Collecting and processing geographic coverage information of mobile networks

José Miguel de Carvalho Branco Maia
Mestrado Integrado de Engenharia de Redes e Sistemas Informáticos
Departamento de Ciência de Computadores
2015

Orientador
Sérgio Crisóstomo, Professor Auxiliar,
Faculdade de Ciências da Universidade do Porto

Coorientador
Rui Prior, Professor Auxiliar,
Faculdade de Ciências da Universidade do Porto

U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

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

O Presidente do Júri,

Porto, _____/_____/_____

Thanks to all my friends throughout all these years who have done their part to make me who I am.

Thanks to my family, for always being there.

Thanks to my supervisors, Professor Rui Prior and Professor Sérgio Crisóstomo, for giving me the opportunity to work with them and to learn from them.

Obrigado, Muchas gracias, Merci Bien, Tudo é Kanimambo **(João Maria Tudela)**

Abstract

Over the last two decades, we have seen an exponential growth in the use of mobile phones. This growth was accompanied by a change in their usage pattern — the access to data services has overtaken voice calling, which was the almost exclusive use of mobile phones. It was also accompanied by a change in the devices themselves, with the market shifting to smartphones.

The selection of a mobile service provider can be difficult for the end users, given the lack of unambiguous, accurate and independent information on coverage and quality of the wireless access, including voice and data, with every provider claiming their particular signal quality, network speed and/or pricing is the best.

This motivated us to develop a service for measuring mobile network parameters using Android smartphones, and display the data, adequately processed, to the users in a mapping application. The service is based on crowdsourcing, an attractive approach due to the pervasiveness of smartphones equipped with both cellular and Wi-Fi hardware and with Global Positioning System (GPS) receivers that allow us to associate obtained measurements with its geographical locations. Users worldwide can contribute to the database, increasing the availability and the quality of the data, obtained independently from the service providers.

We adopted a client/server architecture for the service, based on three main components: a mobile client application that obtains data from the devices, a server that receives that data and processes it, and a web application that displays the processed data to the end-user.

In this work we describe the service and its different components and discuss our solutions to the technical problems that emerged during the design and implementation of the service.

Keywords: Cellular networks, WI-FI, monitoring, Mobile crowdsourcing, Android

Resumo

O papel dos dispositivos móveis na experiência do dia-a-dia da humanidade tem vindo a crescer exponencialmente ao longo das últimas duas décadas — o acesso a serviços de voz, anteriormente, o principal uso dos telefones móveis, foi ultrapassado pelo acesso a serviços de dados. Este crescimento também foi acompanhado por alterações nos dispositivos em si, com o mercado a evoluir para *smartphones*.

A seleção de um provedor de serviços móveis pode ser difícil para o utilizador final, dada a falta de informação independente, precisa e não ambígua sobre a cobertura e qualidade do acesso sem fios, incluindo voz e dados, uma vez que cada provedor alega ter a melhor qualidade de sinal, velocidade da rede e/ou preço.

Isto motivou o nosso desenvolvimento de uma aplicação que possa medir parâmetros das redes móveis em smartphones Android e mostrar esses dados, adequadamente processados, aos utilizadores, através de um mapa. O serviço é baseado em *crowdsourcing*, uma hipótese atrativa, dado a disseminação de smartphones equipados com hardware celular e Wi-Fi e com recetores GPS que nos permitam associar medições à localização geográfica onde foram obtidas. Utilizadores por todo o mundo podem contribuir para a base de dados, aumentando a disponibilidade e qualidade dos dados, que serão obtidos de forma independente dos provedores de serviços.

Adotámos uma arquitetura cliente/servidor para o serviço, baseada em três principais componentes: uma aplicação móvel (o cliente) que obtém dados dos dispositivos móveis, um servidor que recebe esses dados e processa-os, e uma aplicação web que mostra os dados processados ao utilizador final.

Neste trabalho descrevemos o serviço e os seus diferentes componentes e discutimos as nossas soluções para os problemas técnicos que surgiram durante o desenho e implementação do serviço.

Palavras-chave: Redes celulares, WI-FI, monitorização, *crowdsourcing* móvel, Android

Contents

List of Tables

List of Figures

Listings

Acronyms

2G	Second generation mobile network
3G	Third generation mobile network
4G	Fourth generation mobile network
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
ASU	Arbitrary Signal Units
BID	Billing Identification
BSSID	Basic Service Set Identifier
BSS	Basic Service Set
CCMP	Counter Mode CBC-MAC Protocol
CDMA	Code Division Multiple Access
CID	Cell ID
CI	Cell Identification
CSS	Cascading Style Sheet
CSV	Comma-separated Values
dBm	Decibel-milliwatts
DBMS	Database Management System
DNS	Domain Name System
EAP	Extensible Access Protocol
EDGE	Enhanced Datarates for GSM Evolution
ESS	Extended Service Set
FDD	Frequency Divided Duplexing
GIS	Geographical Information System
GPL	GNU General Public License
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile Communications
HSDPA	High-Speed Downlink Packet Access
HSPA	High-Speed Packet Access
HSUPA	High-Speed Uplink Packet Access
HTTP	Hypertext Transfer Protocol
IBSS	Independent Basic Service Set
IMSI	International Mobile Subscriber Identity
ISP	Internet Service Provider
JSON	JavaScript Object Notation

LAC Local Area Code

LTE Long Term Evolution

MAC Media Access Control

MBR Minimum Bounding Rectangle

MCC Mobile Country Code

MIMO Multiple Input Multiple Output

MLS Mozilla Location Services

MMS Multimedia Messaging Service

MNC Mobile Network Code

MVC Model-View-Controller

NID Network Identification Number

ORM Object-relational mapping

OSM OpenStreetMap

OS Operating System

PCI Physical Cell Identification

PSC Primary Scrambling Code

PSK Pre-Shared Key

QoS Quality of Service

RSCP Received Signal Code Power

RSRP Reference Signal Received Power

RSRQ Reference Signal Received Quality

RSSI Received Signal Strength Indicator

RTT Round-trip Time

SDBMS Spatial Database Management System

SDR Software-Defined Radio

SID System Identification Number

SIM Subscriber Identity Module

SMS Short Message Service

SQL Structured Query Language

SSID Service Set Identifier

TAC Tracking Area Code

TCP Transmission Control Protocol

TKIP Temporal Key Integrity Protocol

UDP User Datagram Protocol

UMTS Universal Mobile Telecommunications System

URL Uniform Resource Locator

USRP Universal Software Radio Peripheral

UTC Coordinated Universal Time

WEP Wired Equivalent Privacy

WGS World Geodetic System

WLAN Wireless Local Area Network

WPA Wireless Protected Access

WPS Wi-Fi Protected Setup

Chapter 1

Introduction

Mobile phones have gained an important place in our lives throughout the years, and as their usage grew, so did their data services. According to the Pew Research Center, as of January 2014, 90% of American adults have a cell phone, and 63% of adult cellular service subscribers use their phones to go online [1]. Additionally, one third of these cell Internet users *mostly use their phone to access the Internet.*

In Portugal's case, as of the third quarter of 2014, the penetration rate of cell phones (not considering unused mobile stations and other devices such as cellular internet modems) is 112.9 per 100 inhabitants - furthermore, out of 11 931 thousand mobile users who have potential access to mobile broadband services, 4 824 thousand (40%) have utilized Third generation mobile network (3G) or better services in the last month of the quarter [2].

The report also mentions that, gradually, less users have been claiming as the main reason to choose an operator the fact that people they contact are in that same operator, 7.2% down from last year. From this it can be deduced that people have started valuing other factors - such as packages with multiple services (including any combination from Internet, phone, television and mobile phone), the third main reason cited, at 6.3%.

Naturally, associated to this growth there is an increase in complaints: in the first quarter of 2014, the Portuguese communications regulator ANACOM's most complained-about service was the mobile service [3].

Taking all of this into account, it is clear that the place of mobile networks is more important than it used to be and, therefore, it is necessary for experts to help people process the large

amount of information around them. Every mobile operator is going to look at the data they have and filter it so that they always appear to be the best choice - whether it's 3G or Fourth generation mobile network (4G) access speed, signal strength, or coverage. Looking at the Portuguese case:

- MEO claims they have the best 4G coverage on their website [4];
- NOS also claims they have the best 4G coverage [5];
- Vodafone, however, doesn't claim they have the best 4G coverage; instead, they claim best Second generation mobile network (2G) and 3G coverage [6], as well as the best 4G speed [7].

Consequently, an average user cannot be certain which choice is the best for him if he wants better coverage in his area, and that's one of the motivations behind this project - enabling people to figure out, independently, which is the best network operator for them.

People can generally trust external analysis by market researchers or regulators, like the ANACOM data previously cited, but a more interesting alternative is the ability to give *power to the people* via crowdsourcing information, to allow them to collect data from real, every-day usage phones that reflects real use cases, and to measure the speed and signal strength they have on those phones, so that the data is processed and made available to everyone interested.

On top of that, advantage can be taken of the fact that mobile data needs to be collected to analyze an area that's also directly related to mobile networks, which is Wi-Fi. Even without resorting to hard data, it is easy to tell how popular public Wi-Fi is, just by walking into an average restaurant or café in a big city and seeing just how frequent it is that they have a sign posted somewhere with their Wi-Fi password.

Cisco claims [8] that 70% of mobile users are now using public hotspots, with 57% of those users accessing one at least weekly. Furthermore, while e-mail is the primary communications application used, for 53% of smartphone users, web browsing is the most popular activity in general, for 55% of those users.

It's also important to consider some particular public Wi-Fi networks, such as the international eduroam Wi-Fi network, which is available in 54 countries and allows students, researchers and staff from participating institutions to obtain Internet connectivity not only across campus, but also across the other participating institutions [9]; or the FON access sharing system, which

allows users to share (part of) their network with the public and thereby obtain access to the Internet of other FON users around the world (often with partnerships with a preexisting Internet Service Provider (ISP)) [10].

If the capabilities of smartphone networks are being measured, it seems that Wi-Fi cannot be dissociated from the cell networks - they are complementary, and advantage should be taken of the fact one is being researched to also research the other.

In this project, a system that provides these capabilities is proposed and tested. The data is obtained through an Android smartphone application that measures a number of factors (to be detailed later on in this dissertation) and sends that information to a centralized server, which processes that data and makes parts of it available for public consumption, notably in the form of a web application which represents on a map the mobile coverage information.

1.1 Objectives

The main objective is to develop an application that can collect data from mobile cellular and Wi-Fi networks. From that main objective, the secondary objectives and the requirements of the application can be derived.

Firstly, the application must be able to obtain data relating to the device's Wi-Fi and cellular hardware. The data should be collected, either actively or passively, by it, and stored locally until it can be moved to a server.

The second objective is to be able to move that data from the local storage to a server, and use the server's processing power to collate the data and perform basic analysis on it, such as geographic aggregation.

Finally, the third objective is for the server to be able to display the processed data to the users in an understandable way, since reading raw data is not easy.

Furthermore, we defined a few requirements that it should also fulfill, secondary to those objectives:

- **Making sure the application is transparent**: An important goal to keep in mind is the necessity to make the application **transparent**: whether the source code is publicly available or not, a user needs to be aware of exactly what is being collected and sent - there

should be a way, from the mobile device, to see the data that will be sent to the server.

- **Collecting as much data as we can, even though we may not need it**: Since the users' battery (and time, and goodwill) are already being used, it follows that all the data that may be relevant should be collected. Even if there is no immediate use for it, there is always the potential to keep it in the database for possible future queries.

- **Displaying the data in an easy-to-read way via a mapping web app**: As previously mentioned, it will be necessary to implement a more high-level way to view the data than directly: a web application that shows some form of processed data overlaid on a real world map is the one chosen, as it allows users to get a lot of information at a glance.

- **Guaranteeing that the web app runs well on mobile browsers**: A secondary requirement when designing the web application is to make sure it works well on a mobile browser, as users may want to see the map directly from within our mobile application - even outside of that, the previously mentioned Internet navigation patterns that have more and more people using smartphones for access need to be considered.

- **Granting users alternate methods to access the data**: Finally, it is a good idea to guarantee that the map is not the only way to get data. A possible solution would be to have (part of) the same Application Programming Interface (API) used for the web app to obtain its data available to the public, so that people can query the server in a less abstracted fashion (though not with direct queries to the database).

1.2 Structure

The rest of this dissertation is structured along the following lines: Chapter 2 describes the existing solutions for the problems treated. Chapter 3 has the details on the architectural considerations of this work, while Chapter 4 discusses the actual problems and details of the implementation of that architecture; Chapter 5 is an analysis of the technologies used to accomplish the proposed goals, as well as other issues where a decision had to be made from multiple options; Chapter 6 presents the main conclusions and results of the project.

Chapter 2

Background

The commonplace use of mobile Internet and, specifically, smart devices, makes them a frequent focus of research. This chapter describes various ways of obtaining data of interest to this project, and gives examples of already existing software in the area. It also explains some of the basic terms that will be repeatedly used throughout the dissertation, such as the different network technology parameters.

2.1 Network technology parameters

We are interested in collecting as much information as possible, even if we will not use it all immediately, therefore, a careful study of what parameters are available to be measured both on the cellular network and on the Wi-Fi network side of things is necessary.

2.1.1 Cellular network parameters

This work focuses on four particular technologies, as they are the most common, and the ones explicitly differentiated by Android in their API: Global System for Mobile Communications (GSM) (considered a 2G technology [11]), Code Division Multiple Access (CDMA) (both 2G and 3G, with CDMA2000), Universal Mobile Telecommunications System (UMTS) (3G) and Long Term Evolution (LTE) (4G).

GSM is a standard, developed in Europe, that established itself in the beginning of the 1990s, and is still the most widely used technology around the world [12, Chap. 1]. It forms the basis

for many of the subsequent technologies, such as UMTS and LTE, and is still actively developed to this day, despite its age.

GSM subscribers are identified by their International Mobile Subscriber Identity (IMSI), an internationally unique number that is stored in a physical Subscriber Identity Module (SIM) card. This IMSI includes two sections that store information about the subscriber's network - the Mobile Country Code (MCC) and Mobile Network Code (MNC), and the rest of the number uniquely identifies a user within that network. As the IMSI is internationally unique, it enables the subscriber to utilize their phone abroad, if there is a GSM operator that has an agreement with theirs. GSM frequency bands are very disparate, as the original assigned bandwidth was too small initially (25 MHz) to match growing demand; furthermore, when the technology expanded to North America, the frequency bands used in Europe were already in use by other systems, making the regulators open frequency bands in different frequencies, which made it so that many North American phones could not be used in Europe, and vice versa. The way GSM separates its users is by simultaneously dividing the possible channels into multiple slots both at a frequency and a time level - a user is assigned one of eight timeslots on a specific frequency, similarly to Figure 2.1 [12, Chap. 3.2].

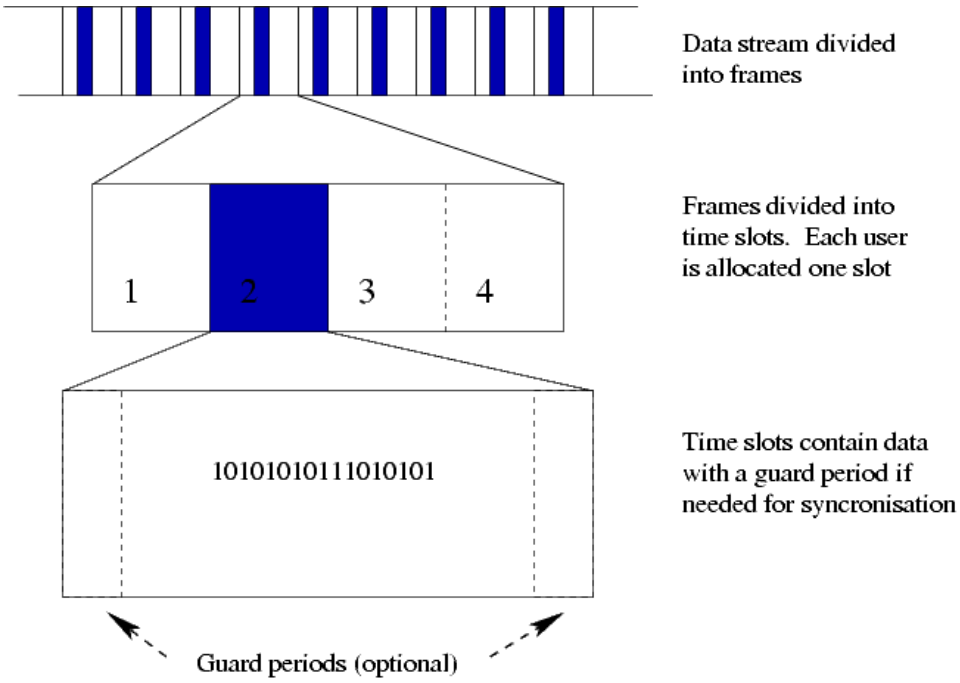

Figure 2.1: Diagram of Time Division Multiple Access's structure, by Mozeratti via Wikimedia Commmons (CC-BY-SA 3.0).

A mobile device needs to maintain the network informed on its location, and it is advantageous

for the communication containing this information to be reduced, as to save power consumption and signaling load on the network. GSM solves this by grouping its cells in what is called location areas - usually, 20 to 30 cells have the same Local Area Code (LAC), and the mobile device does not need to signal the network its new location if it is within the same location area.

Figure 2.2 shows a sample of a GSM cell network - in this case, if a device moved from the area identified by the LAC of 50 and Cell ID (CID) of 1 to the area identified by LAC 50/CID 2, it would not require a location update, but one would be necessary from that area to the one covered by the tower identified by LAC 32/CID 2.

