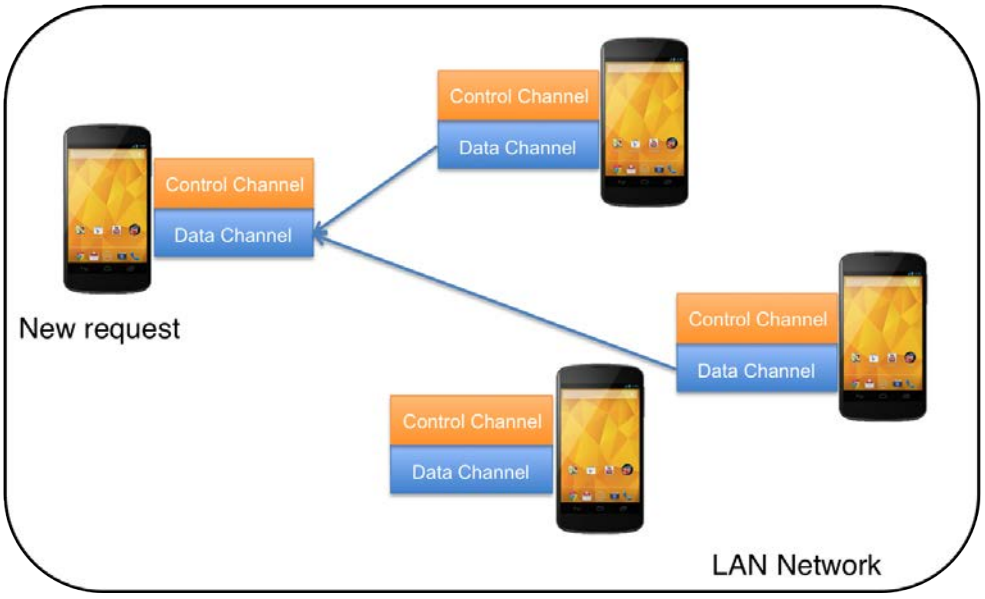

FIGURE 3.6: Data transfer to the request sender.

On the figure 3.6 two of the devices on the network have the same categories as their favourite and have all or part of the information requested by the initial peer, sending to it that data.

The mobile device when receiving the information analyses the data and stores to the corresponding table on the database. If it receives data that had been previously received this information is not stored on the database as it has the same ids of the previous data.

Upon receiving new information, either from the server or from another peer, the device sends through the control channel an update broadcast so that other devices on the network are aware of the new data available on that specific peer.

```
Update <category> <latest date>
```

This way it is possible to keep track of all the information that exists on the network and as soon as new data becomes available it can be distributed over the network to devices that require such information.

3.5 Graphical User Interface

This section is dedicated to the graphical user interface (GUI) of the mobile application that had some requirements from the start, where it should be appealing to the daily user and should keep user completely unaware of the more technical concepts behind it.

In order to create a logo that could transmit the feeling of sharing and building something while still giving it a technological aspect, the application logo is the following:

FIGURE 3.7: Mobile device application logo.

The mobile application should also be intuitive and easy to use and personalise. Taking advantage of the Android UI design principles[33] to produce an GUI with a small learning curve. At the beginning the user is requested to go to the application settings to add to the home screen its favourites categories.

FIGURE 3.8: Initial home screen.

In the settings menu the user can personalise the application to fit his needs, by turning on and off the peer-to-peer sharing and therefore still use the application as a news reader in case the user does not want to share any data. Other settings can be set as well, such as syncing frequency and how far back in time the user wants to receive news information.