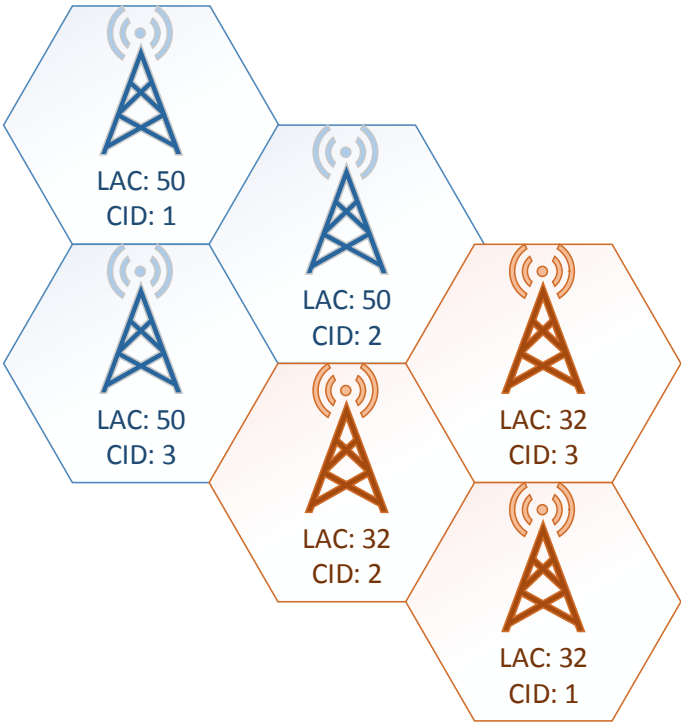

Figure 2.2: Cells in different location areas.

One of GSM's weaknesses is that it was designed for a time where voice calls were the most important service, hence, it was optimized for voice transmission [12, Chap. 2]. However, considering that the importance of the Internet has been increasing since its creation, multiple technologies were developed to enhance the GSM standard and allow it to transport data in an efficient manner - first General Packet Radio Service (GPRS) and later Enhanced Datarates for GSM Evolution (EDGE). Their primary difference is that they function via packet-switching (collecting data in packets before it is sent over the network), as opposed to GSM's circuit-switching, which was optimal for voice, but not for Internet connections.

The successor to GSM and GPRS was UMTS, a 3G system designed to combine packet-switching for data and circuit-switching for voice, reusing many of GSM's systems [12, Chap. 3]. Compared to GSM, UMTS's major improvement in terms of user access is that it replaces GSM's separation of users by frequencies and timeslots by the assignment of a unique code, which, combined with carrier bandwidth increases, allowed much faster data transfers than were previously possible. UMTS has also had improvements throughout the years, such as the introduction of High-Speed Downlink Packet Access (HSDPA) and High-Speed Uplink Packet Access (HSUPA), commonly referred to together as High-Speed Packet Access (HSPA).

Thanks to the code division, frequencies of cells in the same area are often the same in UMTS, this causing the mobile device to require the Primary Scrambling Code (PSC) of all cells it can detect, in order to read their identifying information and decide whether or not it is worth it to switch over to those neighbouring cells - a process called **cell reselection**.

The following evolution of these designs is LTE [12, Chap. 4], which implements an entirely new air interface - while UMTS spreads one signal over the complete carrier bandwidth, LTE transmits the data via many narrowband carriers, splitting the data into many slow data streams that are transmitted simultaneously. This makes the data rate it provides comparable to UMTS, but reduces multipath fading - the effect caused by radio waves bouncing off objects on the way from the transmitter to the receiver, making the receiver get multiple copies of the same signal, arriving at different times. Combined with forcing every LTE device to support Multiple Input Multiple Output (MIMO) transmissions, which allow the transmission of multiple data streams simultaneously, the data rates are beyond those that can be achieved in previous technologies. LTE supports both frequency division (LTE-FDD) and time division (LTE-TDD), although Frequency Divided Duplexing (FDD) is the more common of the two - in the case of time division, an important parameter is timing advance, a value also used by GSM in order to avoid overlapping when communicating in a timeslot-separated channel [12, pp. 34, 226].

LTE isn't just a successor to GSM/UMTS, it's also a successor to CDMA technologies. IS-95A, also referred to as CDMAOne, was a voice-centric 2G network designed in the 1990s as an alternative to GSM [12, Chap. 3.14]. Like GSM, it is circuit-switched, and it was the most common 2G network type in North America. CDMAOne also has a 3G evolution which parallels the relationship between GSM and UMTS - CDMA2000, which was developed in parallel to UMTS and is still the most common 3G technology in North America; however, further enhancements of the CDMA standard were discarded in favor of LTE, which made it

so that LTE had to be backwards compatible not only with UMTS, but also with the latest versions of CDMA.

Depending on the technology used, the way individual cell towers are identified varies: [13]: In GSM (and UMTS), the parameters are MCC, MNC, and LAC, while in CDMA, they are System Identification Number (SID), Network Identification Number (NID), and Billing Identification (BID). However, in terms of functionality, they are similar, as, for the same tower, the BID or CID are typically related numerically, in groups of three, each covering an 120^{o} sector, with the sector number 1 representing North, and 2 and 3 clockwise rotations. In the case of omni-directional antennae, the sector number 0 may be used. The way these sector numbers are used, however, varies - in GSM/UMTS's case, the most common positions to include the number are either the CID's fifth digit from the right (e.g. tens of thousands) or the last digit; in CDMA, since the BID is treated as a hexadecimal number, it may be the least significant digit (e.g. 4441_{16}, 4442_{16}, 4443_{16}) or the third hexadecimal digit from the right, so as to separate related stations by 256_{10} (e.g. 4144_{16}, 4244_{16}, 4344_{16}).

In the case of LTE, however, the unique IDs are different fields, the Tracking Area Code (TAC), Cell Identification (CI) and Physical Cell Identification (PCI). PCI is the least useful of the three for unique identification, as it may be reassigned to avoid interference, although we can obtain from it the sector number, after dividing the PCI by three and adding one to the remainder. CI is a 28 bit number, where the first 20 bits are the eNodeB number, and the last eight are the aforementioned sector number. The last two digits, when expressed hexadecimally, represent a particular antenna in a group, and are attributed in a similar way to the GSM/CDMA systems mentioned above. The TAC can also be used for regional identification purposes - [13] mentions the case of Verizon Wireless utilizing the low order 8 bits of the TAC to encode a *super region* covering a small state or a large city; in this case, they also encode this *super region* in the eNodeB number in the CI, where dividing eNodeB by 1000_{10} returns the *super region* part of the TAC. This partial redundancy of LTE's identification may be used to reject invalid base station IDs that can occur through inaccurate decoding or through asynchronous updating of the components of the antenna's ID in the mobile device's telephony software.

After the identification parameters, another very important parameter - perhaps the most important, from a user perspective - is signal strength. The signal strength values are typically expressed in Decibel-milliwatts (dBm) (logarithmic scale). As in the last case, each technology has their particularities, but the values used are often common to several of them [12]:

- Received Signal Strength Indicator (RSSI) represents the total signal power received in milliwatts. Used in all the network types.

- Received Signal Code Power (RSCP) is the power the pilot channel of a base station is received with. Exclusive to UMTS.

- Reference Signal Received Power (RSRP) is the LTE-exclusive parameter used for deciding whether to perform operations such as handover or cell reselection.

- Reference Signal Received Quality (RSRQ), also LTE-exclusive, is the RSRP divided by the RSSI, and the better this value, the better the signal of the cell can be received compared to the interference generated by the other cells. Expressed in decibel (dB).

These measurements are often reported in Arbitrary Signal Units (ASU) as well, an arbitrary unit that is obtained differently for each technology, detailed in Subsection 4.1.2.

A final relevant parameter is the mobile device's reported operator name, which can be associated with the reported MCC and MNC for categorization purposes.

2.1.2 Wi-Fi technology parameters

Wi-Fi is a lot less variable than cellular technologies, as it is defined by a single standard, 802.11, and all of the variations of that standard differ mostly in their data rate [12, Chap. 6]:

- 802.11b was the first breakthrough for Wi-Fi, and offers a data rate between 1 and 11 Mbit/s.

- 802.11g, backwards compatible with the b standard, achieves data rates up to 54 Mbit/s.

- 802.11a also has the same data rate, but it works in a different frequency band, 5 GHz, as opposed to the 2.4 GHz band used by b/g.

- 802.11n works in both frequency bands and can theoretically support data rates up to 600 Mbit/s, as well as supporting MIMO antennas [14].

- 802.11ac [14] is the latest amendment to the standard, capable of achieving gigabit transmission rates, due to new features such as its multiple-user MIMO antennas.

There are two ways of establishing a Wi-Fi network, namely the ad-hoc, or Independent Basic Service Set (IBSS) mode, where two or more wireless devices communicate directly, and the infrastructure mode, where multiple stations connect to a single access point, forming a Basic Service Set (BSS) - a group of BSS that share the same network name and security credentials is called an Extended Service Set (ESS). The network name is also called an Service Set Identifier (SSID) - usually human-readable, for ease of configuration.

On top of the SSID, when configuring an access point, the frequency, or channel number, also has to be defined. A Wi-Fi channel consists of a subdivision of the valid frequency range (whether 2.4GHz or 5) in 5 MHz slices. In the 2.4GHz band, this corresponds to 11 channels in America and 13 in Europe. When configuring a Wi-Fi network, care should be taken to avoid selecting a channel that is too close to other BSS operating nearby, as a Wireless Local Area Network (WLAN) channel requires a bandwidth of 22 MHz to operate, therefore requiring a separation of at least five channels (except in channel 14's case) to avoid interference (as seen in Figure 2.3).

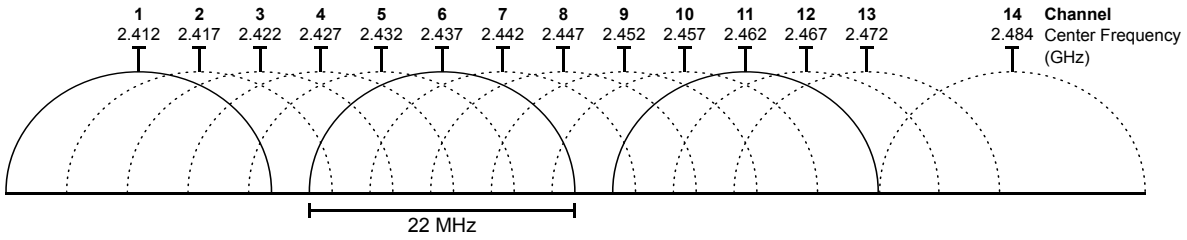

Figure 2.3: Graphical representation of 2.4 GHz band channels overlapping, by Michael Gauthier via Wikimedia Commons (CC-BY-SA 3.0).

Another important Wi-Fi parameter is security. The oldest security standard, available in 802.11a/b/g, is Wired Equivalent Privacy (WEP), which is considered not to provide enough security, as the key can be trivially obtained by an attacker through obtaining enough data from the frames transmitted in the network. [12, p. 349] estimates 5 million frames of information are necessary, though attacks such as [15] succeed in 95% of cases.

Some additional security features were added to the access points that strengthen WEP, such as a Media Access Control (MAC) address filter, which prevents devices that have not been previously authorized by an administrator to connect, or hiding the network's SSID. However, these features are not very secure, as a MAC address filter can be circumvented by manually changing the hacker's device's MAC to an accepted one, and the SSID can be obtained even if

not broadcasted.

To solve these problems, the industry developed the Wireless Protected Access (WPA) standard, which offers an improved authentication scheme and a new encryption algorithm. WPA also has a more secure and more restrictive specification, WPA2, which is backwards-compatible with WPA-only devices [12, Chap. 6.7.2].

WPA has two forms of authentication: Pre-Shared Key (PSK), which uses a pre-shared key in the access point and all client devices, in a similar way to WEP, and Extensible Access Protocol (EAP), which is the name given to all protocols that include communication with external authentication servers.

In terms of encryption, WPA supports two different algorithms, Temporal Key Integrity Protocol (TKIP), designed to replace WEP's weak algorithms, and Counter Mode CBC-MAC Protocol (CCMP), which is mandatory in WPA2.

2.2 Alternative ways to collect data

If a consumer wants to obtain information on mobile network statistics, there are two alternatives: resort to already completed studies (which can be done by entities like market regulators such as ANACOM or by external analyst companies such as Gfk, Marktest [2] or Pew [1]), or performing their own studies.

In the case of market studies, their data can be, at least in part, provided by the operator. In [16] an example can be seen: that's the current form that all mobile service providers in Portugal must submit to ANACOM every trimester, and it includes various types of data, such as the origin of the income from each type of service provided, the amount of subscribers of every type of service (Short Message Service (SMS), Multimedia Messaging Service (MMS), broadband, 3G), or the megabytes transmitted by different Internet services.

Other than that data, ANACOM also uses external information (for example, for population data, or to confront their data with other reports such as Marktest's).

A weakness of industry data, however, is exactly what the provided data means - "If one individual actively uses two SIM connections, that person will be counted by the industry as two mobile connections although he or she is only one mobile subscriber" [17]. This is especially

relevant if you're dealing with data from developing countries, as *device sharing* is a reality there - the survey in [17] found that the number of phones per respondent was lower than the number of SIM cards per respondent, in contrast to a ratio that is closer to 1:1 in developed markets.

However, if you want to perform your own studies, there are several ways to do so. For example, in the aforementioned Marktest's case, they obtain data through phone (mobile and land-line) interviews, taking into account the usual sampling methods (geographic, gender, age, etc.). After gathering these interviews, at a rate of 1250/month, they process the data and release it to their customers [18]. Gfk also utilizes surveys, but they are from retail data instead of interviewing [19]. The Pew Research Center also collects its data via surveys in the aforementioned report [1].

Surveys are not always the most adequate way to handle things, though. In the case of this dissertation, for example, they would not be sufficient, as an objective way of measuring mobile network quality is necessary.

A way to do that would be obtaining manual measurements throughout the area that the consumers are interested in. In 2010, German consultant P3 was hired by the magazine *Exame Informática* [20] to evaluate the quality of 3G networks in the biggest cities of Portugal. Their methodology is relevant to this project, as they were specifically evaluating conditions that common users would encounter: monitoring download and upload speeds, ping, e-mail sending and reception, and web navigation.

It is certainly a correct way to perform measurements, but it is definitely inefficient. In this project's case, as the data should be obtained without specific geographical constraints, it would imply traveling all around the world with smartphones or tablets - definitely not within the realm of plausibility.

A way to counter some of the aforementioned issues is to pass the geographical burden on to the users - if people are collecting data from around the globe, the people designing the application do not have to travel to obtain that data. That kind of strategy is usually called crowdsourcing - "a form of *peer production* that outsources works to a large group of people" [21].

In the 1760s, a Hungarian nobleman build the first machine capable of beating a human at chess, the Turk (Figure 2.4). Although the Turk toured Europe and beat famous people such as Benjamin Franklin and Napoleon, it was not really technology - it had a cabinet that hid the human intelligence moving the machine. Amazon's crowdsourcing marketplace, Mechanical

Turk, was named after this machine, for the way it uses human mental power to solve complex tasks [22].

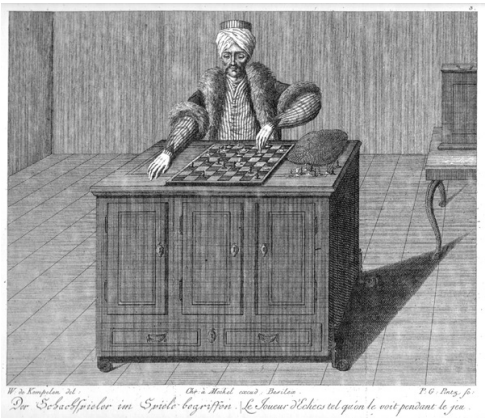

Figure 2.4: The original Turk, represented in a copper engraving by Karl Gottlieb von Windisch, via Wikimedia Commons (public domain).

However, Mechanical Turk functions by providing monetary rewards via micro-payments, which are paid directly by the requester to the users who complete the tasks, and when monetary rewards are involved, things become significantly more complex - [21] shows an example of several strategies that are required to fight problems such as "free-riding", where if the payment is granted before the task is solved, the user can take the payment and provide no effort to solve the task, or "false-reporting", where the requester can refuse payment by lying about the outcome of the task.

The cited paper's primary contribution is to propose incentives based on social norms, notably integrating reputation mechanics to provide better service, e.g., by granting a higher chance to participate in tasks to a worker with a higher reputation.

2.3 Measurement techniques

We surveyed multiple different publications that performed types of measurements with objectives similar to ours, and this section describes their techniques.

A team from the University of Michigan [23] developed a cross-platform application for 3G measurements called *3GTest* that was "executed by more than 30,000 users throughout the world", and which was used to measure real world performance, by resorting to devices that consumers use in the four major cellular carriers in the U.S.

Their technique consists of measuring actual performance (Transmission Control Protocol (TCP) throughput, ping latency, Domain Name System (DNS) lookup delay) that they decided reflected real world usage.

They also reached some conclusions with regards to time of day and signal strength's correlation with 3G quality:

- There are noticeable time of day patterns, for example, Round-trip Time (RTT) varying from 300 ms during late nights to as high as 700 ms during peak times for the AT&T carrier;
- However, the time of day effect is less pronounced for uplink throughput compared to downlink throughput, likely due to higher demand for downlink capacity;
- As it is not easy to control signal strength, they monitored it during a week, and they concluded that the most important thing was the signal strength not being too weak - below a certain value, TCP connections would disconnect;
- Above that threshold, there is a correlation between signal strength and downlink throughput until a specific point, above which additional signal strength does not make a difference.

Many other conclusions from the paper depend on the hardware used, and, as such, are outdated (tests were done with iOS 2.1, Windows Mobile 6.1 and Android 1.6).

A different technique was done in [24], where the used metrics were to measure both TCP goodput (defined as the actual throughput at an application level, i.e., excluding protocol overhead and retransmissions) and signal strength, and the main objective was to measure TCP goodput's correlation with signal strength.

To achieve this, they designed a mobile network measurement platform called Netradar, which is available for a very wide variety of smartphones and tablets (details on the application are further ahead in this chapter).

They propose that in client-server measurements, the server side should not be the bottleneck - in their case, they decided to implement a ticketing system that makes users wait in a queue when the server is over capacity.