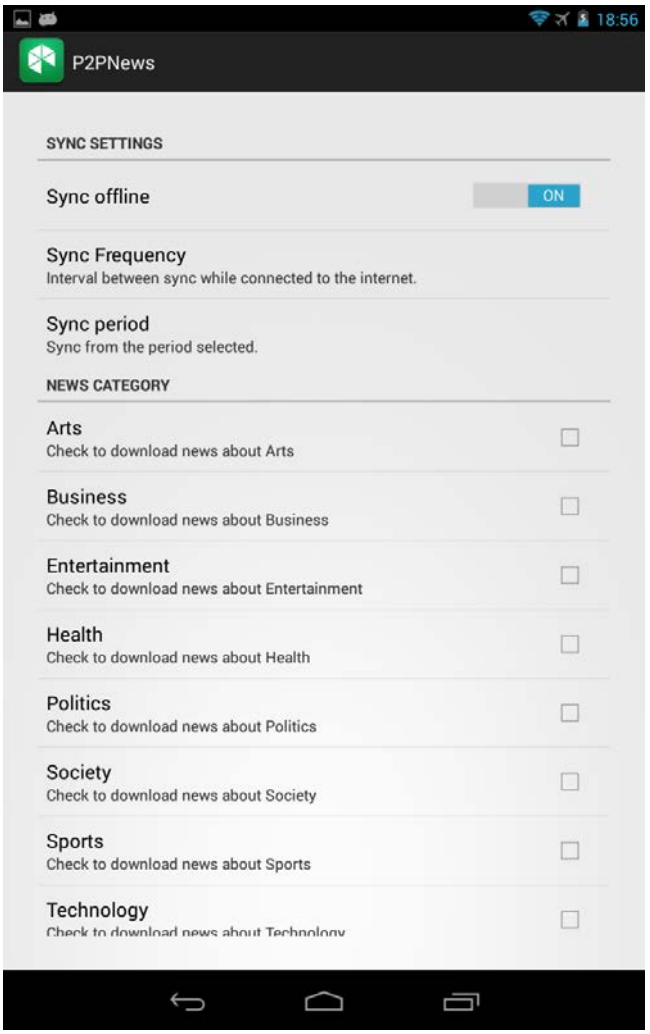

FIGURE 3.9: Settings menu.

As it is possible to see in the figure 3.9, in this menu the user can determine which news category are relevant to him.

After adding the relevant categories they will be added to the home screen, creating an easy to use interface to navigate through the categories (figure 3.10).

FIGURE 3.10: Home screen with categories.

Once the categories are added the mobile application service starts searching for news information of those categories as well, sending requests if there are any peers on the network to synchronise the information as soon as possible or download from the internet if connected.

After the mobile device receives the data for any of the categories the user can simply click on the category slot and it will display an UI that contains a list with all the news that have been transferred previously to the device (figure 3.11).

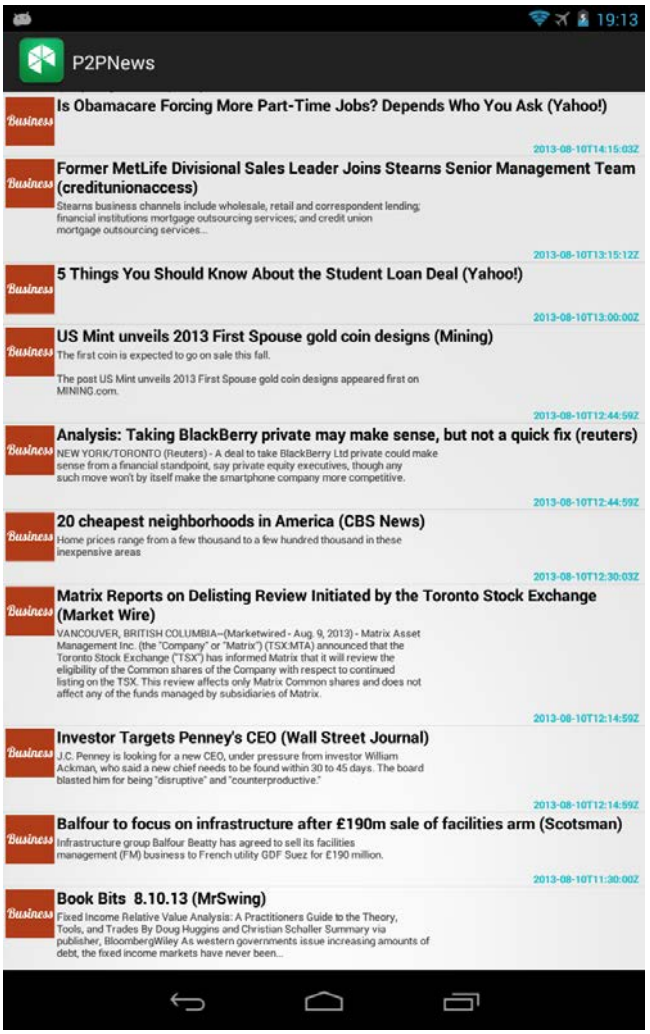

FIGURE 3.11: List of news of a category.

This way the user have quick access to all the news of one specific category and can read them as he browses them.

Furthermore, by clicking on the Android's settings button the user can select to synchronise the news at that moment if connected to the internet (figure 3.12).

FIGURE 3.12: Synchronisation button.

Chapter 4

Evaluation

In order to analyse the feasibility of such system it was decided to take four metrics to study the impact of such architecture on the mobile device and to check if it fits the environment it is inserted on.

The metrics chosen for evaluation of the system were the rate of success transfers among peers that have the requested data, the transfer time and speed between the devices on a LAN network and the impact of the application and service on the device's battery consumption.

To evaluate the system the devices used during the experiments were an ASUS Nexus 7 and a LG Nexus 4, both devices developed closely with Google, developer of the Android OS.