Their results analysis is mostly from Finland, but they draw some relevant conclusions:

- There is a strong signal strength/TCP goodput correlation; however, the correlation slows down above 40% signal strength;
- When the phones report greater than 90% signal strength, the TCP goodput tends to be lower. This may be due to having a very high number of measurements for this signal strength, causing secondary effects such as network congestion to be more pronounced;
- The size of the areas chosen for grouping measurements is very important: while GPS has an average accuracy of 52 meters, network positioning has an average of 1125 meters - still, with 500x500m tiles, the standard deviation they found for the average tile size is 15%, thus allowing three separate non-overlapping classes of signal strength to be displayed.

Their most important conclusion is that signal strength measurements are not enough to draw coverage maps by themselves, thus making it important to collect other measurements too.

Another relevant related work is [25], an end-of-degree paper produced by a student of the Universitat Politècnica de València, which talks about two applications: a crowdsourcing one and a 3G measurement one.

The first application is an Android application that interacts with a pre-existing server to register for appointments in several areas (pharmacies, hairdressers, etc.). Since the application has cloud computing features, the most essential feature in the phone is the Internet connection; therefore, it is dependent on the cell phone's coverage.

A way to mitigate those problems is to have a server that contains information related to the mobile network quality in a given location, so that applications can take that quality into account for Quality of Service (QoS) purposes.

The author then developed a crowdsourcing-based mobile application to measure 3G/HSDPA network bandwidth which connects to a server via User Datagram Protocol (UDP).

His methodology consisted of obtaining a packet train with different sizes and calculating the downlink bandwidth. Afterwards, he compared the obtained results with the commercial tool Speedtest.net mobile. Although the 100 byte, 500 byte and 1400 byte packets all performed in a very similar way to Speedtest.net's test, the comparison may be inconclusive, as Speedtest also uses multiple file sizes in its speedtests ("Small binary files are downloaded from the web server to the client to estimate the connection speed (...) Based on this result, one of several file sizes is selected to use for the real download test" [26]).

The data collected by this application is pretty much only that which is necessary for the aforementioned technique to work: device/operator/base station identifiers, location data, and the network connection information (bandwidth and signal strength).

Furthermore, and within the realm of Wi-Fi only, researchers from the University of Michigan designed a solution to measure network performance in large networks, such as corporations and universities, via crowdsourcing [27].

The application they created, MCNet, uses a mobile crowd sensing approach. Rather than relying on technicians carrying out expensive manual measurements, they take advantage of crowdsourcing. Since users have a vested interest in having better Wi-Fi performance, using this approach is likely to have high participation.

It runs on Android devices only, and it has both active and passive components: while metrics such as latency and throughput have to be done actively, they find it is important to also collect data such as connection RSSI and data rate from the device's already registered data, in order to minimize power consumption.

The conclusions to be taken from these techniques are:

- Signal strength alone is not enough, making it a good idea to enable association with other data;
- Measuring real world usage provides accurate results that relate to users' experience, while also being a lot less costly to implement compared to performing manual measurements;
- Bottlenecks should be avoided on the server side;
- When performing speed tests, there is not enough information on whether or not multiple file sizes are an advantage over a single file;
- It is viable to collect data that is already on the device in order to avoid extra power consumption.

2.4 Existing applications

There is a large number of applications already on the market that take measurements from mobile networks. However, the way that they perform them, the way they display them and

even the software they use is interesting to examine for comparative purposes.

2.4.1 Sensorly

Sensorly [28] is a free, crowdsourced coverage mapping service for wireless networks, developed by a French startup in 2010. It collects mobile cellular and Wi-Fi network information, namely related to the geographical location, the mobile terminal and its connections, including (but not limited to) signal quality, terminal device model, Operating System (OS) version, network speed, call logs, and Wi-Fi SSID [29].

It also provides some interesting features, such as mapping the environment for a planned trip, or comparing to neighboring operators.

2.4.2 OpenSignal

OpenSignal [30] is an application designed to create a comprehensive database of cell phone towers, cell phone signal strength readings, and Wi-Fi access points around the world. They claim to collect data from their application that is stripped of any identifying information and is made available online for free, both in a coverage map form and in an API form. The API has the ability to get mobile network data via latitude/longitude in a bounding box or to collect tower information based on the cell id and local area code [31]. It also collects Wi-Fi data, but it is not visible either in the coverage maps or via an API.

2.4.3 4gmark

This tool [32] was designed for quality of service testing, and it allows users to either do a simple speed test or a more complete test that evaluates real use cases (web browsing, YouTube) and compare the results with other users in the same country, zone or with the same smartphone. It also allows them to configure their own custom test.

The data collected by the application is not publicly available, in either API or coverage map form.

2.4.4 Rootmetrics

Rootmetrics [33] is yet another measurement tool, but it differentiates itself from the others seen so far by having a very concrete focus on the United States - on top of a (fairly sparse) international coverage map created via a combination of crowdsourcing and their professional testing, they perform manual, individualized testing throughout all 50 states and publish regular reports on network quality within the USA.

2.4.5 Netradar

The previously mentioned Netradar [34] is the Aalto University-developed system that allows users to measure and share the quality of their mobile Internet connection. There is a coverage map available, although not an API. The application measures only 3G and 4G networks, and it allows the users to utilize Google's login to keep record of their own measurements or browse them.

2.4.6 Ookla Speedtest

On top of being one of the most popular of the speedtest sites [35], speedtest.net has a dedicated mobile application that tests the performance of both cellular and Wi-Fi mobile networks. Its methodology is public [26], as well as the exact type of the data the application collects [36]. It is interesting to note just how much data they collect that is not directly related to the speedtest, for later processing and publishing under their (paid) NetMetrics service.

2.4.7 MobiPerf

Developed mostly in the University of Michigan, by the Robust Net Research Group (same research group as [23]), MobiPerf [37] is an open source application for measuring network performance on mobile platforms. The data is collected either anonymously or from a selected account, which allows users to see their own data. The user credentials collected are not shared outside of this site, and any data used in research projects in universities is anonymized before use.

The source code is available on GitHub, and they allow users to view their previous

measurements if they are logged in, as well as view a coverage map that shows the data collected in the past 48 hours. The (anonymized) data is also available for researchers. Furthermore, this application measures the Radio Resource Control states in the phone, and the implementation of those measurements is detailed in [38].

MobiPerf won two awards from the US Government's Federal Communications Commission in 2011, the Open Internet App Award and the People's Choice App Award [39].

2.4.8 Mozstumbler

The most recent of the applications mentioned (released in October 29th 2014), Mozstumbler [40], is a Mozilla-developed Android application which collects GPS and wireless network data for their crowd-sourced location database (Mozilla Location Service).

The application collects data from Wi-Fi and cellular networks as users move from one location to another, and that data is either used to fill in the coverage map or to improve the precision of areas that have already been filled in.

Mozilla allows you to download data from cell networks [41], but not from Wi-Fi networks, as they may be considered personal data - they also offer an opt-out mechanism for Wi-Fi (hidden SSID or SSID ending with *_nomap*).

Furthermore, Mozilla incorporates data from the OpenCellID [42] project, which "puts a stronger emphasis on public data compared to possible privacy risks, whereas (mozstumbler) has a stronger emphasis on privacy".

2.4.9 Global comparison

The most important conclusion to take from Table 2.1 is that support for all generations of mobile networks is very important - all applications support 3G and 4G, and most support 2G (even if, as in Sensorly's case, they group it with 3G). Also interesting is that every application that uses a map to display results utilizes Google Maps' API and maps, with the sole exception of Mozilla's - Google Maps's popularity is easy to surmise from this table.

Another relevant aspect is the multiplatform capability of most of these applications, which leads to the conclusion that it is very important to carefully study what this dissertation's application should be developed for.

	Wi-Fi	Cellular	Map	Map API	Accessing Data	Source	Android	iOS	Other OS
Sensorly	No	2+3,4G	Yes	Google	No	Closed	2.2+	6+	No
OpenSignal	Yes	2,3,4G	Yes	Google	API	Closed	2.2+	7+	No
4gmark	No	2,3,4G	No	N/A	User's	Closed	2.3.3+	6+	No
RootMetrics	No	2,3,4G	Yes	Google	No	Closed	2.3.3+	7+	No
NetRadar	No	3,4G	Yes	Google	User's	Closed	2.2+	6+	Windows Phone, etc.
speedtest.net	Yes	2,3,4G	No	N/A	User's	Closed	2.2+	6+	Windows Phone
Mobiperf	Yes	2,3,4G	Yes	Google	Global; User's	Open	2.2+	No	No
MozStumbler	Yes	2,3,4G	Yes	OSM	Global	Open	2.3.3+	No	Firefox OS

Table 2.1: A comparison of mobile network measuring applications.

2.5 Chapter Summary

The main purposes of this chapter were to study existing technologies in order to realize what the possibilities are, as well as to figure out where our research can break new ground, and to study different methodologies for obtaining data. The already existing applications share some characteristics between themselves, notably the way a majority of them provide a coverage map and that most of those maps use Google's mapping API. With regards to methodology, it is observable that crowdsourcing seems to be the best way to obtain the data, although it is not without its flaws, such as the importance of incentives.

Chapter 3

Design

This chapter discusses the architecture of the system that will be implemented, as well as each of its components in more detail.

3.1 Overview

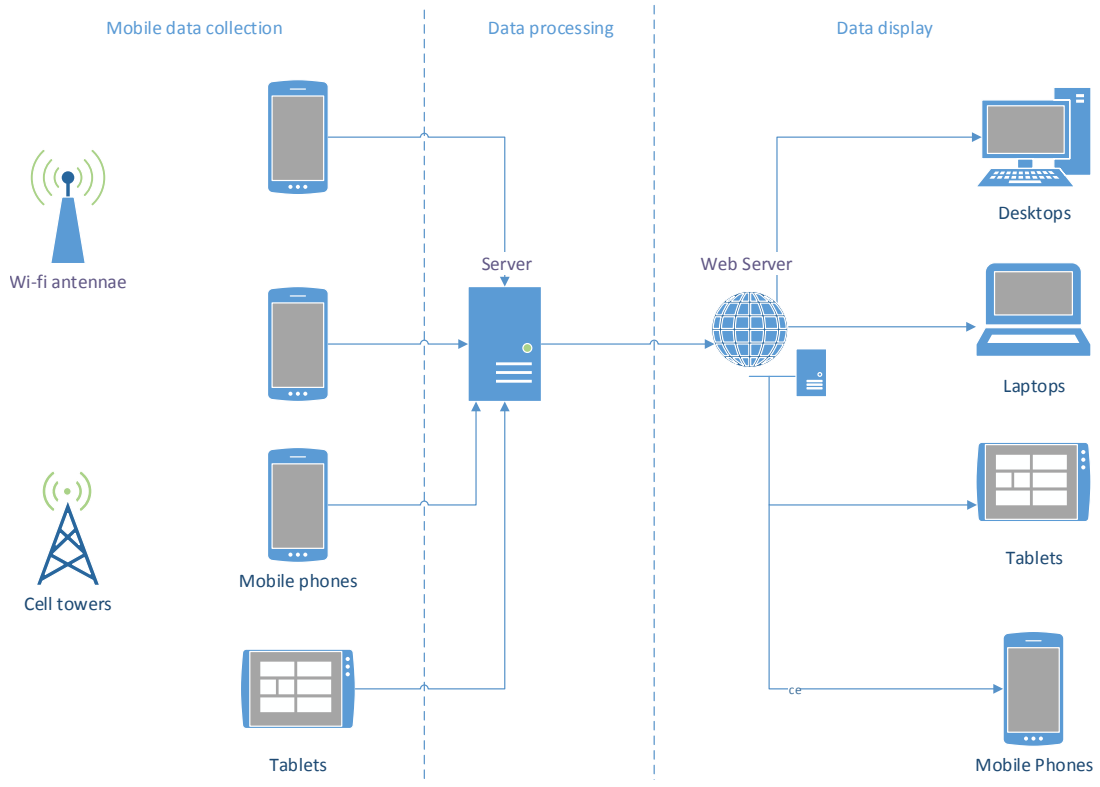

Figure 3.1: General overview of the structure of the planned system.

Figure 3.1 displays the structure of the system, showing the three main components: the crowdsourcing app, running on a mobile device (whether a tablet or a phone) can collect data related to either Wi-Fi access points or cell towers in its vicinity and send it to a server. The server then gathers the data and processes it so as to display it in a Web server. The server's web pages should be viewable and legible in any form factor possible - desktop computers, laptops, smartphones or tablets.

3.2 Mobile data collection

The component of the application that deals with data collection has to collect measurements from the mobile device's Wi-Fi components, from its cellular components, and, when possible, from its sensors in order to enrich the wireless information - e.g. utilizing the barometer, combined with the device's location services, in order to calculate the current altitude level of the mobile device.

On top of obtaining either the cellular or the Wi-Fi data, we couple it with an additional set of parameters that are common to both, which will be referred to as **generic** measurements. These measurements include the latitude/longitude of the mobile device, as well as the aforementioned pressure from the barometer, and identifying information of the device such as its model and manufacturer.

Obtaining the altitude based on the mobile device's barometer requires an Internet connection, as the altitude has to be calculated based on the difference between the current atmospheric pressure and the pressure at sea level [43].

Finally, the mobile application also requires a way to synchronize the measurements with the server. We decided that it was unnecessary to perform synchronization at the same time the measurements were made. Users may want to perform measurements while they only have access to mobile data, and wait to send the measurements to the server when they are utilizing Wi-Fi at home; they may not have a mobile data plan at all, and want to avoid the heavy expenditure data usage takes in those conditions.

In order to perform this delayed synchronization, the application needs to store the measurements in a local database, and then only communicate with the server at the users' command. It might also be relevant to warn users if they have not performed a synchronization in a long time.

3.2.1 Cellular measurements

The cellular measurements can be further divided into two different groups: the general cellular information, which includes the current operator, voice network type, data network type and the signal, and a series of one or more objects containing information on the cells visible by the phone. These cell objects include the location area code and the cell ID of the tower that defines them, coupled with the mobile operator's unique identifier.

The signal included in these measurements is also not a simple observation of the signal level at the instant the measurements are started. It is the result of listening to the mobile device's telephony system's signal level over a short period of time, and then averaging those signals. This is done in order to prevent situations such as the reported signal not being representative of the 'real' signal within the observed area, due to phenomena such as shadowing ("the received signal power fluctuat[ing] due to objects obstructing the propagation path between transmitter and receiver" [44]).

Cellular measurements are only possible to perform in a device with functional telephony; as such, most tablets and phones without a SIM card will not be able to perform them.

3.2.2 Wi-Fi measurements

The Wi-Fi measurements, unlike their cellular counterparts, are just a group of individual objects containing the information of every Wi-Fi network visible from the mobile device. These contain its identifying information (SSID/Basic Service Set Identifier (BSSID)), a signal strength value, and other details on the network such as its encryption scheme and whether or not it supports Wi-Fi Protected Setup (WPS).

On top of this, if the network is currently active (currently connected to on the mobile device), there are a few extra values we can register, such as the link speed and whether or not the network has a hidden SSID.

Finally, the measurements can be performed either instantly or, similarly to the cellular measurements, over a period of time.

3.2.3 Other measurements

The application also includes a speedtest function. This speedtest is performed over the mobile device's network interface - using mobile data, Wi-Fi or Ethernet, depending on which it has got defined as the default route - and it only consists of a download test (upload tests can be added in the future).

This download test can be performed in one of two ways: through a simple Hypertext Transfer Protocol (HTTP) download from a content delivery network (in order to prevent bottlenecking, as well as not having a single point of failure if possible, we decided that it should be avoided to resort to the application's server exclusively), or through a peer-to-peer system, implemented by using a torrent client and measuring its download speed.

3.3 Data processing

After the data is collected, it will have to undergo some processing. This will be performed by a different part of the application, running on a server, that can receive the previously stored measurements from the mobile application, store them in a backend database, and then run a different component on top of that database in order to prepare the data for easy visualization.

3.3.1 Client-Server communication

The communication will be handled via HTTP. As seen in figure 3.2, the server will be listening for input on a given HTTP Uniform Resource Locator (URL), and the mobile application will send data there via an HTTP request - after the request is complete, the server will attempt to process the data and send back a message confirming correct reception. If this message is not received by the client after a short period of time, or if it instead receives a message explicitly stating that the data sent was not correct, it will attempt to send it again the next time the user performs a synchronization task. Otherwise, it will mark the measurement as synchronized and will skip it in the next synchronization attempt.

After receiving these measurements, the server processes them and, according to their type (Wi-Fi or cellular), stores them into the backend database in different tables, which will be described in the next section.