4.1 Transfer Analysis

In this section it will be explained how the transfer analysis were implemented in order to produce results to be studied afterward.

With the purpose of analysing the rate of success transfer, the time and speed of each transfer it has been implemented an *AnalysisTest* class that can reproduce testing scenarios and log the outcome.

This class runs on background in order to reproduce a normal usage environment and do not interfere with the user's interaction with the device so it can work as well without any user awareness.

4.1.1 Procedure

In order to create a testing environment there are some variables that are needed to be reproduced. In the environment it is required to exist at least two devices and from time to time one device must send a request and wait to receive the data from the other device.

This experiment was reproduced two hundred times with requests being sent every thirty seconds and it was measured the number of bytes transferred through the data channel as well as the time elapsed between when the request was sent and all the data requested was received, finishing a "request cycle".

The first step to each experiment is to delete all the data stored in the database previously so the device has no prior data and can always ask the same data to another peer. Afterward the device executes a data request and waits for thirty seconds to start another experiment.

```
int i = 0;
int num_Of_Experiment = 100;
while(i<num_Of_Experiment){
    Log.d("TestEnvironment","Trial number: "+i);
    try{
        //Delete all the stored data
        DataHelper dh = new DataHelper(mContext);
        dh.deleteAll();
        dh.close();
        //Send a new data request
        new SyncManager(mContext).executeOffline();
```

```
        //Wait 30 seconds until new trial
        Thread.sleep(30000);
    }
    catch(InterruptedException ie){
        Log.e("TestEnvironment", ie.getMessage());
    }
    i++;
}
```

The *executeOffline* method and the background process to receive data through the data channel were modified in order to create a log file to save each one of the actions with a timestamp of the exact moment.

```
Calendar calendar = new GregorianCalendar(TimeZone.getTimeZone("GMT"));
//BufferedWriter for performance, true to set append to file flag
BufferedWriter buf = new BufferedWriter(new FileWriter(logFile, true));
buf.append(calendar.get(Calendar.HOUR)+":"+calendar.get(Calendar.MINUTE)+
    ":"+calendar.get(Calendar.SECOND)+":"
    +calendar.get(Calendar.MILLISECOND)+"|"+text);
buf.newLine();
buf.close();
```

An example of the output of a single experiment to the log file is:

```
2:49:47:973|Search 8c20fff20c0be722-1376318987819 1 Business
    2013-07-29T15:49:47 192.168.1.3 8000
2:49:52:990|Received data for: Business
2:49:53:123|Size: 1159519 Bytes.
```

By running this procedure it is possible to reproduce data to be further analysed to check the success rate, mean time and speed for the data transfers.

Although, due to the length of the log file it was necessary to create a script that could analyse the output and calculate the results. This script was created using the script programming language Python which is a powerful and fast language that allows quick prototyping and data analysis[34].

The Python script analyses every line of the log file and gathers all the data of each experiment and while it is gathering all the information also searches for outputs that are not complete, meaning that there was an error during the transfer and it was completed.

This is done by analysing every single line and storing the timestamp along with the first word on the metadata (search, received and size), if for some reason the following sequence pattern does not occur it registers the error:

```
Search -> Received -> Size
```

After the file has been analysed through the Python script it is possible to check how many transfers were successful and how many failed.

Once the information of each successful experiment has been gathered it is possible to compare the timestamps in order to get the time elapsed for each experiment and calculate a mean of them all. Since the size information is also available it is also possible to analyse the transfer speed between the two peers.

In the two hundred experiments run the success rate was 93.5%, which means that only 6.5% of the transfers did not complete successfully.

The results regarding this analysis can be displayed in the following chart:

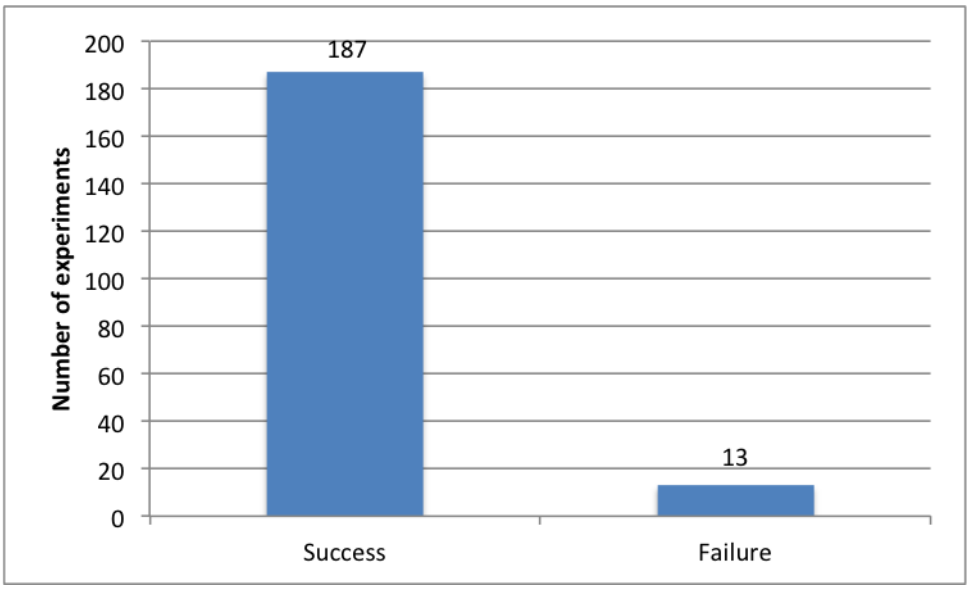

FIGURE 4.1: Transfer success rate chart.

Then by analysing the subset of successful transfer cycles it is possible to get the elapsed time for each one of the 187 experiments. The average elapsed time for these 187 experiments is 4,79 seconds.

The following chart demonstrates the time elapsed for every experiment and can be compared with the average time line.

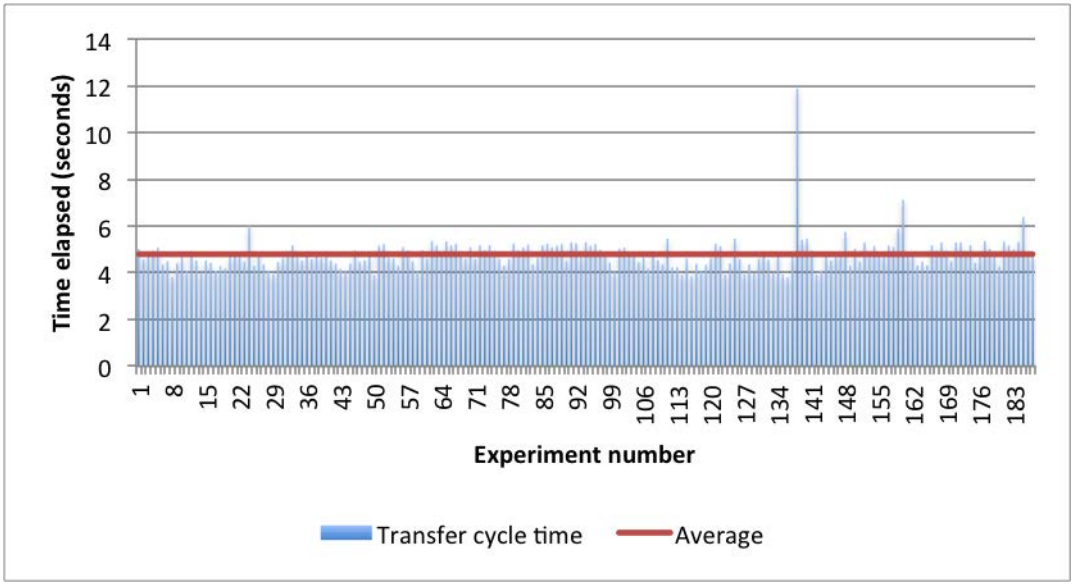

FIGURE 4.2: Average transfer cycle time.

As it is possible to verify by analysing the chart on the figure 4.2 that even though there are in a very few cases a big delay on the transferring time, almost every

time it keeps under 6 seconds which means that even on a unstable environment the results have a small deviation.

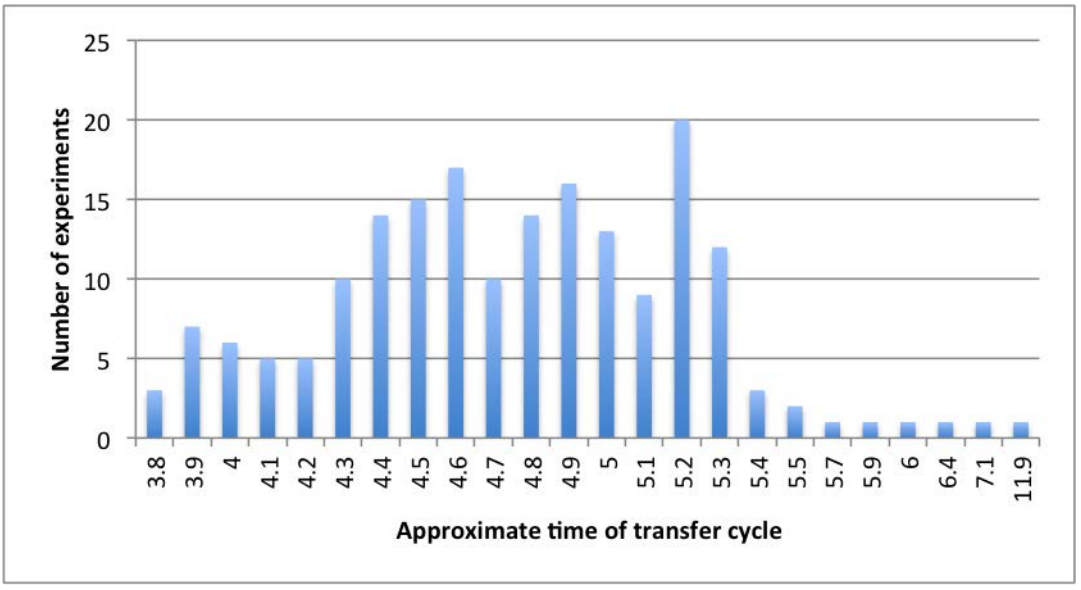

FIGURE 4.3: Approximate transfer cycle time.

By rounding the time of the transfer cycles to 1 decimal place it is possible to gather some of the results together to display a clean view of the outcome in the figure 4.3.

The last metric used to analyse the transfer cycles is the transfer speed (kb/s) which can be calculated by the following formula:

$$speed = \frac{num\_bytes}{time\_elapsed * 1024}$$

The Python script was used to calculate the speed per experiment and then calculate the average of all the experiments.

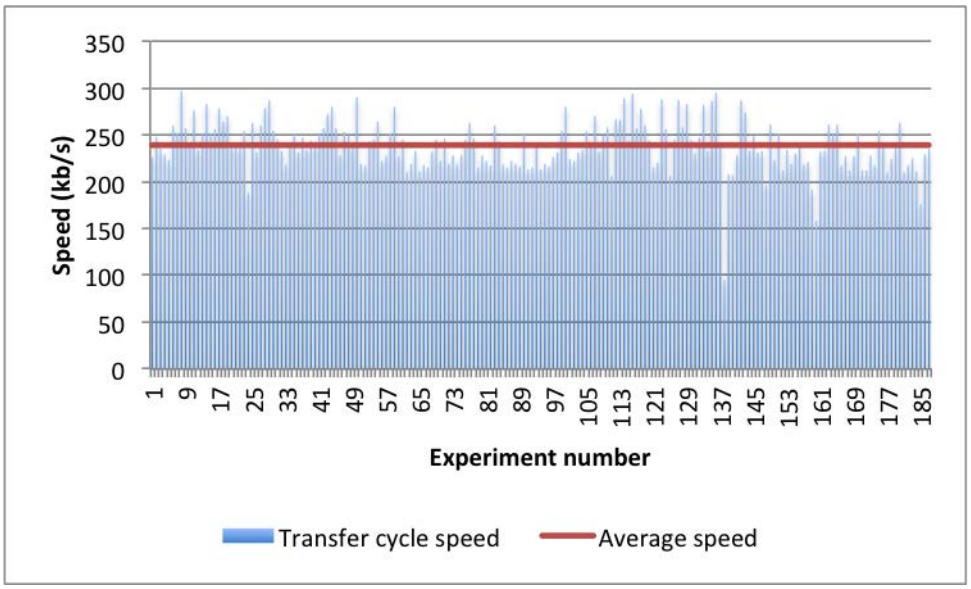

FIGURE 4.4: Transfer cycle speed.

The average speed achieved with the 187 experiments for a transfer cycle is $238kb/s$. In the figure 4.4 it is also possible to see that on the experiment number 138 the speed of the transfer achieves its minimum with $95kb/s$, taking around 11 seconds to complete the transfer cycle.

4.2 Battery Consumption Analysis

In order to analyse the battery consumption there are three scenarios that can be compared, when the service is being run on the background of the device and from time to time it finds new services and transfer data between the peers simulating a normal usage environment, the normal device without the service running but with WiFi turned on and for last the device with no WiFi radio usage.

The service was running for two hours and every time an action was taken it would analyse the battery level and log it into a file. The battery percentage was calculated by using the Android's available API to do so:

```
IntentFilter ifilter = new IntentFilter(Intent.ACTION_BATTERY_CHANGED);
Intent batteryStatus = mContext.registerReceiver(null, ifilter);
int level = batteryStatus.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
```

```
int scale = batteryStatus.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
float batteryPct = (level / (float)scale)*100;
Protocol.appendLog("Battery: " + batteryPct);
```

After the trial run is complete the log file can be analysed through a Python script to transform the raw data into a comma separated value (*.csv*) file to be able to later reproduce a chart with the battery usage for both scenarios.