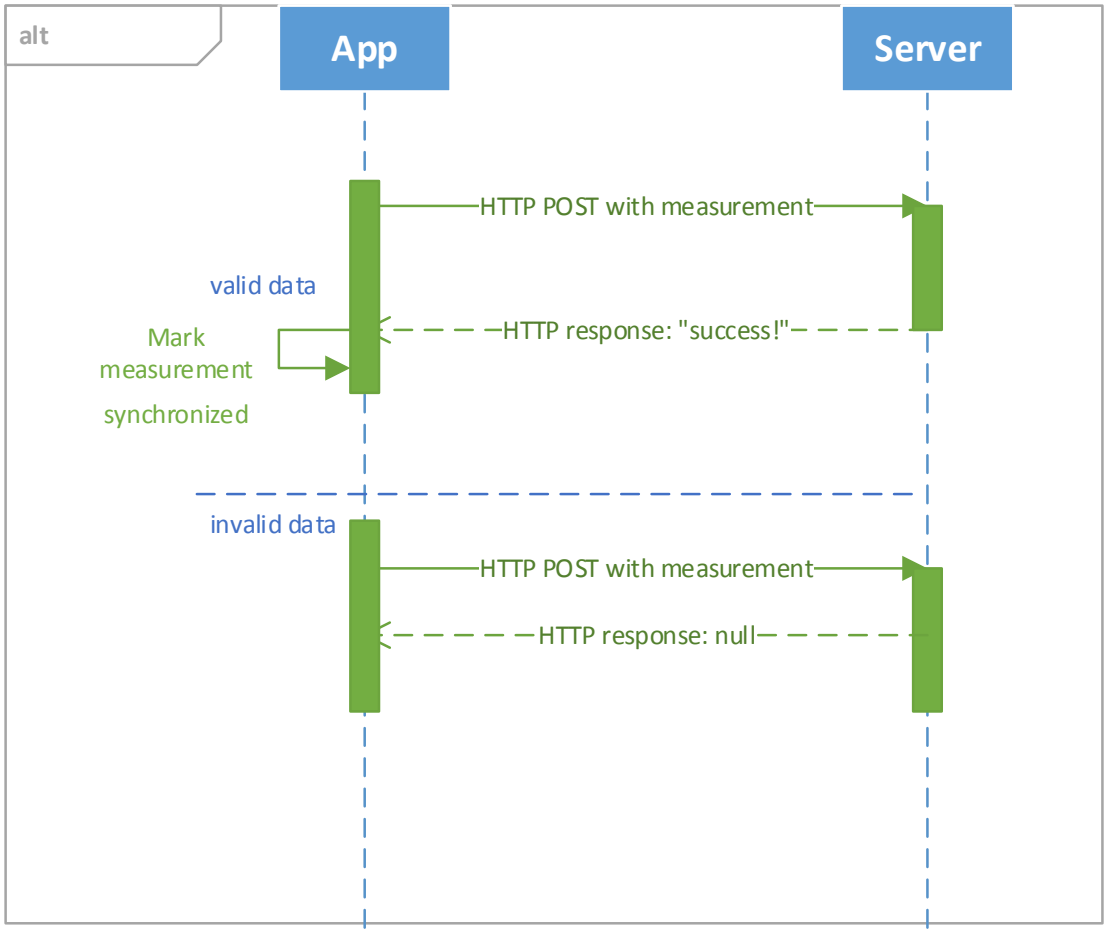

Figure 3.2: UML diagram representing synchronization of a single measurement.

3.3.2 Database design

There are a few requirements the database should fulfill, namely, being able to store the data received from the phone's Wi-Fi and cellular measurements, being able to store external data that we decide to import, and storing the results of our processing of the phone's raw measurements.

Figure 3.3 shows how our database is organized, which this section will describe in more detail. As we can see from the figure, the two major database entities are RawWifiMeasurements and RawCellMeasurements, as it is from these that many of the other entities derive.

RawWifiMeasurements contains the information on an individual Wi-Fi measurement - almost none of it is specific to Wi-Fi, as it is mostly what we defined in section 3.2 as generic measurements, such as the device's data, the geographical position (which may be null), and the time at which the measurement was performed. For each RawWifiMeasurement entry, however,

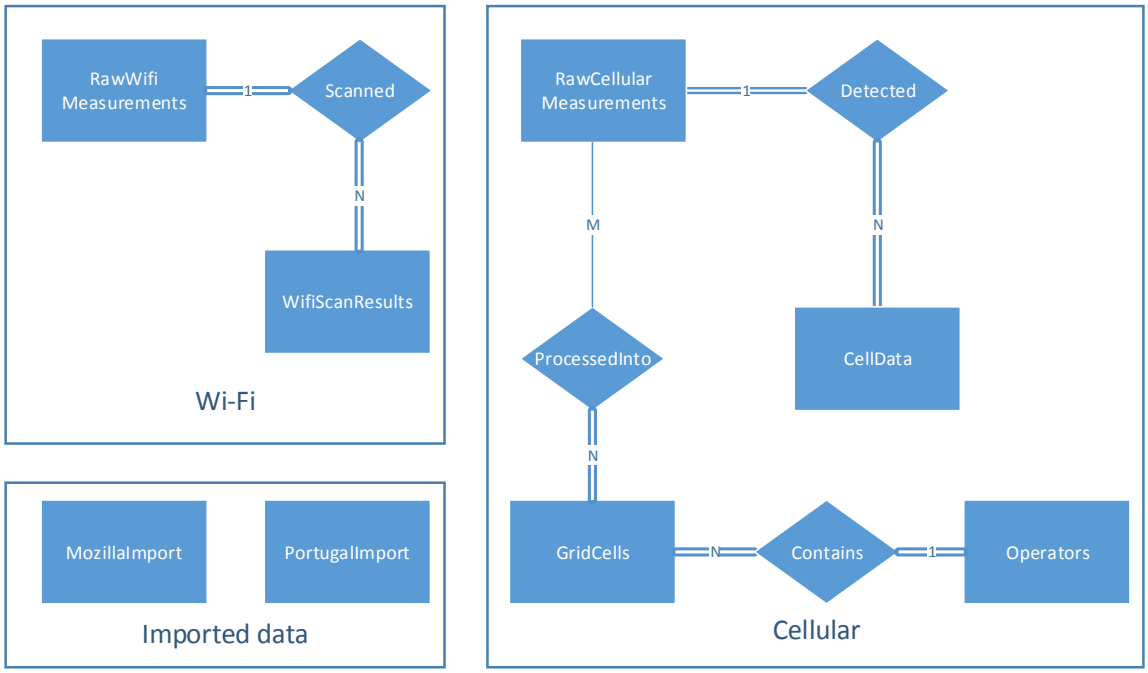

Figure 3.3: Chen's Entity-Relationship diagram representing our database.

there is at least one WifiScanResults entity, which contains the actual details of the Wi-Fi data we collected, such as the SSID, BSSID and signal strength of the detected access points.

As for the cellular measurements, RawCellularMeasurements also contains the generic measurement information, but it has additional, cellular-specific attributes, such as the device's telephony type, network type and operator name and ID. Furthermore, similarly to the raw Wi-Fi measurements, each RawCellularMeasurement entry also has one or more associated entity, in this case, instances of the CellData entity. Every CellData entry consists of information on a single detected tower, containing values such as the tower's MCC, MNC, CID and LAC.

The cellular measurements are also going to be used for drawing the grid of signal strengths that we will describe in the following section. The relationship between these is unique - every grid cell is defined by containing at least one measurement, but measurements may not be in any grid cell, as there is the possibility that the user has turned off the location services on his mobile device, therefore making it impossible to categorize that measurement.

Finally, an auxiliary entity is obtained from the processed grid cells, the Operators entity, which contains information on every mobile operator. For every operator for which we have data, it contains all the different names that were reported by mobile devices, Possible cases of different names for the same operator are when a mobile operator changes its corporate name, or

when it has multiple different brand names under the same MNC. The database will concatenate all of these names, for display purposes.

The two other entities in this diagram represent data which can be imported into our database, namely, the Mozilla Ichnaea tower data in MozillaImport (detailed in subsection 3.3.4) and, in PortugalImport, the geographical data of Portugal's administrative regions which was imported from the GADM administrative areas found in [45].

3.3.3 Visualization processing

After the data is processed into the raw measurements tables, it has to go through an additional step before it is ready for display on the server. We defined that step as the creation of a "grid" covering the whole Earth, where each grid cell corresponds to a subdivision of the Earth's total latitude/longitude.

The grid building starts when the program initializes the grid by either defining a value for the maximum latitude and longitude we want, as well as the size of the grid cell in degrees, or by utilizing our default values, described in Chapter 5. Then, the program goes through all of the raw cell measurements, and, for each one, determines whether or not there is already a grid cell that matches it (defined here as having the same grid position, the same mobile operator ID and the same network class - 2G, 3G or 4G).

The grid position of a measurement is determined by the following formula, applied once for the x axis (latitude) and once for the y axis (longitude):

$$index_x = \lfloor (x + max_x)/total_x * (total_x/delta_x) \rfloor \tag{3.1}$$

where x represents either the latitude or longitude, max_x the maximum value of either of those (necessary, in order to convert from, e.g., in longitude's case, values from -180 to 180 to values from 0 to 360), $total_x$ the total size, in degrees, of the geographic coordinate (e.g. 360 for longitude), $delta_x$ the size of each cell in degrees and $total_x/delta_x$ the number of cells we divided the world in).

- If there is a matching grid cell, the number of samples in that cell is incremented, and the signal strength in that cell is changed - the chosen way to perform this change, detailed below, is an **exponentially weighted moving average** (see Appendix B), where the lambda is at 0.6: every time we get a new measurement, the value of the grid cell's signal

strength is changed to $0.6 * newVal + 0.4 * oldVal$, a value which results in every new measurement having a very large influence over the previous ones.

- Otherwise, there will be a new entry in the table, corresponding to a new grid cell, which stores the cell's geographical position, its index, its signal strength in three different forms (dBm, ASU and a 0-4 level, used to categorize the cell for viewing in the map), its operator ID and reported operator name, and its network class.

After going through all the measurements, the program also stores in a different table all the operators that appeared, and concatenates all the different reported names for each one of them. Finally, it defines the color that this operator's towers will use when being displayed on the map. This color can be either determined programatically, via iteration (each operator from a given country gets the next color in the list), or manually (we know all the available operators in a country and we want the color of the towers on the map to reflect their primary branding color).

3.3.4 External data import/export

Another feature of the program is to import data from external tower databases. We decided to perform imports from the Mozilla Ichnaea API, as all of its cell tower information is available on the Mozilla Location Services page [41], and the description of the data exchange format is also public [46].

Mozilla makes all their data available in a Comma-separated Values (CSV) file containing all of the towers detected up to any given day, as well as smaller *diff* CSV files containing simply the towers that were added in that day.

In the future, it should also be possible to export our data to Mozilla's servers, as their API has a function for data submission [47], as our database was designed from the start to be easily converted into that format.

The data exchange format, that Mozilla created in collaboration with the OpenCellID project, is described in Appendix A (taken from [46]).

3.4 Data display

When all the necessary data is imported and processed, the server is ready to display the received data to the users. This data will be displayed in a web page, although at some point in the future it might be interesting to make it available under a public API as well.

The web page must support the following functions: displaying the grid cells processed in the previous step, displaying a map of the towers that were obtained from importing the Mozilla data and associating those towers with the server's own processed measurements, and displaying a leaderboard with the statistics for the users that performed the most measurements with the mobile application, in order to motivate users to contribute. In the future, it might be interesting to associate the leaderboard with an alias system (e.g., instead of displaying the Android user ID, a user could opt to replace that with a nickname).

The association of the Mozilla towers with the server's measurements is performed in a very simple way: if the user clicks a tower in the map, a box will pop up with details on the tower (cell ID, local area code, operator name...) and the browser will query the server as to whether or not there were any registered measurements in the database. If there are any, a circle containing all the positions where the tower was observed will be displayed on the map in addition to the aforementioned tower detail popup. In cases where the circle isn't large enough to be shown (for example, there is only one measurement where the tower was visible), the program will enlarge the circle - we defined this minimum size as 50 meters of radius, for visibility purposes.

3.5 Chapter summary

In this chapter the design structure behind the three main components of the program to be implemented was described: Firstly, the mobile application that can obtain measurements, both cellular and Wi-Fi, and can also perform a speedtest. Afterwards, the server's processing component, that can both communicate with the mobile application and store and process the data locally, as well as import external data. Finally, its display component that shows the data obtained to the end-user in a more graphical fashion than raw data.

Chapter 4

Implementation

This chapter describes the implementation, detailing the functional aspects of each of its components. A mobile Android application, for the data collection; a Java-based group of servlets and Java applications deployed in a virtual machine, for the data processing; a web server that displays the map utilizing OpenStreetMaps tiles and Leaflet rendering.

4.1 Mobile data collection

The Android application collects data actively via responding to user input, sends it to the server and displays the already-processed map. In this section, we describe the implementation of the data collection function. Details on all of the possible measurements collected, and their values, are in Appendix C.

4.1.1 Generic data

The first step, before the application performs either a cellular or a Wi-Fi measurement, is to obtain a set of generic data. These values are defined as those that are important for geographical information and for identifying information of the device.

First, whenever a measurement is started, the application contacts the Google Location Services API in order to obtain the device's location. If the location on the mobile device is disabled, the data, when sent to the server, will not be included in the cell map. When the contact with the API is finished, the generic information can be obtained, which is done via the

application calling an **AsyncTask**.

This task, **GenericInfoTask**, displayed in Figure 4.1, receives four parameters: the mobile device's location, a pressure sensor listener, the object to place the results in, and a boolean informing it whether or not there is an active network connection. First, the task always starts by adding to the results object some parameters (containing the mobile device's identifying information) from the Android Build class: the manufacturer of the hardware, the user-visible name for it, and its name at a kernel level. Afterwards, the location information is processed: if the location is null, it will not add it to the results; otherwise, it is saved in three separate fields, indicating the latitude, the longitude, and the accuracy of the reported location in meters.

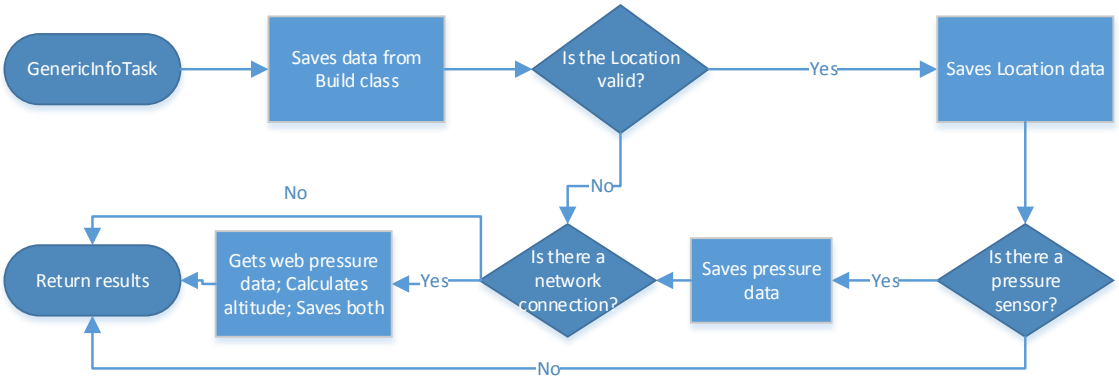

Figure 4.1: How the generic information is obtained via GenericInfoTask.

Finally, it verifies whether or not the received pressure sensor listener is valid - if it is valid, it attempts to calculate the phone's altitude. This is done by a call to the Weather Underground API, following the criteria described in [43], to obtain the pressure in the given location at sea level. With the sea level pressure and the pressure reported by the phone, the altitude is calculated by using **SensorManager.getAltitude()**, which uses the following formula [48]:

$$h = 44330.0 * (1 - (p/p_0)^{1/5.255}) \tag{4.1}$$

where h represents the altitude in meters, p represents the current pressure and p_0 the sea level pressure, both in millibar.

On completion of the task, the class that called it first assigns the user ID [1] to this generic

[1] determined via Android's ANDROID_ID, a parameter constant over the lifetime of the device, which only

information, then saves the generic information to a JavaScript Object Notation (JSON) string, and checks whether the other component (cellular or Wi-Fi measurement) has completed; if so, all the measurements are saved to Structured Query Language (SQL); otherwise, that will be done when the other component completes.

4.1.2 Cellular data

In case we perform a cellular measurement, before any active measurements are actually started, the device resorts to all the already-existing information that was not used in the generic measurements, namely, the phone's reported SIM operator name and unique identification, as well as the current network's reported operator name and ID. These should generally be the same; however, sometimes the SIM card and the network report a different name for the same operator.

The cellular measurements are comprised of information on the mobile device's reported state over a short period of time, and information on all of the cell towers it can detect nearby.

In the first case, that is achieved by extending the Android class **PhoneStateListener**, which is able to listen to specific telephony-related events, such as the cell location, the call state or the data connection state. By using this inheritance mechanism we can override what it does when one of these events occurs, in this case, a signal strength change.

The extended class, **myPhoneStateListener**, when instantiated, obtains the current phone type (GSM or UMTS) and network type, and categorizes the network class depending on the network type (e.g., if the device is using GPRS, it is considered to be using 2G). These values are all saved in a **CellResults** object, together with a timestamp. It also overrides the **onSignalStrengthsChanged()** method so that, when the telephony detects a change in the signal strengths, the signal strength's value is added to a list.

After some time has elapsed (determined by the class calling **myPhoneStateListener**), the listener is stopped. First, the program calculates an average of all the measured values, the standard deviation, and the maximum and minimum values. These may or may not be displayed to the user later, depending on a setting of the application. Afterwards, it returns the signal strength value, both in dBm and ASU, which requires a conversion step:

changes on factory resets or to identify multiple users on multiple-user devices

- In GSM and UMTS, the reported signal strength (RSSI) comes in ASU, so we need to convert it to dBm. The formula is $dBm = 2 * ASU - 113$ [49, 8.5].

- In CDMA, the reported signal strength (RSSI) comes in dBm, so we need to convert it to ASU instead. The formula, taken from [50], is shown in Listing 4.1.

- Finally, in LTE's case, we have the option to choose what we want to obtain. Since the API for LTE signal strength was only added in API level 17, and we want our code to be compatible with a wider range of phones, it is necessary to use a workaround, which consists of using Reflection: if the method with name X exists, call it and return the results [51]. Android provides **getLteRssi()**, **getLteSignalStrength()**, and **getLteRsrp()**, and we chose to use RSRP, both for consistency with Mozilla Ichnaea and for consistency with the cell tower data. RSRP is returned in dBm, and the corresponding ASU is $ASU = dBm + 140$ [49, 8.69].

RSSI	ASU
dBm <= -100	1
-100 < dBm <= -95	2
-95 < dBm <= -90	4
-90 < dBm <= -82	8
-82 < dBm <= -75	16

Table 4.1: How to convert CDMA's received signal strength from dBm to ASU

The other values to be measured, corresponding to the cellular towers, are obtained after the listener stops. This is done by resorting to multiple Android methods, as there is not one single solution to get the data. This process is shown in Figure 4.2.

- **getAllCellInfo()** returns all the observed information from all the radios on the device, including the current and the neighboring cells, and it is the recommended solution; however, it was only added in API level 17, or Android 4.2, and thus it cannot always be relied on by itself. Furthermore, even if it exists on the phone, it may return a null object, particularly in the case of older devices;

- **getNeighboringCellInfo()** returns information on only the neighboring cells. It is planned to be deprecated in the future, and Google suggests only using it in the case of **getAllCellInfo()** returning null.