Every trial run was made with the device screen on at its lowest brightness without any user interaction.

Having done the three trial runs the chart obtained by processing the log raw data it was the following:

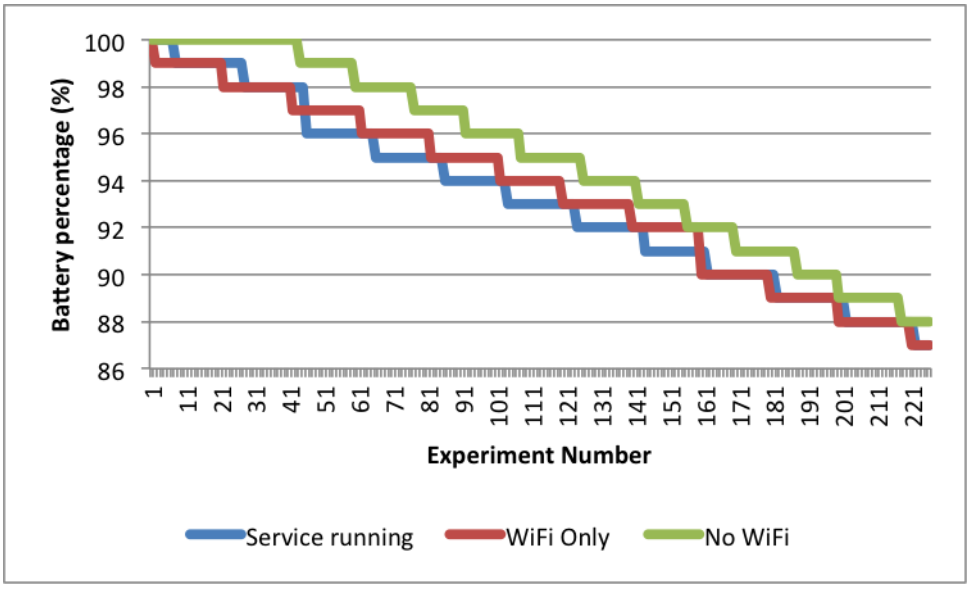

FIGURE 4.5: Battery consumption at each experiment.

As it is possible to see in the figure 4.5 the battery usage for when using the service is fairly similar to when the device has the WiFi radio on and have no service related tasks. Surprisingly, the device's energy consumption when not using the WiFi radio was also similar to the other two.

This proves that by using the service implemented the battery consumption is not deeply affected, which means that the service itself does not have a big impact on the mobile device's normal battery usage.

Chapter 5

Discussion

The evaluation shown in the chapter 4 can demonstrate a successful implementation of the concept proposed in this research project. This proves that using peer-to-peer networking to share social data between mobile devices can be implemented using the constrains chosen for this project.

During the development of this research project there were several paths that could be taken and in this chapter they will be explained and how that affected the outcome. Such architecture can be implemented in the three different technologies the mobile devices have to transfer information among other peers: Bluetooth, WiFi and WiFi Direct.

The Bluetooth is a low power radio, which would mean a lower battery consumption but on the other hand it would also decrease substantially the radius for finding peers as well as the transfer rate is much smaller than the other two available.

Due to the reason stated above, the only two viable choices were the WiFi and the WiFi Direct. Both of these technologies use the same radio, meaning that the power consumption is roughly the same as it inherits all the power saving and quality of service that has been developed for WiFi infrastructure mode over the years[35].

Prior research on WiFi Direct performance shows that the current technology is unstable on its implementation on Android, mainly on discovering peers and successfully connecting to them[36]. Such instability can be also seen by several topics on highly regarded forums such as the open source Google groups for the Android platform[37].

Concerning that one of the main goals of this project is to turn possible peer-to-peer social data sharing without any user interaction, the WiFi Direct at its currently implementation (Android 4.2.2) makes it necessary that the user accept all incoming connections. This is another problem that has been researched, discussed and due to currently security issues there is no work around for such interaction.

A WiFi Direct implementation was made using the framework presented in this research project but only to discover the drawbacks stated above.

These reasons make it impossible to use WiFi Direct to implement this architecture in order to analyse its true impact on the mobile environment with the goals initially set to this project.

Even though the WiFi implementation have the constrain that users must be connected to the same network, the framework implemented can be used as a base for future implementation on a WiFi Direct infrastructure when it becomes more stable.

Due to the technical similarities between WiFi and WiFi Direct (except peer connection) the results can be also taken in consideration for a future implementation on WiFi Direct.

As the results show, due to the advances in the hardware of the mobile devices available peer-to-peer data sharing is a possibility and can be used for several purposes without having a great impact on the mobile device battery consumption comparing to when having WiFi turned on.

Chapter 6

Conclusion

The main goal of this research project was to verify the possibility to use the mobile devices capabilities to create a framework that allows peer-to-peer data sharing across a network with peer discovery and no user interaction.

Following the methods presented on chapter 3 it was possible to implement a framework to discover peers across a network and keep track of them. Furthermore the framework implemented allows the device to acknowledge all the existing information across the local network and request data from others peers when necessary.

This meaning that all the goals set to this research project were successfully achieved through the implementation of a web service to provide information about all the news categories as a single source and the development of a mobile application to allow the devices to share information across the network without any user interaction prior or during the process.

It is possible to conclude from the results demonstrated on the chapter 4 that this implementation allow a steady method, having in mind the mobile network instability, to share information with a high rate of success without having a great impact on the device's battery consumption.

The successful implementation of the work presented in this research project allows it to be used as reference to future projects regarding the same topic using other technologies for comparison.

Chapter 7

Future Work

Regarding social data sharing using peer-to-peer networks there are a lot of further concepts and implementations that could be achieved by changing the metadata provided with each news. An example is the possibility to also share the number of people across the network that liked a certain news, comments on some news by other peers and much more.

Through the usage of metadata on social data sharing it is also possible to gather demographic information and analyse the behaviour of a social network in several ways, such as preferences, influence a social network has on each individual and others.

Although this research project designed and implemented a method to share information across a network using mobile devices as peers with capability to acknowledge each others through service discovery, there is still a lot to research on this field as the technologies improve.

Appendix A

WS News Management

In order to allow content management through a web browser, the following methods were implemented:

```
# GET /arts/new
# GET /arts/new.json
def new
    @art = Art.new

    respond_to do |format|
      format.html # new.html.erb
      format.json { render :json => @art }
    end
end

# GET /arts/1/edit
def edit
    @art = Art.find(params[:id])
end

# POST /arts
```

```
# POST /arts.json
def create
    @art = Art.new(params[:art])

    respond_to do |format|
      if @art.save
        format.html { redirect_to @art, :notice =>
            'Art was successfully created.' }
        format.json { render :json => @art, :status =>
            :created, :location => @art }
      else
        format.html { render :action => "new" }
        format.json { render :json => @art.errors, :status =>
            :unprocessable_entity }
      end
    end
end

# PUT /arts/1
# PUT /arts/1.json
def update
    @art = Art.find(params[:id])

    respond_to do |format|
      if @art.update_attributes(params[:art])
        format.html { redirect_to @art, :notice =>
            'Art was successfully updated.' }
        format.json { head :no_content }
      else
        format.html { render :action => "edit" }
        format.json { render :json => @art.errors, :status =>
```

```
            :unprocessable_entity }
      end
    end
end

# DELETE /arts/1
# DELETE /arts/1.json
def destroy
    @art = Art.find(params[:id])
    @art.destroy

    respond_to do |format|
      format.html { redirect_to arts_url }
      format.json { head :no_content }
    end
  end
end
```

Bibliography

[1] The Nielsen Company. The social media report, state of the media: 2012. Technical report, NM Incite, 2012. URL http://www.targetspot.com/wp-content/uploads/2012/12/The-Social-Media-Report-2012.pdf. [Accessed April 3, 2013].

[2] International Data Corporation. IDC Worldwide mobile phone tracker, February 2013. URL http://www.idc.com/getdoc.jsp?containerId=prUS23946013#.UWCkuKt4ZAj. [Accessed April 3, 2013].

[3] Android developers. About Android, 2012. URL http://developer.android.com/about/index.html. [Accessed April 4, 2013].

[4] Fumiyuki Adachi. Wireless past and future–evolving mobile communications systems–. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 84(1):55–60, 2001.

[5] KH Gamble. Wireless tech trends 2010. trend: smartphones. *Healthcare informatics: the business magazine for information and communication systems*, 27(2):24–26, 2010.

[6] Ravi Kumar, Jasmine Novak, and Andrew Tomkins. Structure and evolution of online social networks. In *Link Mining: Models, Algorithms, and Applications*, pages 337–357. ISBN 1441965149.

[7] Mehedi Bakht et al. United we find: enabling mobile devices to cooperate for efficient neighbor discovery. In *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications*, page 11. ACM, 2012.

[8] Mehedi Bakht, Matt Trower, and Robin Kravets. Searchlight: helping mobile devices find their neighbors. In *Proceedings of the 3rd ACM SOSP Workshop on Networking, Systems, and Applications on Mobile Handhelds*, page 9. ACM, 2011.