- **getCellLocation()** (which will also be deprecated in the future) has to be called if we use **getNeighboringCellInfo()**, as it obtains data on the current cell and completes the information given by it, if combined with the information from **myPhoneStateListener**. It is not compatible with LTE, which should, generally, not be a problem, as LTE devices are mostly on higher API levels and should be compatible with **getAllCellInfo()**.

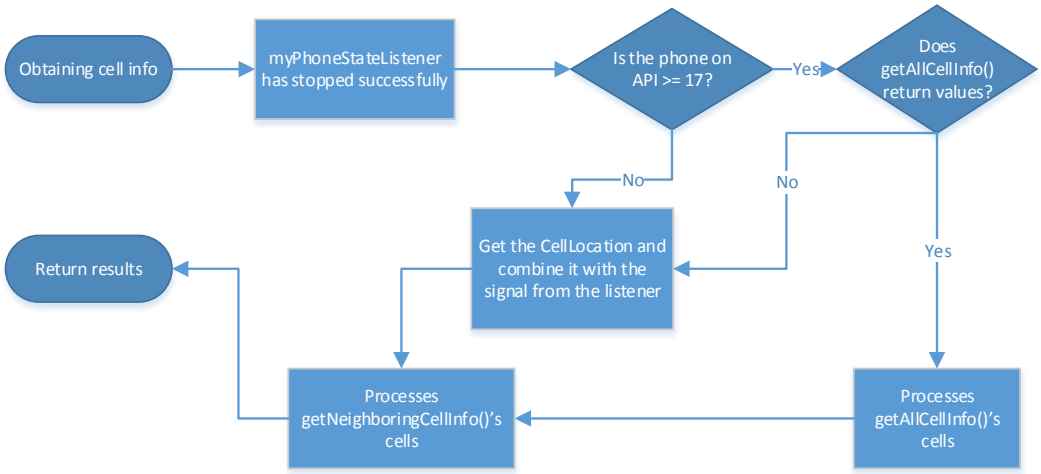

Figure 4.2: How the cell tower info is obtained

The program first tries to access **getNeighboringCellInfo()**, then **getAllCellInfo()**. In the case **getAllCellInfo()** returns null or is not accessible due to the device's API level, it creates a new **CellData** object representing the current cell; otherwise, it processes the cells given in **getAllCellInfo()**, and creates a **CellData** object for each. In either case, it also processes the list given in **getNeighboringCellInfo()**, as there is no reason not to get it, even if the detected towers may be redundant. This redundancy may allow us to get extra data that may provide insights into parts of the program that are not working correctly, or avoid potential issues with poor implementations on the hardware level of one of the used methods.

After getting the **CellData** list, the information is saved to a JSON string, and the program checks whether or not the generic measurements have finished; if so, all the measurements are saved to SQL; if not, that will be done when the generic measurement task completes.

4.1.3 Wi-Fi data

When the user selects to perform a Wi-Fi measurement, the application begins by verifying whether or not Wi-Fi is enabled. If it is disabled, the program enables it and waits until the Wi-Fi hardware is ready, so as to begin measuring.

Afterwards, it obtains information on the current Wi-Fi connection, if the mobile device is connected to any. This is necessary to do because there is some information one can only obtain from the current network, namely, the link speed and whether the SSID is hidden or being actively broadcasted.

After this setup process, one of two things can happen, depending on the user's settings: by default, a single scan will be performed, but if the user so wishes, the application can perform multiple scans, for a short period of time (as of right now, six seconds, as that is the longest time a tested device took between two scans).

Each of these solutions is implemented using an extended **BroadcastReceiver** class. The only significant difference is that for every scan that the **IterativeWifiReceiver** gets, it adds the signal to a list per access point, then calculates the average signal for each access point in the end, whereas in the regular **WifiReceiver**, the signal is a one-time, final measurement, as shown in 4.3.

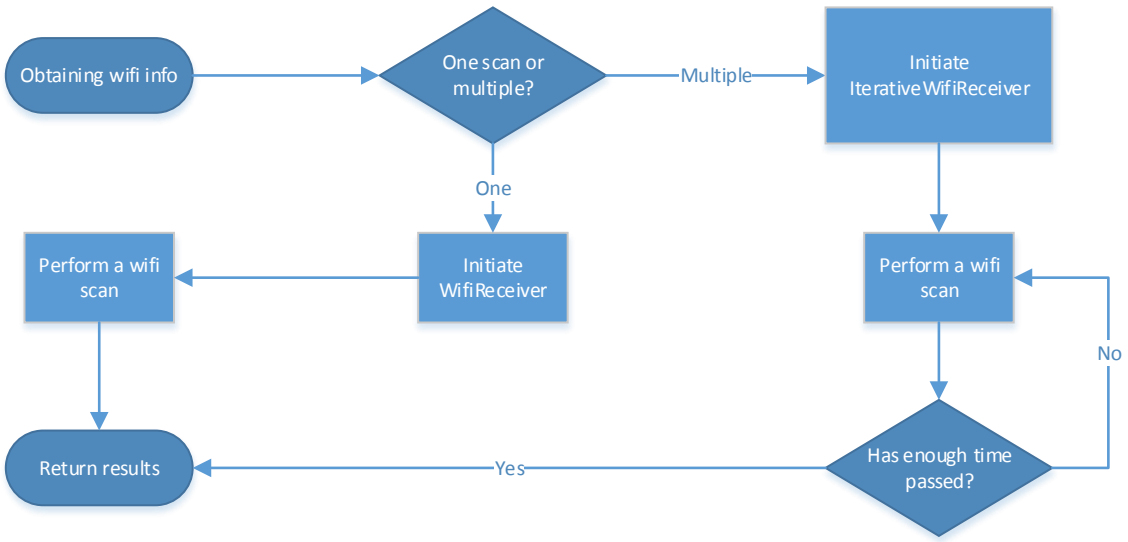

Figure 4.3: How the Wi-Fi scans are performed

As in the cellular measurements, when the scan(s) complete, the information is only saved to SQL after the generic measurements are also completed.

4.1.4 Connection speed

Besides the two primary measurement options, the mobile application also has the ability to perform speed tests. Due to lack of time, the implementation of the speedtest exclusively measures the downlink speed. The speedtest can be performed in one of two ways: via a direct HTTP connection, or utilizing torrents.

The regular speedtest utilizes a file hosted in a content delivery network as its test target, as mentioned in Chapter 3. It then begins an AsyncTask (the main loop of which is shown in Listing 4.1), which, in a separate thread, performs the download. For every 1000 downloaded bytes, the task sends a message to the main thread to update the current download speed, then registers the fastest downloaded block so far (blocks are defined as 1/50th of the total download limit we want). This task ends when it either passes the download limit defined (by default, 20 MB) or it exceeds the maximum alloted time (by default, 1 minute). When the task ends, the UI is updated with the total download speed and the speed of the fastest downloaded block.

```
while ((currentByte = in.read()) != -1) {
   totalBytes++;
   currentblock++;
   if ((totalBytes%1000 == 0)) {
      publishProgress();
      if (currentblock > blockSize){
         double speed = currentblock /
               (System.currentTimeMillis() - blockStart);
         if (speed > fastestBlock) fastestBlock = speed;

         currentblock = 0;
         blockStart = System.currentTimeMillis();
      }
      if (totalBytes>=BYTE_LIMIT_THRESHOLD)
         handler.post(taskCanceler);
   }
}
downloadTime = System.currentTimeMillis() - start;
return ((double)  totalBytes / downloadTime);
```

Listing 4.1: Simplified excerpt of regular speedtest code's main loop

The torrent-based speedtest, which utilizes the Java library ttorrent [52], has very similar logic to the regular speedtest. The primary difference is that it first utilizes Android's default DownloadManager to get a torrent file that is hosted on our server (this makes it easy to swap out the torrent file without pushing an update to the application). Then, it begins an AsyncTask which gets the local path to that torrent as an argument and starts up a ttorrent Client to start the download (described in Listing 4.2).

```
while(client.getState()!= Client.ClientState.DONE){
   long newBytes = client.getTorrent().getDownloaded();
   currentblock=currentblock + (newBytes - totalBytes);
   if (currentblock > blockSize){
      double speed = currentblock /
            (System.currentTimeMillis() - blockStart);
      if (speed > fastestBlock) fastestBlock = speed;

      currentblock = 0;
      blockStart = System.currentTimeMillis();
   }

   totalBytes = newBytes;
   downloadTime = System.currentTimeMillis() - start;
   publishProgress((double) totalBytes / downloadTime);
   try {
      if (totalBytes>=BYTE_LIMIT_THRESHOLD)
         handler.post(taskCanceler);
      Thread.sleep(2000);
   } catch (InterruptedException e) {
      return (double) totalBytes / downloadTime;
   }
}
```

Listing 4.2: Simplified excerpt of torrent-based speedtest code's main loop

Utilizing this library required some effort, as it is not native to Android. Notably, it was necessary to compile it manually, as Android includes an outdated version (v1.3) of *apache-commons-codec* as an internal library, therefore making it impossible for an application to use a more recent version [53]. Compiling ttorrent manually allowed us to make it refer to a renamed version of *apache-commons-codec*, which solves this problem.

It was also necessary to replace an Exception the ttorrent Client throws when it tries to download a piece that has already been downloaded, as the thrown Exceptions made Android force close our measurement application.

As in the previous case, the program keeps updating the UI. However, the updates are done once every two seconds instead of once every block, as the thread the task is on is not the same as the thread the torrent client executes on, and the loop is time-based. Also like the previous case, the task ends when it either exceeds the download or the time thresholds.

This particular test was only performed in Wi-Fi networks. It should work with cellular networks as well, but the ports or the torrent protocol itself may be blocked by the network operators, in which case the test will return a total speed of 0 kB/s.

Finally, due to lack of time, the speedtest data is not currently sent to the server. In the future, it should be possible for the application to associate a few identifying values (such as network type, location and mobile device ID) and to synchronize them with the server. This would allow, for example, the drawing of an extra layer on the mobile coverage map displaying the network speed instead of only having the signal strength.

4.2 Data processing

For implementation of the data processing component we resort to servlets, small Java program modules designed for web servers, deployed in a Jetty application server and started by the server whenever necessary, and accompanied by a few pure Java files deployed separately in the same machine, executed periodically. The machine it was tested on, where the data display portion of the application was also deployed, is a virtual machine with a single-core AMD Opteron 63xx CPU and 1 gigabyte of RAM. Figure 4.4 shows the structure of these files: everything on the left side of the divider is discussed in this section, while the other servlets are in the next.

4.2.1 Client-server communication

The communication, on the client side, is handled by a class that calls an **AsyncTask** which goes through all the measurements stored in the local database and, for the ones that were not sent to the server yet, attempts to send them via HTTP POST to a particular URL on the server. As seen in the previous chapter's Figure 3.2, measurements are only synchronized once - after sending the measurement in JSON, if it receives a confirmation, a *hasSynchronized* boolean in the measurement's local database entry is set to true, so that in the next synchronization attempt it is skipped over.

On the server side, there is a servlet, **AndroidJSONServlet**, that receives input over HTTP and attempts to parse it as a JSON object with a defined structure (it has to contain the fields we defined for generic/cell/Wi-Fi measurements). If the parsing succeeds, the program attempts to process it - first, it detects which kind of measurement it is, then attempts to validate the data

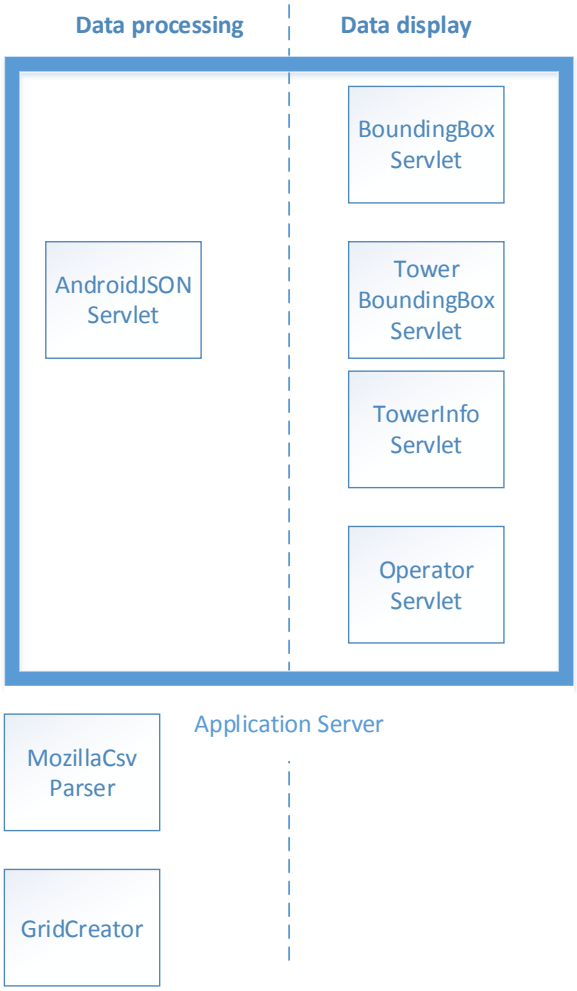

Figure 4.4: List of Java applications on the server, including data processing and data display servlets, and data processing external components.

and insert it into the correct raw measurements table in the backend database. If the process is successful, the servlet sends an HTTP Response informing the client that the measurement has been successfully synchronized.

4.2.2 Visualization processing

The grid creation process follows the design we described in 3.3.3: The program goes through all our measurements and either creates a new grid cell or increments an already existing one. Since the grid creation was implemented as a normal Java application (**GridCreator.java**), it was deployed in the server as a .JAR. This file will be run to recreate the grid with a relatively low frequency, to avoid too much server load. Once a day should be enough in order to achieve a

balance between server load and users being able to see their data collection reflected on the server, although, in this starting period where we don't have too many measurements, we could opt for faster updates without worsening performance.

An implementation detail is how measurements are categorized in order to display them in the server. Ideally, there would be a gradient of colors representing signal quality, but for simplicity, we defined a four-color solution, inspired by how mobile devices generally show their signal levels in a 4-bar graph, each color representing a signal strength 'level'.

The definition of the levels for each of the mobile technologies was taken from how Android categorizes them for the aforementioned 4-bar signal graph, and is displayed in Table 4.2.

	GSM/UMTS	CDMA	LTE
None/Unknown	ASU <= 2 / ASU = 99	dBm <= -100	-140 <= dBm < -115
Poor	2 < ASU < 5	-100 <= dBm < -95	-115 <= dBm < -105
Moderate	5 <= ASU < 8	-95 <= dBm < -85	-105 <= dBm < -95
Good	8 <= ASU < 12	-85 <= dBm < -75	-95 <= dBm < -85
Great	ASU >= 12	dBm >= -75	dBm >= -85

Table 4.2: Classification of signal strengths

The other element of visualization processing performed on the server is the data import from the Mozilla Ichnaea project. It was implemented in **MozillaCsvParser.java**, a Java application that reads CSV files in the Mozilla Ichnaea standard, and inserts them one by one in the database. The current implementation only supports complete files containing all the data about a given tower, but in the future, it will also support updating already existing towers, allowing us to use the much smaller *diff* files provided by Mozilla. The reason this feature has not been implemented yet is the lack of the mechanism to do so (the UPSERT, or ON CONFLICT DO UPDATE), which will only be supported in a future version of PostgreSQL [54].

4.3 Data display

This section describes the display of the data. Visualization is implemented via a web page deployed on a web server, which uses JavaScript to communicate via Asynchronous JavaScript and XML (AJAX) with the servlets displayed in Figure 4.4 and obtain data from the processed backend database.

4.3.1 Web server display

The web server has two different sections: a map of the grid cells and a map of cell towers. Moreover, it has two auxiliary web pages, containing a leaderboard and an About page describing what the website can do and why it was created.

The grid cell map contains rectangles with multiple colors (each rectangle representing a grid cell), an overlay that allows users to toggle between showing a single operator or all of them, and checkboxes to enable each of the 2G, 3G and 4G layers, as seen in Figure 4.5. The ideal use case for the map is for the user to only select one network operator and one network class, since that will give the least 'polluted' information. Any other combination will cause cell overlapping.

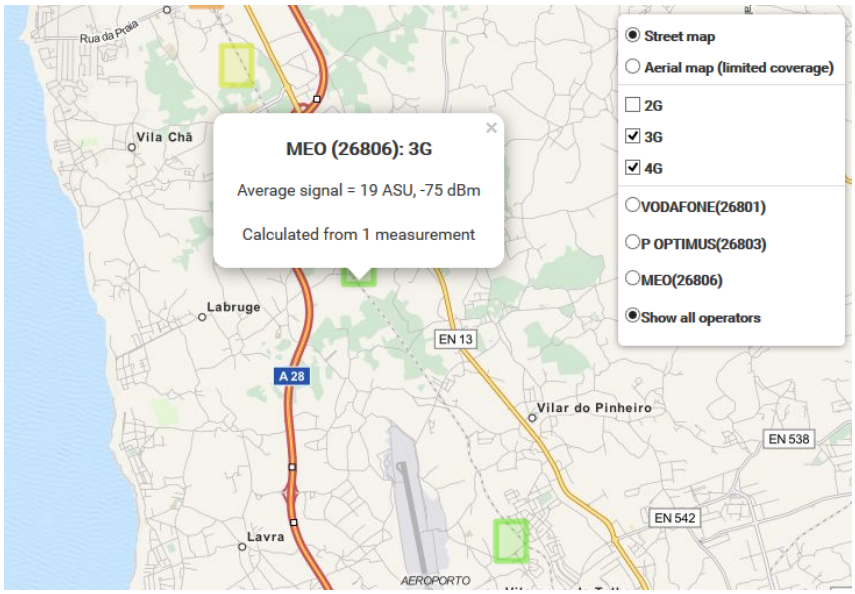

Figure 4.5: A sample of the grid cells map.

The way this map is implemented is having the Leaflet.js library perform a call, via AJAX, to a servlet which returns information on all the grid cells within the current bounds of the screen. After the servlet returns all the cells, the client goes through them all, determines the information that will show up when the user clicks on them (also in Figure 4.5), determines what color to display based on the signal level (see Table 4.2), and puts them into the list that matches the network class of the cell. Furthermore, it also filters the results by the operator currently selected by the user (or does not filter at all, if "Show all operators" is selected in the radio buttons).

The weakness of this implementation strategy, however, is that, for example, even if a user

only wants to see the data for a given operator, the server will have to reply with all of them, which may become costly with a large number of users. A possible solution is to instead make a new call to the server every time the user changes their desired operator or desired network class. However, this solution has instead the problem of increasing both delay and the number of connections to the server. More study is required to determine the correct methodology.

The tower map uses the previously imported Mozilla Ichnaea tower list in order to present cellular tower locations. The implementation is very similar, as there is a lot of code reuse, but there are a few differences: The AJAX call returns a list of towers from the Mozilla database which are within the map's bounds, then displays all of them on it, with no filtering implemented yet. Afterwards, if the user clicks on one of the towers, a second servlet is contacted, which reads our cell tower database in order to see if we also have data for that tower. If so, a circle containing the positions from where that tower was observed is displayed on the map, as show in Figure 4.6. If not, it just displays some information about the tower, namely its Cell ID/Local Area Code, the operator it belongs to, and whether the position is an estimate or an authoritative position (i.e., if it was added to Mozilla Ichnaea from an external database).

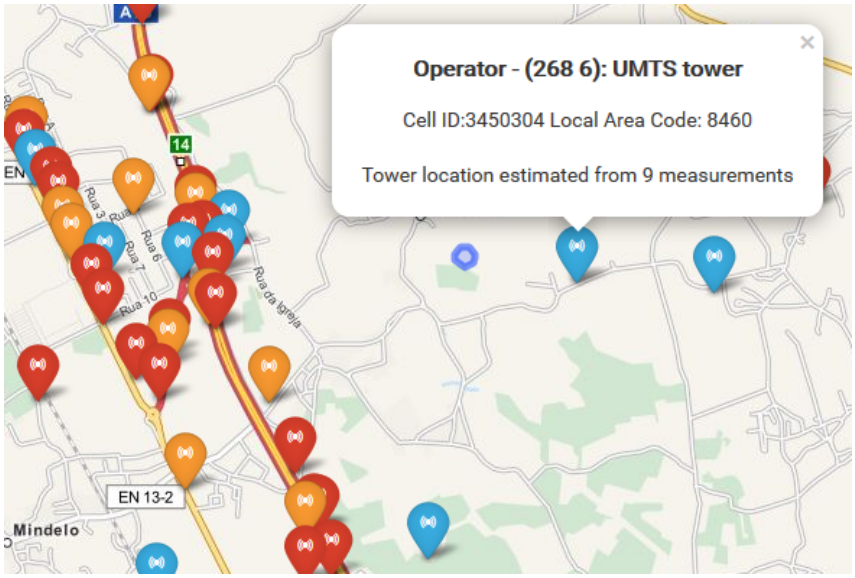

Figure 4.6: A sample of the cell tower map.

A significant weakness of this tower map at first was the sheer dimension of the Mozilla database - the imported database contains more than eight million entries, which makes it impractical to query whenever the map scrolls. A simple bounding box query in the center of Porto took, during testing, 12 seconds. The solution we chose was to split the Mozilla database into 'slices'. We began testing by utilizing one of those slices with the latitude from -7 to -9

(containing the entirety of continental Portugal), and the same sample query takes 90 ms instead.

In the future, different scaling options could be explored, such as splitting these slices between different machines, or verifying whether or not a more powerful machine could be enough to significantly reduce the delay.

4.3.2 Android-side display

For the implemented website to be compatible with all the different devices that will view it, we decided to make the site identical on every platform, but allow for some parameters to shift so as to make it readable.

Notably, in the grid cells map, the control layer was too large to view correctly, as it hid a large part of the map, especially on smaller devices. However, in large monitors, having it show up as a small square that the user would have to click in order for it to expand made it difficult to use. The decision was made, instead, to make the control layer start automatically hidden in smaller resolutions (below 800px horizontal or 600px vertical). This, combined with taking care to avoid absolute sizes while writing the Cascading Style Sheet (CSS), resulted in a website that is readable no matter what form factor you are reading it on.

The mobile application includes an Activity that opens an Android WebView directly to the page. It behaves like a regular browser, with the exception that the user cannot navigate outside our website and the links within it; furthermore, it simplifies execution, as we know that the user has already given us permission to access their location by installing/executing the application. Therefore, geolocation can be done immediately, provided that the device's GPS is enabled, instead of requiring user confirmation.

4.4 Chapter summary

In this chapter we described the implementation of the three main components of the program to be implemented: Firstly, the way the mobile application performs measurements, generic, cellular and Wi-Fi, and how it performs speed tests. Afterwards, the server's processing component, both the way it communicates with the mobile application using a servlet and the way it prepares the data for visualization. Finally, its display component, both the way the map is displayed and the specificities of the way it is viewed on different devices.

Chapter 5

Discussion

This chapter describes the different alternatives that were available for each part of this dissertation, as well as the one decided upon and why, not just in terms of technological alternatives but also other decisions that involved considering multiple possible choices.

5.1 Dedicated measurement hardware

First, we considered mobile technology testbeds, such as the Rohde&Schwarz CMW500 [55]. This platform is a universal testbed for wireless technologies, supporting cellular technologies such as GSM or LTE, satellite ones like GPS, or wireless connectivity ones including 802.11 a/b/g/n and Bluetooth.

This device's characteristics are interesting; however, its price makes it completely out of the reach of this work, because prices, even for used ones, are the order of the tens of thousands of dollars. Furthermore, it would restrict the measurements to being physically performed everywhere, which, as shown before, is not the best option for this project.

An affordable hardware alternative would be to utilize a device based on GNU Radio. GNU Radio [56] is a development toolkit for implementing signal processing/radio applications. It is GNU General Public License (GPL)-based, and some of the examples of developed applications are spectrum analysers or basic signal generators.

A Universal Software Radio Peripheral (USRP) is a hardware board designed to offer a simple, cheap way to create radio-based applications. USRPs are frequently utilized in combination with

GNU Radio to create complex radio systems [57]. USRPs are cheaper than the testbeds, but they are still fairly expensive and have the same geographical limitations [58].

An example of a project that can use a combination of these two technologies to create a Software-Defined Radio (SDR) system is the OpenBTS project [59], which is an open-source application that uses SDR devices such as USRPs to present GSM handsets with an entirely open-source backend.

There are many other software-defined radio projects. The most relevant one is the rtl-sdr [60]: a way to take advantage of TV tuners based off a specific Realtek chip (the RTL2832U) to provide SDR features at reasonable prices.

A few related projects done with rtl-sdr radios are an LTE scanner, an open source implementation of the LTE standards, or a study [61] which managed to crack GSM via capturing communications with one of these devices. This class of devices is definitely the most interesting of those seen, and a possiblity is open for future research in the area, although it remains difficult to match with this project's objectives due to the geographical constraints.

5.2 Mobile Operating System

After assessing the dedicated hardware solutions, the different mobile OSs were studied: for a crowdsourcing solution, it would be particularly important to consider market penetration, ease of development, and ease of deployment of the application on the devices.

Android was the first operating system to be considered, as the author has some experience with it and with the language it is based on, Java. Moreover, it is easier to find a wide variety of devices to test it on. In terms of deployment, it supports manual installation of the application package file (for example, over the Android Debug Bridge [62]), and even publishing in the Google Play store is accessible: all you need to do is pay a one-time $25 USD registration fee [63]. With regards to development, the language required to program in Android is a slightly different version of Java, that runs on top of a custom virtual machine [64].

In terms of market penetration, [65] indicates that Android dominates the basic phone segment (where the average phone costs $100 USD), not being as strong in the premium phone segment (average of $447 USD). However, their worldwide device shipment statistics present a dominant position: Android is expected to ship 1,454,760 (59%) thousands of units in 2015, compared to

iOS/Mac OS's 279,415 (11%) and Windows's 355,035 (14%). Although these statistics include PCs, and not just mobile devices, it is clear that Android has a wider install base from which to obtain crowdsourced data than the other platforms.

Another feature that Google provides is a Dashboard [66] that gives information on the devices that are active in the Google Play store. The dashboard has a platform versions table that displays the amount of Android devices on any given version of the system, as well as a screen size / screen density graph, which is important for developers who wish to provide different versions of their application for different screen sizes.

Other than Android, iOS was also briefly looked into. On top of the author not having any experience with iOS and any of the possible development languages (such as Swift [67] and Objective-C), it is necessary to pay an annual fee of $99 USD, and to have an "Intel-based Mac running OS X 10.8 Mountain Lion or later" [68]. Furthermore, essential functions for this work, such as manually requesting new Wi-Fi scans, have been made impossible, or at least very difficult, by Apple [69].

5.3 Mapping tools

There are multiple different tools to select from, for the purposes of drawing the map: on top of picking a base layer for the map, the API used to display that map has to be chosen, and also it is necessary to choose where and how to obtain the tiles for that API to display.

In terms of the map itself, the leading option is Google Maps. As was shown in Section 2.4, most existing applications with similar requirements use it. However, there is a downside to using Google Maps, as it has relatively strict limits on the usage. After 25,000 map loads per day for a period of more than 90 days, you have to start paying Google [70] to continue using the service. Even so, Google Maps solves the three problems, since it provides the map layers, mapping API and tiles.

An alternative to Google's maps is OpenStreetMap (OSM), an editable map of the whole world built on mapping contributions from the community [71]. The fundamental difference from Google is that while Google's API and data may be free to use (*gratis*), it is sourced from private companies, and, as such, it is copyrighted (not *libre*). Instead, OSM provides programmers, cartographers or social activists with data which can be reused without creating a "derived

work" [72], which would make these users subject to the copyright conditions of the original map.

There are also other commercial options, but they suffer from the same problems as Google Maps does, such as Microsoft's Bing Maps (limit of 125,000 map loads per year). In the case of Microsoft, however, they do have an education license available, which may be interesting to consider as an alternative in the future [73].

Even after opting for OSM, we have to be aware that, as a free service, the OSM tile server cannot be used directly for large volumes of data. As such, we have to obtain the actual rendered tiles of the map elsewhere [74].

There are several ways to use the OSM tiles, which we can divide into two types of solutions: downloading them and making your own tile server that renders the tiles based on their database, or resorting to an already existing server such as MapQuest Open or MapBox, both of which provide these services (with slightly modified tiles). We opted for MapQuest Open, as MapBox has usage limits, even if MapBox's payment plan has scaling costs (designed for increases in the plan as your application's number of users grow), and their map has a higher visual quality.

Finally, there is the question of the API used to display the map. Commercial solutions like Google Maps provide their own API. For OSM, there are two major open source alternatives [74]: OpenLayers, the oldest of the two and the most well supported, and Leaflet, a more recent, lightweight library.

We chose Leaflet because of its lightweight size (the total file size is in the kilobytes versus OpenLayers' megabytes, which is particularly important for mobile connections), as well as its flexibility: being designed with community-developed plugins as a focus makes it very powerful. For example, there are plugins that implement features such as automatically requesting to the SQL server only the points of data within the user's browser's viewport [75], or freehand drawing on the map [76].

5.4 Geographical databases

A regular Database Management System (DBMS) can deal very well with simple data such as numbers, addresses, names or descriptions. In order to obtain that efficiency, even with very large databases, DBMS utilize indexation to reduce the scope of the search to perform. Database indexes are similar to indexes of any given book - they have a key and a pointer to the section of

the database where the corresponding data is available [77, p. 496].

However, in case we are asking a question as simple as *list all the customers that live less than 50 km from store X*, that question will have to transform their addresses into a reference system (such as latitude/longitude), calculate the distance between the store and the customers, and if the distance is less than 50 km, report back with their names. A standard DBMS cannot deal with multi-dimensional coordinate data. This led to the creation of a dedicated type of database to do this [78, Chap. 1.1] - an Spatial Database Management System (SDBMS).

An SDBMS is a software module that can interface with a preexisting DBMS and that supports spatial versions of components of the DBMS, such as spatial data types (polygons, points) and a query language that can interact with these new data types. It also handles spatial indexation, more efficient algorithms for spatial operations, and specific optimization rules [79, Chap. 1].

A Geographical Information System (GIS) is a software tool designed to visualize and analyze spatial data through spatial analysis functions such as shortest path analysis, adjacency calculus or regional searches. A GIS typically utilizes SDBMS to store and search large sets of geographical data, therefore increasing its efficiency.

The difference between the two can be summed up in the fact that GIS offer a large set of operations on a small set of objects, while SDBMS allow us to perform simple queries on very vast sets of objects.

One of the tools that allows SDBMS to be so effective is the *filter-and-refine* paradigm [78, Chap. 1.6.3]:

- The objects to be queried are represented by the smallest rectangles that contain them - a Minimum Bounding Rectangle (MBR) - as it is easier to compute the intersection between the desired region and a rectangle than if it was an arbitrary shape;
- **Filter** step: every polygon whose MBR does not intersect the query region is discarded;
- **Refine** step: The results of the previous step are processed utilizing their exact geometries. This step is computationally expensive, but it has low cardinality due to the execution of the filter step immediately before.

Another important point is the data structures used as a basis for spatial indexation. If we

only want to query the data, we can just add a field to a standard database. However, if we want to perform operations on it, we require a more appropriate way of representing them [80]. A typical solution, that is compatible with filter-and-refine, are hierarchical trees, or R-Trees.

B-Trees [78, Chap. 1.6.4] are the most popular indexation method in standard relational databases. They are binary self-balanced trees, which are built so it takes approximately the same time to access any line of the index [77, p.497]. However, they cannot be used directly for a spatial object index, as there is no default natural order in these objects, as opposed to numbers, strings or dates, where each value is greater, lesser or equal to the next value [81].

SDBMS instead resort to R-Trees, departing from the default B-Tree concept to accommodate spatial objects: each node in the tree represents the smallest rectangle that contains its child-nodes, and the leaf nodes of the tree are pointers to the objects in the database. There are a few other alternatives, such as quadtrees or methods based in spatial occupation (bucketing) [80], but R-Trees are the default indexation method included in most SDBMS, such as PostGIS.

As we are going to store geographical data, it is important to keep these factors in consideration, so as to optimize the way that data is stored and utilized.

5.5 Backend Database

If we opt for an SQL DBMS to store our measurements on the server, there are a few alternatives to consider: the two we will focus are MySQL and PostgreSQL. PostgreSQL is an open-source DBMS [82, Chap. 1.3.1], with functionality and speed competitive with commercial offerings. An advantage in relation to its primary competitor, MySQL, is the fact it is entirely open source (no company backing it), allowing it to have a very permissive license. While PostgreSQL uses a Berkeley-like license, MySQL uses a dual licensing model [83], GPL-based for regular users but not for companies, where their license involves royalties.

Another further advantage for PostgreSQL in the SQL field is the GIS built on top of it, PostGIS. PostGIS is a tool that allows location-based queries using PostgreSQL, adding spatial functions, operators, data types and indexing modes to the DBMS, and integrating several auxiliary tools, such as the Proj4 projection system [82, Chap. 1.3.2].

Considering databases other than SQL, though, there are a few potential alternatives. NoSQL (often interpreted as Not Only SQL) is an unspecific term that is generally used to define languages

with one, or more, of the following characteristics, according to [84]:

- Not using the relational model (neither the SQL language);
- Open source;
- Designed to run on large clusters;
- Based on the needs of 21st century web properties;
- No schema, allowing fields to be added to any record without controls;

The leading NoSQL DBMS is MongoDB [85]. It utilizes a JSON-like document data model (BSON, or binary JSON) [86], and it supports geospatial secondary indexes, which could make it usable with our application, particularly as the data is being sent to the server in JSON already. Still, previous experience with Postgres and the power of PostGIS has made us decide to opt for this solution instead.

Even after making that choice, however, there are a few details to consider, notably, how to store the geographical data within Postgres. Since PostGIS 1.5, there are two different spatial data types, *geometry* and *geography*. The geography data type allows us to store data in longitude and latitude and get results from the built-in functions in meters; it is not as precise as geometry, however, and it supports less [87, Chap. 13.3] and slower [82, Chap. 1.5.2] functions than geometry. Nonetheless, in our case, since we are storing data at a global scale, geometry would not be a good match, as that data type is mostly designed for storing data with a local point of view, unlike geography, which was designed for global scale data [88] [87, Chap. 4.2.2].

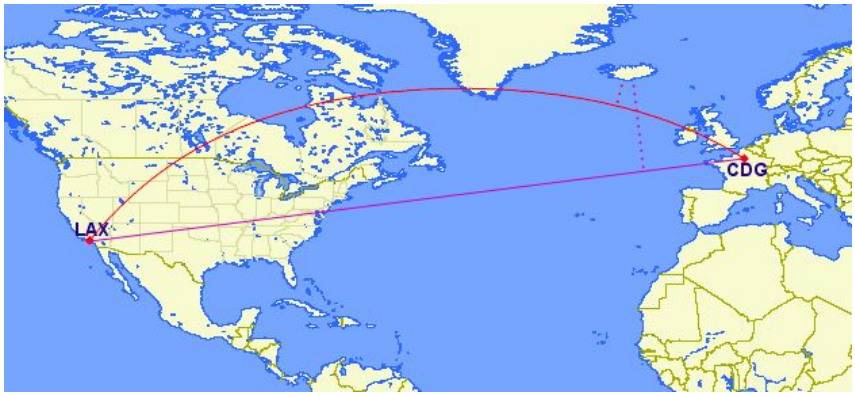

Figure 5.1: How close will a flight from Los Angeles to Paris come to Iceland, by Boundless via Boundlessgeo.com (CC-BY-NC-SA 3.0).

A concrete example of the comparative advantages of geography in this case is in [88], where they refer to a situation where a user wants to calculate the distance between Los Angeles and Paris. This query would return 9124km if the data was stored in geography, and a meaningless value of 121 degrees when done in geometry, but with the same spatial reference system geography uses. Figure 5.1 shows a further example, as it represents the answer to the question "How close will a flight from Los Angeles to Paris come to Iceland?". The Cartesian approach used by the geometry data type is in purple, while the great circle route used by geometry is in red. Figure 5.2 further represents this problem, as the geometry data type returns a completely wrong solution for the shortest distance between the Narita and Los Angeles airports.