[9] Daniel H Steinberg and Stuart Cheshire. *Zero configuration networking: The definitive guide*. O'Reilly, 2010. ISBN 9780596101008.

[10] Se Gi Hong, Suman Srinivasan, and Henning Schulzrinne. Accelerating service discovery in ad-hoc zero configuration networking. In *Global Telecommunications Conference, 2007. GLOBECOM'07. IEEE*, pages 961–965. IEEE, 2007.

[11] Stuart Cheshire and Marc Krochmal. Dns-based service discovery. *Work in Progress*, 2011.

[12] Stuart Cheshire and Marc Krochmal. Multicast dns. *Internet Engineering Task Force*, 2011.

[13] Arthur van Hoff. Java implementation of Multicast DNS. URL http://jmdns.sourceforge.net/. [Accessed April 15, 2013].

[14] Rüdiger Schollmeier. A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, pages 101–102. IEEE, 2001.

[15] Android developers. Android technical information. Android Open Source Project, 2010. URL http://source.android.com/tech/index.html. [Accessed April 4, 2013].

[16] Ed Burnette. *Hello, Android: Introducing Google's Mobile Development Platform*. Pragmatic Bookshelf, 2nd edition, 2009. ISBN 1934356492.

[17] Android developers. Get the Android SDK, 2010. URL http://source.android.com/tech/index.html. [Accessed April 4, 2013].

[18] Bluetooth SIG. A look at the basics of Bluetooth wireless technology, 2013. URL http://www.bluetooth.com/Pages/basics.aspx. [Accessed April 4, 2013].

[19] Bluetooth SIG. Fast facts - Bluetooth SIG, 2013. URL http://www.bluetooth.com/Pages/Fast-Facts.aspx. [Accessed April 4, 2013].

[20] Wi Fi Alliance. Wi-Fi Direct, 2013. URL http://www.wi-fi.org/discover-and-learn/wi-fi-direct. [Accessed April 5, 2013].

[21] Sang Shin. Introduction to JSON (JavaScript Object Notation). *Presentation www.javapassion.com*, 2010.

[22] Alex Rodriguez. Restful web services: The basics. *Online article in IBM DeveloperWorks Technical Library*, 36, 2008.

[23] Yukihiro Matsumoto. Ruby Programming Language, 2013. URL http://www.linuxdevcenter.com/pub/a/linux/2001/10/25/ruby.html. [Accessed April 6, 2013].

[24] Colin Steele. An introduction to Ruby, 2001. URL http://www.linuxdevcenter.com/pub/a/linux/2001/10/25/ruby.html. [Accessed April 6, 2013].

[25] Viswa Viswanathan. Rapid web application development: a ruby on rails tutorial. *Software, IEEE*, 25(6):98–106, 2008.

[26] Atif Aziz and Scott Mitchell. An Introduction to JavaScript Object Notation (JSON) in JavaScript and .NET, 2007. URL http://msdn.microsoft.com/en-us/library/bb299886.aspx. [Accessed April 6, 2013].

[27] International Data Corporation. IDC Worldwide mobile phone tracker, May 2013. URL http://www.idc.com/getdoc.jsp?containerId=prUS24108913. [Accessed July 28, 2013].

[28] Ryan Bigg et al. Configuring Rails Applications, 2013. URL http://guides.rubyonrails.org/configuring.html#configuring-a-database. [Accessed July 29, 2013].

[29] Heroku Inc. Heroku — Cloud Application Platform, 2013. URL https://www.heroku.com/. [Accessed August 2, 2013].

[30] New Relic Inc. Application Performance Management & Monitoring — New Relic, 2013. URL http://www.newrelic.com/. [Accessed August 2, 2013].

[31] John Mettraux. Rufus—Scheduler gem library, 2013. URL https://github.com/jmettraux/rufus-scheduler. [Accessed May 8, 2013].

[32] Paul Dix. Feedzirra gem library, 2013. URL https://github.com/pauldix/feedzirra. [Accessed May 9, 2013].

[33] Android developers. Android design principles, 2013. URL http://developer.android.com/design/get-started/principles.html. [Accessed June 21, 2013].

[34] Python Software Foundation. Python programming language, 2013. URL http://www.python.org/about/. [Accessed August 7, 2013].

[35] Daniel Camps-Mur, Andres Garcia-Saavedra, and Pablo Serrano. Device to device communications with wifi direct: overview and experimentation. *IEEE Wireless Communications Magazine*, 2012.

[36] Hugo Negrette Otaola and Miguel Sosa. Using multiple transport networks in netinf enabled android devices. Master's thesis, KTH, 2012.

[37] Android Software Developers. WiFi Direct questions, 2013. URL http://goo.gl/RwfZxS. [Accessed August 13, 2013].