Figure 5.2: What is the shortest route from Los Angeles to Tokyo, by Boundless via Boundlessgeo.com (CC-BY-NC-SA 3.0).

Finally, there is a way to account for the limited number of functions available for the geometry data type. If we want to use a function that only supports geometry data type, all we have to do is add :: *geometry* to the variable we pass to that function, for example, `ST_X(geog::geometry)` will return the horizontal coordinate of a geography object (longitude). However, this may have performance implications, and the selection of geography may be a part of the performance issues mentioned in Chapter 4.

5.6 Server-side technologies

When developing the web server, there were two requirements that it should fulfill: hosting the web page and handling the communication and data processing. We decided to opt for a solution that could handle both in the same machine, for ease of deployment. We looked at several alternatives to achieve this:

- Firstly, we considered the *close to the metal* approach, where we would implement our own web server. This would be the most powerful of the options, because it would allow us to develop it exactly according to the specifications of the work in question. However, it would require a lot of knowledge in the area, and would naturally be a lot less mature than the alternatives, which have been developed by large teams over a period of years (sometimes decades).
- Secondly, we considered the Java Servlet approach, utilizing a HTTP server/servlet container such as Apache Tomcat or Eclipse's Jetty. Jetty is entirely standalone, both working for deploying an application and a web server. Tomcat, on the other hand, requires the Apache web server to run, which would require extra configurations [89].
- Finally, there was the option to go with an Model-View-Controller (MVC) web framework such as the Play framework (Java) or Django (Python).

Out of the three, we opted for the servlet approach, since it is the easiest to implement and understand. Jetty is Java-based, requires small effort in learning, and the programming of a servlet is simple (implement a **doGet()** and/or a **doPost()** method and add a few lines to a configuration file) compared to the time it would take to learn an MVC framework.

5.7 Geographical options

Google Maps's API utilizes a particular, Google-implemented version of the Mercator projection, commonly known as Google Web Mercator [90]. This variation of the Mercator projection has become a standard in web mapping, being used by virtually all online mapping providers, including OpenStreetMap [91]. However, the APIs we use to draw the data on top of the map require us to provide the coordinates in latitude / longitude, in the reference coordinate system called World Geodetic System (WGS) 84. This coordinate system, which is used by GPS,

represents the coordinates on the surface of an ellipsoid (the Earth) while Google Web Mercator is the **projection** of the surface of that ellipsoid into a two dimensional plane.

This projection has some flaws, however: The scale of the map increases vertically, from the Equator to the poles, where it becomes infinite - Google's version of Mercator cuts off the map at approximately 85°N/S. It also distorts the size and shape of large objects. Furthermore, for our particular case, it means that, if drawn in meters, the grid squares that we want to draw will not have the same meaning at (0,0) and in the north of Europe, for example - Google might call its units meters, but they are not real meters. The more to the north (or to the south) you go, the more distorted they get.

A way to solve this would be to draw a grid of rectangular cells using an equal-area projection, such as the Lambert equal area projection [92], seen in Figure 5.3, which distorts shape but preserves area. However, this would also require conversion work in order to display the grid on the Leaflet map, as we would have to convert to WGS 84 in order to pass the values onto the application, and this conversion would make the grid squares show up rotated and distorted on the map. Figure 5.4 shows what that could look like: both squares are supposedly a 500x500m area and have the same bottom-left point; however, the Lambert-projected square, in blue, appears to cover a much wider area due to the reprojecting.

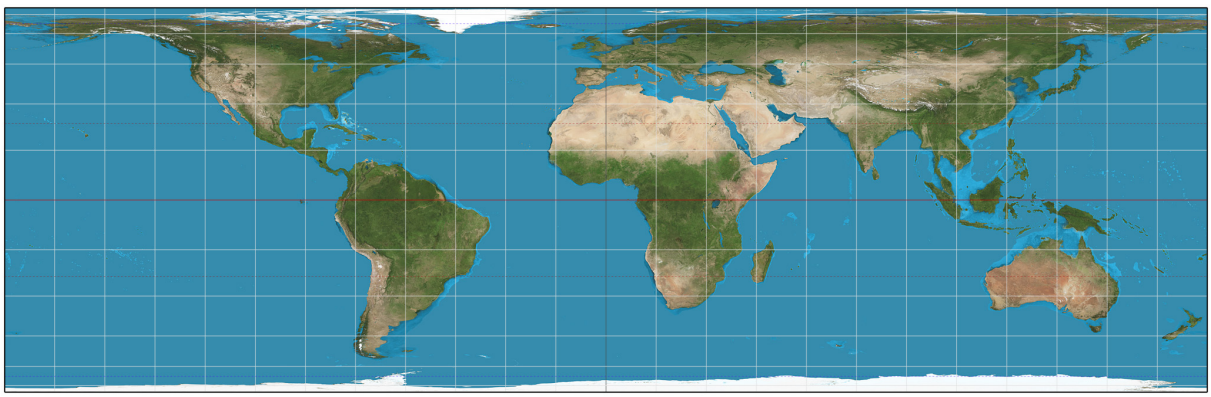

Figure 5.3: A version of the Lambert cylindrical equal-area projection, by Strebe via Wikimedia Commons (CC-BY-SA 3.0).

We decided to go for the simplest solution: take the horizontal and vertical sizes of the Google Web Mercator projection, and divide them into degrees. This makes the area represented by the grid squares inconsistent from pole to pole, but preserves their shape, and makes the number of grid cells constant throughout each "slice" of the map.

The sizes we decided for this were done by projecting a 500x500m square at the Equator, and taking the latitude/longitude degrees of it as a base for what a grid cell should look like. We had to work under the limitations of Google Web Mercator, namely, the cutoff at 85 degrees north/south. This made it so that some adjustments were necessary to get a whole number of grid cells, making the default values the application uses:

- 0.0044º per cell for the latitude, which goes from -85.0014 to 85.0014, resulting in a total of 38 637 vertical units in the grid;

- 0.0045º for the longitude, which goes from -180 to 180, resulting in 80 000 horizontal units.

With these values, the grid can have a maximum of 3 090 960 000 cells, which would make the database very hard to query, if they were all on the same machine. Therefore, if the database got large enough, we would have to use a solution similar to that done for the Mozilla cells database, and divide it among multiple machines. Another alternative could be having more than one possible size for the cells, and depending on the level of zoom used, displaying a different one.

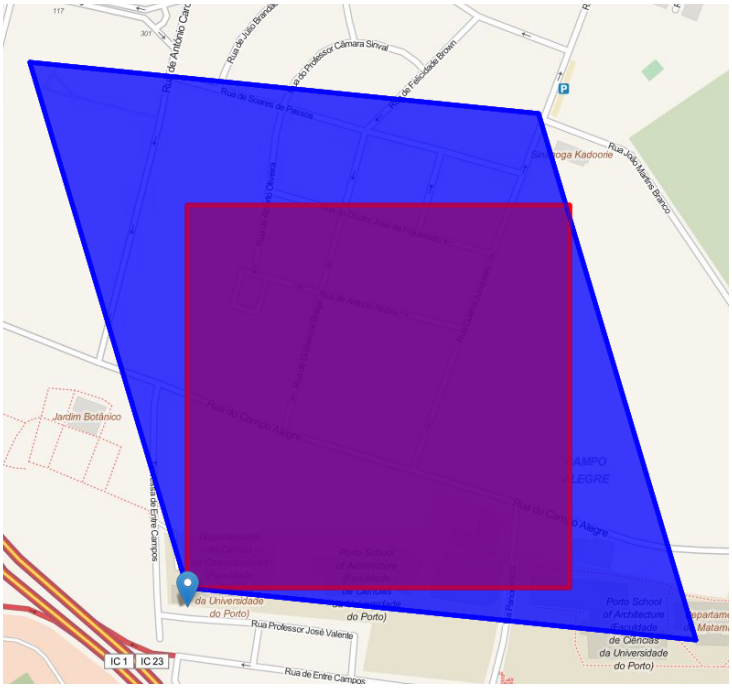

Figure 5.4: An example of what a Lambert-projected grid square could look like compared to a 500x500m square in Google Web Mercator.

5.8 Chapter Summary

We studied the different hardware possibilities for the project and decided to utilize mobile devices with the Android operating system for the collection of data. Furthermore, we briefly examined the alternatives in terms of dedicated hardware. We also studied the technology required to draw a coverage map, having decided to utilize the Leaflet JavaScript API in combination with tiles provided by OpenStreetMap to draw the map, to use PostgreSQL and PostGIS to store the data for the map, and to implement the server required utilizing Servlets in Java with the Jetty server. Finally, we considered the alternatives for the geographical representation: the database support, the projection used, and the size of the grid cells.

Chapter 6

Conclusion

This work began with the objective of developing an application that would empower people to collect data from their everyday experience in order to have a trusted, independent source of information on wireless network coverage. The developed solution implements all the requirements we proposed for that purpose, and successfully achieves our objectives.

The application is available publicly, at github.com/josemaia/openNetworkMeasurer and github.com/josemaia/openNetworkMeasurer-server.

We developed a data collection tool in Android that can measure data on cellular and Wi-Fi networks whenever the user wants to do so, and associate them with the geographic position where they performed those measurements. The obtained information can be sent at the user's convenience.

An important lesson learned while developing this component was to consider the wide variety of Android hardware. Every phone the application was tested on had its own particularities, of which the most notable ones were the minimum time between Wi-Fi scans and the data that the hardware reported from the cell towers.

We also created an architecture that can store the data sent from every user and process it. A particular way of processing this data was also implemented, consisting of creating a signal strength grid covering the Earth to be displayed in a coverage map. This coverage map was also implemented, as well as a second coverage map which combines external data with data obtained by our application to display the position of cell towers. This external data is obtained from the Mozilla Ichnaea database, and a solution to obtain that data was also proposed.

We decided to use crowdsourcing, and the conclusion we took from that option was that it is necessary to provide users with a motivation to contribute. In our work, the motivation is both intrinsic (everyone naturally wants good, independent information) and extrinsic (the leaderboard we provide).

The work now serves as a basis for others to build upon in the future. With more measurements in our database, it will be interesting to cross them over with external data, in order to perform studies such as the coverage within different geographical areas, the correlation of signal strength with the topographical information in a given area, or the location of public Wi-Fi hotspots.

The most notable weakness of this work is the database performance. As shown in Chapter 5, the maximum possible number of cells is very large, thus making optimizations to this performance important, and investigation of any possible problems a focus for future work.

The data import and export features also have some work left to do: right now, we only obtain data from the Mozilla Ichnaea database, but we could also submit data to it, and potentially import/export data from other sources, such as OpenCellID. It should also be possible to import the data from Ichnaea iteratively, instead of all at once, but only when PostgreSQL's support for that feature is finalized.

There is also more work to be done on the speedtest, as it only calculates the download speed and not the upload speed, and doesn't yet associate the test with the geographical position. This association could be performed and used to draw an extra layer on the coverage map.

Finally, the imported database of the administrative regions ended up not being used. If we had a sufficient amount of data on the database, this external information would make it possible to perform comparative studies such as "what municipality has the best 4G signal" or "which network provides the best signal, on average, in the district of Porto".

Bibliography

[1] PewResearch Internet Project. Mobile Technology Fact Sheet. http://www.pewinternet.org/fact-sheets/mobile-technology-fact-sheet/, January 2014. Consulted on 09/12/2014.

[2] ANACOM. Versão integral do relatório Serviços Móveis - 3.º trimestre de 2014. http://www.anacom.pt/streaming/STM_3Trim2014.pdf?contentId=1340610&field=ATTACHED_FILE, November 2014. Consulted on 09/12/2014.

[3] ANACOM. ANACOM recebe mais de 17 mil reclamações no 1.º trimestre. http://www.anacom.pt/render.jsp?contentId=1278112, July 2014. Consulted on 09/12/2014.

[4] MEO. Prémios - MEO. http://www.meo.pt/diversos/premios#bloco(cobertura), December 2014. Consulted on 09/12/2014.

[5] NOS. Há mais em NOS - NOS. http://www.nos.pt/empresas/Pages/ha-mais-em-nos.aspx, December 2014. Consulted on 09/12/2014.

[6] Vodafone. Porquê escolher a Banda Larga Móvel da Vodafone? http://www.vodafone.pt/main/Ajuda/Cobertura/estudo-melhor-rede, December 2014. Consulted on 09/12/2014.

[7] Vodafone. Teste a Cobertura da Rede Móvel da Vodafone. http://www.vodafone.pt/main/Particulares/cobertura-internet-movel.htm, December 2014. Consulted on 09/12/2014.

[8] Stuart Taylor and Tine Christensen. What do consumers want from public wi-fi? gain insights from cisco's mobile consumer research. *Cisco White Papers*, 2013. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/service-provider-wi-fi/white-paper-c11-729797.pdf, Consulted on 09/12/2014.

[9] eduroam. eduroam > faq. https://www.eduroam.org/index.php?p=faq, December 2014. Consulted on 09/12/2014.

[10] Fon Wireless, Ltd. Home | Fon. https://corp.fon.com/en, December 2014. Consulted on 09/12/2014.

[11] Amit Kumar, Dr Yunfei Liu, and Jyotsna Sengupta. Evolution of Mobile Wireless Communication Networks 1G to 4G. *International Journal of Electronics & Communication Technology, IJECT*, 1(1), 2010.

[12] Martin Sauter. *From GSM to LTE: an introduction to mobile networks and mobile broadband.* John Wiley & Sons, 2010.

[13] Berthold K.P. Horn. Base Station Numbering Schemes. http://people.csail.mit.edu/bkph/cellular_repeater_numerology.shtml, 2014. Consulted on 14/06/2015.

[14] Oscar Bejarano, Edward W Knightly, and Minyoung Park. Ieee 802.11 ac: from channelization to multi-user mimo. *IEEE Communications Magazine*, 51(10):84–90, 2013.

[15] Erik Tews, Ralf-Philipp Weinmann, and Andrei Pyshkin. Breaking 104 bit wep in less than 60 seconds. In *Information Security Applications*, pages 188–202. Springer, 2007.

[16] ANACOM. Serviços Móveis - Deliberação de 08.07.2009, alterada pelas deliberações de 17.06.2010, de 19.08.2010 e de 30.08.2012. http://www.anacom.pt/render.jsp?contentId=963861, November 2012. Consulted on 16/12/2014.

[17] GSMA Intelligence. Measuring mobile penetration. https://gsmaintelligence.com/analysis/2014/05/measuring-mobile-penetration/430/, May 2014. Consulted on 04/01/2015.

[18] Grupo Marktest. Barómetro Telecomunicações - Voz Móvel. http://www.marktest.com/wap/a/grp/p~2.aspx#, 2014. Consulted on 16/12/2014.

[19] GfK TEMAX® Portugal. Results for GfK TEMAX® Portugal for the third quarter of 2014. http://www.gfk.com/temax/Documents/2014-Q3/2014-11-15_GfK_TEMAX_Press_Release_Portugal_en.pdf, 2014. Consulted on 16/12/2014.

[20] Hugo Séneca. A 3G do Minho ao Algarve. http://exameinformatica.sapo.pt/especiais/2010-02-12-a-3g-do-minho-ao-algarve, 12/02/2010. Consulted on 16/12/2014.

[21] Yu Zhang and Mihaela van der Schaar. Reputation-based incentive protocols in crowdsourcing applications. In *INFOCOM, 2012 Proceedings IEEE*, pages 2140–2148. IEEE, 2012. http://arxiv.org/ftp/arxiv/papers/1108/1108.2096.pdf.

[22] Jeff Howe. The rise of crowdsourcing. *Wired Magazine*, 14(6):1–4, 2006.

[23] Junxian Huang, Qiang Xu, Birjodh Tiwana, Z. Morley Mao, Ming Zhang, and Paramvir Bahl. Anatomizing application performance differences on smartphones. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 165–178, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-985-5. http://research.microsoft.com/pubs/131377/MobiSys10-3GTest.pdf. doi:10.1145/1814433.1814452.

[24] S. Sonntag, L. Schulte, and J. Manner. Mobile network measurements - it's not all about signal strength. In *Wireless Communications and Networking Conference (WCNC), 2013 IEEE*, pages 4624–4629, April 2013. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6555324. doi:10.1109/WCNC.2013.6555324.

[25] Miquel Martínez Raga. An Android application for crowdsourcing 3G user experience. Final project report, Universitat Politècnica de València, September 2011. http://riunet.upv.es/bitstream/handle/10251/11987/memoria.pdf?sequence=1.

[26] Ookla. Ookla | what-is-the-test-flow-and-methodology-for-the-speedtest. http://www.ookla.com/support/a21110547/what-is-the-test-flow-and-methodology-for-the-speedtest, December 2014. Consulted on 17/12/2014.

[27] S. Rosen, Sung-Ju Lee, Jeongkeun Lee, P. Congdon, Z.M. Mao, and K. Burden. Mcnet: Crowdsourcing wireless performance measurements through the eyes of mobile devices. *Communications Magazine, IEEE*, 52(10):86–91, October 2014. ISSN 0163-6804. doi:10.1109/MCOM.2014.6917407.

[28] Sensorly. Sensorly | About Us. http://www.sensorly.com/about-us, December 2014. Consulted on 17/12/2014.

[29] Sensorly. Sensorly | About Us. http://www.sensorly.com/confidentiality, December 2014. Consulted on 17/12/2014.

[30] OpenSignal, Inc. About Us - OpenSignal. http://opensignal.com/about.php, December 2014. Consulted on 17/12/2014.

[31] OpenSignal, Inc. OpenSignal API - OpenSignal. http://developer.opensignal.com/, December 2014. Consulted on 17/12/2014.

[32] Trois Petis Points. 4GmarkMobile performance test. http://4gmark.com/, December 2014. Consulted on 17/12/2014.

[33] RootMetrics. Home | RootMetrics. http://www.rootmetrics.com/us, December 2014. Consulted on 17/12/2014.

[34] Netradar.org. Netradar. https://www.netradar.org/en/about, December 2014. Consulted on 17/12/2014.

[35] Ookla. Speedtest.net by Ookla - About Ookla Speedtest. http://www.speedtest.net/about, December 2014. Consulted on 17/12/2014.

[36] Ookla. Ookla | What-fields-are-included-in-Android-data-extracts. http://www.ookla.com/support/a24633461/What-fields-are-included-in-Android-data-extracts, December 2014. Consulted on 17/12/2014.

[37] Robust Net Research Group @ UMich. MobiPerf. http://www.mobiperf.com/home, December 2014. Consulted on 17/12/2014.

[38] Sanae Rosen, Haokun Luo, Qi Alfred Chen, Z Morley Mao, Jie Hui, Aaron Drake, and Kevin Lau. Understanding RRC state dynamics through client measurements with mobilyzer. In *Proceedings of the 6th annual workshop on Wireless of the students, by the students, for the students*, pages 17–20. ACM, 2014. http://web.eecs.umich.edu/~sanae/sss09i-rosen.pdf.

[39] Federal Communications Commission. Chairman Announces Challenge.gov Competition Winners. http://www.fcc.gov/blog/chairman-announces-challengegov-competition-winners, August 2011. Consulted on 04/01/2015.

[40] Mozilla. MLS - Client Applications. https://location.services.mozilla.com/apps, December 2014. Consulted on 17/12/2014.

[41] Mozilla. MLS - Downloads. https://location.services.mozilla.com/downloads, December 2014. Consulted on 17/12/2014.

[42] OpenCellID. OpenCellID - OpenCellID. opencellid.org, December 2014. Consulted on 17/12/2014.

[43] Samsung. Developing Android Application Using Atmospheric Pressure Sensor. http://developer.samsung.com/technical-doc/view.do;jsessionid=QfQVVtHL216n2NwHG8TlB1LCCMJQmlGMttkRJg427HwQtx2L04Jn!1387855660?v=T000000127, February 1, 2013. Consulted on 02/06/2015.

[44] Jean-Paul M.G. Linnartz. Shadowing. http://www.wirelesscommunication.nl/reference/chaptr03/shadow/shadow.htm, 1993-1995. Consulted on 02/06/2015.

[45] Global Administrative Areas. Global Administrative Areas. http://gadm.org/, January 2012. Consulted on 12/06/2015.

[46] Mozilla. Data Import / Export. https://mozilla-ichnaea.readthedocs.org/en/latest/import_export.html, 2015. Consulted on 03/06/2015.

[47] Mozilla. Geosubmit Version 2. http://mozilla-ichnaea.readthedocs.org/en/latest/api/geosubmit2.html#api-geosubmit-latest, 2015. Consulted on 14/06/2015.

[48] Android Open Source Project. SensorManager.java. https://android.googlesource.com/platform/frameworks/base/+/master/core/java/android/hardware/SensorManager.java, February 10, 2015. Consulted on 24/06/2015.

[49] European Telecommunications Standards Institute. ETSI TS 127 007 V12.6.0 (2014-10). *3GPP*, October 2014.

[50] Android Open Source Project. CellSignalStrengthCdma.java. https://android.googlesource.com/platform/frameworks/base.git/+/master/telephony/java/android/telephony/CellSignalStrengthCdma.java, April 11, 2014. Consulted on 24/06/2015.

[51] Adam J. Hodges. Reading LTE signal strength (RSSI) in older versions of the Android SDK. Also, Java reflection. http://blog.ajhodges.com/2013/03/reading-lte-signal-strength-rssi-in.html, March 28, 2013. Consulted on 19/06/2015.

[52] Maxime Petazzoni. Ttorrent, BitTorrent library in Java. http://mpetazzoni.github.io/ttorrent/, 2013. Consulted on 28/06/2015.

[53] stackExchange inc. How to resolve a library conflict (apache commons-codec). http://stackoverflow.com/questions/12285615/how-to-resolve-a-library-conflict-apache-commons-codec/16916552#16916552, September 5, 2012. Consulted on 28/06/2015.

[54] The PostgreSQL Global Development Group. INSERT - PostgreSQL 9.5devel Documentation. http://www.postgresql.org/docs/devel/static/sql-insert.html, 2015. Consulted on 25/06/2015.

[55] ROHDE&SCHWARZ. R&S CMW500 - Platform Overview. http://www.rohde-schwarz.com/en/product/cmw500_overview-productstartpage_63493-10844.html, 2015. Consulted on 05/01/2015.

[56] GNU Radio. Welcome to GNU Radio! http://gnuradio.org/redmine/projects/gnuradio/wiki, 2015. Consulted on 05/01/2015.

[57] GNU Radio. The OpenBTS Wiki Subspace. https://gnuradio.org/redmine/projects/gnuradio/wiki/UsrpFAQIntro, 2015. Consulted on 05/01/2015.

[58] Ettus Research. Ettus Research - Quick Order. http://www.ettus.com/product/quick-order, 2015. Consulted on 05/01/2015.

[59] GNU Radio. The OpenBTS Wiki Subspace. http://gnuradio.org/redmine/projects/gnuradio/wiki/OpenBTS/98, 2015. Consulted on 05/01/2015.

[60] OsmoSDR. rtl-sdr - OsmoSDR. http://sdr.osmocom.org/trac/wiki/rtl-sdr, 2015. Consulted on 05/01/2015.

[61] Domonkos P. Tomcsányi. The big GSM write-up – how to capture, analyze and crack GSM? – 1. http://domonkos.tomcsanyi.net/?p=418, Oct 13, 2013. Consulted on 05/01/2015.

[62] Google. Android Debug Bridge. http://developer.android.com/tools/help/adb.html, 2015. Consulted on 16/05/2015.

[63] Google. Get Started with Publishing. http://developer.android.com/distribute/googleplay/start.html, 2015. Consulted on 16/05/2015.

[64] Andrei Frumusanu. A Closer Look at Android RunTime (ART) in Android L. http://anandtech.com/show/8231/a-closer-look-at-android-runtime-art-in-android-l/, July 1, 2014. Consulted on 16/05/2015.

[65] Gartner, Inc. Gartner Says Tablet Sales Continue to Be Slow in 2015 . http://www.gartner.com/newsroom/id/2954317, January 5, 2015. Consulted on 16/05/2015.

[66] Google. Dashboards. https://developer.android.com/about/dashboards/index.html, 2015. Consulted on 16/05/2015.

[67] Apple. Swift. https://developer.apple.com/support/ios/enrollment.php, 2015. Consulted on 16/05/2015.

[68] Apple. Program Enrollment. https://developer.apple.com/support/ios/enrollment.php, 2015. Consulted on 16/05/2015.

[69] Guvener Gokce. iPhone Wireless Scanner iOS5. http://blog.guvenergokce.com/iphone-wireless-scanner-ios5/170/, November 5, 2011. Consulted on 16/05/2015.

[70] Google. Usage Limits and Billing. https://developers.google.com/maps/documentation/javascript/usage?hl=en-US, January 15, 2015. Consulted on 21/01/2015.

[71] OpenStreetMap Wiki. About. http://wiki.openstreetmap.org/wiki/About, January 2015. Consulted on 21/01/2015.

[72] OpenStreetMap Wiki. Why not Google? http://wiki.openstreetmap.org/wiki/FAQ#Why_don.27t_you_just_use_Google_Maps.2Fwhoever_for_your_data.3F, January 2015. Consulted on 28/06/2015.

[73] Microsoft. Microsoft® Bing™ Maps Platform APIs' Terms Of Use. http://www.microsoft.com/maps/product/terms.html, May 2014. Consulted on 21/01/2015.

[74] OpenStreetMap and Contributors. SWITCH2OSM. http://switch2osm.org/the-basics/, 2013. Consulted on 21/01/2015.

[75] Stefano Cudini. AJAX Example: load json data by Ajax request. http://labs.easyblog.it/maps/leaflet-layerjson/examples/simple.html, 2014. Consulted on 21/01/2015.

[76] Vladimir Agafonkin. Notable Leaflet Plugins. http://leafletjs.com/plugins.html, 2014. Consulted on 21/01/2015.

[77] Carlos Coronel, Steven Morris, and Peter Rob. *Database systems: design, implementation, and management*. Cengage Learning, 2009.

[78] Shashi Shekhar and Sanjay Chawla. *Spatial Databases: A Tour*. Prentice Hall, 2003.

[79] Shashi Shekhar and Sanjay Chawla. *Slides for Spatial Databases: A Tour*. University of Minnesota, 2003.

[80] Hanan Samet. *Spatial Data Structures*, pages 361–385. ACM Press and Addison-Wesley, 1995.

[81] BoundlessGeo.com. Introduction to PostGIS Section 1: Introduction. http://workshops.boundlessgeo.com/postgis-intro/introduction.html, 2014. Consulted on 14/06/2015.

[82] Regina Obe and Leo Hsu. *PostGIS in action*. Manning Publications Co., 2011.

[83] Oracle Corporation. Commercial License for OEMs, ISVs and VARs. http://www.mysql.com/about/legal/licensing/oem/, July, 2010. Consulted on 21/01/2015.

[84] Martin Fowler. NosqlDefinition. http://martinfowler.com/bliki/NosqlDefinition.html, 9 January 2012. Consulted on 21/01/2015.

[85] MongoDB, Inc. MONGODB – THE LEADING NOSQL DATABASE. http://www.mongodb.com/leading-nosql-database, 2015. Consulted on 21/01/2015.

[86] MongoDB, Inc. JSON and BSON. http://www.mongodb.com/json-and-bson, 2015. Consulted on 21/01/2015.

[87] Paul Ramsey et al. PostGIS manual 2.0. http://postgis.net/docs/manual-2.0/, 2015. Consulted on 21/01/2015.

[88] BoundlessGeo.com. Introduction to PostGIS Section 17: Geography. http://workshops.boundlessgeo.com/postgis-intro/geography.html, 2014. Consulted on 14/06/2015.

[89] Webtide.com. Why Choose Jetty? https://webtide.com/why-choose-jetty/, 2014. Consulted on 21/01/2015.

[90] Google. Google Maps JavaScript API. https://developers.google.com/maps/documentation/javascript/maptypes?csw=1#MapCoordinates, 2015. Consulted on 27/06/2015.

[91] OpenStreetMap Wiki. EPSG:3857. http://wiki.openstreetmap.org/wiki/EPSG:3857, May 2015. Consulted on 27/06/2015.

[92] Karen Mulcahy. Cylindrical Projections . http://www.geo.hunter.cuny.edu/mp/cylind.html, September 5, 1997. Consulted on 27/06/2015.

[93] Mary Natrella. *NIST/SEMATECH e-Handbook of Statistical Methods*. NIST/SEMATECH, July 2010. http://www.itl.nist.gov/div898/handbook/.

Appendix A

Mozilla Ichnaea import/export

Records should be written one record to a line, with \n (0x0A) as line separator.

A value should be written as an empty field – two adjacent commas, for example – rather than being omitted.

The first five fields (radio to cell) jointly identify a unique logical cell network. The remaining fields contain information about this network.

The data format does not specify the means and exact algorithms by which the position estimate or range calculation was done. The algorithms might be unique and changing for each source of the data, though both Mozilla Location Services (MLS) and OpenCellID currently use similar and comparable techniques.

- **radio**

  Network type. One of the strings GSM, UMTS, LTE or CDMA.

- **mcc**

  Mobile Country Code. An integer, for example 505, the code for Australia.

- **net**

  For GSM, UMTS and LTE networks, this is the Mobile Network Code (MNC). For CDMA networks, this is the System Identification number (SID). An integer, for example 4, the MNC used by Vodaphone in the Netherlands.

- **area**

For GSM and UMTS networks, this is the Location Area Code (LAC). For LTE networks, this is the Tracking Area Code (TAC). For CDMA networks, this is the Network Identification number (NID). An integer, for example 2035.

- **cell**

  For GSM and LTE networks, this is the Cell ID (CID). For UMTS networks this is the UTRAN Cell ID / LCID, which is the concatenation of 2 or 4 bytes of Radio Network Controller (RNC) code and 4 bytes of Cell ID. For CDMA networks this is the Base station Identifier number (BID). An integer, for example 32345.

- **unit**

  For UMTS networks, this is the Primary Scrambling Code (PSC). For LTE networks, this is the Physical Cell ID (PCI). For GSM and CDMA networks, this is empty. An integer, for example 312.

- **lon**

  Longitude in degrees between -180.0 and 180.0 using the WGS 84 reference system. A floating point number, for example 52.3456789.

- **lat**

  Latitude in degrees between -90.0 and 90.0 using the WGS 84 reference system. A floating point number, for example -10.034.

- **range**

  Estimate of radio range, in meters. This is an estimate on how large each cell area is, as a radius around the estimated position and is based on the observations or a knowledgeable source. An integer, for example 2500.

- **samples**

  Total number of observations used to calculate the estimated position, range and averageSignal. An integer, for example 1200.

- **changeable**

  Whether or not this cell is a position estimate based on radio observations, and therefore subject to change in the future, or is an exact location entered from a knowledgeable source. A boolean value, encoded as either 1 (for “changeable”) or 0 (for “exact”).

- **created**

  Timestamp of the time when this record was first created. An integer, counting seconds since the Coordinated Universal Time (UTC) Unix Epoch of 1970-01-01T00:00:00Z. For example, 1406204196, which is the timestamp for 2014-07-24T12:16:36Z.

- **updated**

  Timestamp of the time when this record was most recently modified. An integer, counting seconds since the UTC Unix Epoch of 1970-01-01T00:00:00Z. For example, 1406204196, which is the timestamp for 2014-07-24T12:16:36Z.

- **averageSignal**

  Average signal strength from all observations for the cell network. An integer value, in dBm. For example, -72.

  This field is only used by the OpenCellID project and historically has been used as a hint towards the quality of the position estimate.

Appendix B

Data smoothing

Smoothing is a statistical technique designed to cancel the effect of random variation to reveal trends in data collected over time. [93, Chap. 6.4] There are many ways to smooth data - from a simple average of all the past data (terrible when it comes to predicting trends) to complex exponential formulas designed to gradually weigh the past measurements less and less.

One example of such exponential formulas is the Exponentially Weighted Moving Average, which can be formulated like this [93, Chap. 6.3.2.4]:

$$\mathrm{EWMA}_t = \lambda Y_t + (1 - \lambda)\mathrm{EWMA}_{t-1} \text{ for } t = 1, 2, \ldots, n\,. \tag{B.1}$$

where,

- EWMA_0 is the mean of the historical data;
- Y_t is the observation at time t;
- n is the number of observations to be monitored including EWMA_0;
- $0 < \lambda \leq 1$ is the **weighting factor**, a constant that determines the depth of memory of the formula.

The choice of λ is very important, as a larger value gives more weight to recent data, while a small value gives more weight to older data - a choice that has to be carefully made. The starting value also affects the data, which makes it complicated to use without pre-existing data.

In this case, we only perform the average when there are two or more values to average, therefore assuming the first value as the historical data.

Appendix C

Application API description

First, there's the generic information we store, that's common to both kinds of measurements, and is stored into either raw measurements tables.

```
String userID;
String manufacturer;
String model;
String hardware;
Integer SdkVersion;
String OsVersion; //all the previous are identifying information of the phone,
    for statistical purposes

String Latitude;
String Longitude; //we can use these two to build a Geography object

Float Accuracy; // accuracy of the latitude/longitude measurement, in meters

String SeaLevelPressure; //only obtained if there's a phone pressure

String PhonePressure; //obtained from an android pressure sensor

float altitude; //calculated in the phone if the two previous ones exist
```

Listing C.1: Complete description of the generic measurements' fields.

Afterwards, there is either the Wi-Fi information, listed immediately below, or the cellular information, listed after it.

```
String timestamp; //measured when the first scan starts

//A set of ScanResultObjects, each containing the following:
String SSID; //shouldn't be necessary to store, and may be identifying
    information

String BSSID; //MAC address, used to identify the access point

String signalStrength; //either the instant value or an average over 5 seconds

String maxSignal;
String minSignal;
Double stDev; //optional statistical information on the signal strength
    measurements

String security; //WPA/WPA2/WPA+WPA2/WEP/OPEN

String keyManagement; //WEP/PSK/EAP/OPEN

String encryption; //CCMP/TKIP/CCMP+TKIP/OPEN

Boolean hasWPS;

Boolean isPublicWifi; //deprecated - if implemented in the future, should be
    done on the server side.

String serviceSet; //ESS/IBSS/BSS

Integer channel; //calculated based on frequency

Boolean hasHiddenSSID; //only for the network we are currently connected to

Integer linkSpeed; //same
```

Listing C.2: Complete description of the Wi-Fi measurements' fields.

```
String timestamp; //saved on measurement start
String networkTypeString; //HSPA, HSPA+, LTE, etc.
String networkClass; //2G, 3G, 4G
String phoneTypeString; //GSM/CDMA
String operatorName;
String operatorID;
String SimOperatorName; //same as operatorName and operatorID, ideally; usually
    only necessary when the other one returns null. Right now, we save both
    types.
String SimOperatorID;
String measurementList; //are these two interesting to save?
String signalStrengthString; // the signal strength, in a string that contains
    both dBm and ASU
// this should probably be changed to two separate integers, as like this it is
    going to be unnecessarily parsed back into integers on the server side.

Double stdev;
String maxSignal;
String minSignal; //optional statistical information on the signal strength
    measurements

//A set of CellData objects, each containing the following (based on the
    Mozilla Ichnaea format):
String radio; // gsm, umts, lte, cdma
Integer mcc; // mobile country code
Integer mnc; // mobile network code; system identifier in CDMA
Integer lac = -1; // local area code in GSM/UMTS; tracking area code in LTE;
    network ID in CDMA;
Integer cid = -1; // Cell ID; base station ID in CDMA;
Integer signal; // Cell signal strength (RSSI in GSM and CDMA / RSCP in UMTS /
    RSRP in LTE)
Integer asu; // arbitrary signal strength
Integer ta; //only valid for GSM and LTE
Integer psc; //only on UMTS and LTE (where it represents PCI instead)
```

Listing C.3: Complete description of the cellular measurements' fields.

The generic information and one of the others are then packed into a JSON file, and sent to the server.