SignalMapper – Signal Strength Analysis Tool

Joseph Long

April 25th 2022

CS4843 – Wireless and Mobile Computing

1. Introduction

For many years now and throughout many generations of supporting technology, people have been able to communicate with one another wirelessly and with great distances between them through use of cellular technology. This is done through use of radio frequencies to maintain a near constant connection with the public network infrastructure[1]. The quality of the signal connection one's cellphone maintains with their closest network base station is most often measured in decibels or db, and this unit of measurement is used since signal strengths vary logarithmically as opposed to linearly. One can determine how well their connection is performing by checking their signal strength in db, where a negative number large in magnitude indicates a lower quality signal than a negative number smaller in magnitude[1]. As these measurements allow one to determine the relative quality of the signal strength at their immediate location, accumulating more measurements in the same location would improve the original measurement, and enough measurements over a specific geographic location, perhaps a university campus, could give useful information regarding the overall quality of signal strength within that location, along with the specific areas where strength fluctuates. While most modern smartphones feature methods of checking the db of their current signal strength, most often these methods are hidden, obfuscated, and lack the features one might expect when taking measurements[2]. Furthermore, any aggregation of these numbers over time must be done manually when using these methods, not to mention any computation of aggregate statistics. In this report, we will discuss the design and implementation of the SignalMapper application, an application developed for the Android OS platform designed to solve these exact problems, allowing for automatic and effortless collection of signal strength measurements. Along with that, GPS coordinates are collected and paired with measurements, allowing the app to plot said

signal strength measurements on a Google Map automatically. The app includes many other similarly useful features as well, such as automatic measurement grouping and automatic calculations of aggregate statistics.

2. Background

This project is a design and development oriented one. Due to this fact and the resulting nature of the application being built, very little background knowledge not already discussed in the introduction is required for a proper understanding of the reports remaining sections. The is further accentuated by the app relying very little to no theoretical knowledge other than the nature of what is being measured, that being decibel signal strength measurements. Despite this, the methods and architecture the app is developed with, that being the Jetpack Compose framework, is relatively new and should be briefly discussed.

2.1 Android Development and Jetpack Compose

In traditional Android development, a developer must develop not only in their programming language of choice, a selection which is usually restricted to either Java or Kotlin, but also in XML, or the extensible markup language. While the programming language of choice is used to define business logic and update the user interface of the app, XML is used to define the views and fragments of the user interface itself, as elements[3]. This is a language pairing quite like that of web languages, where HTML is used to define the contents of a webpage and a scripting language such as JavaScript is used to define its logic, modifying the HTML elements when needed. The need to switch back and forth between a programming language and a markup language throughout development can be seen as cumbersome, though, which is why JavaScript frameworks like React[4] and Vue JS[5] have been developed for making web applications. These

frameworks blur the line between the two languages by enabling declarative and stateful user interface definitions. Just as the need to blur the lines for web languages led to the development of these frameworks, the same can be said for Android development, where the need to define UI components without a separate language led to the development of Jetpack Compose.

Jetpack Compose is a framework developed by Google's Android team, which offers a toolkit for UI development that allows developers to define app user interfaces and components in Kotlin alongside their business logic, replacing the need for XML user interfaces whilst still allowing an appropriate separation between interface definitions and business logic[3]. Compose uses a declarative design philosophy, meaning that as opposed to imperative design, where a programmer must describe how they would like something to be done, a programmer instead simply must describe what they want done. This preserves the simple nature of markup languages, in which a developer only must describe the user interface, allowing the runtime to decide how to render it, whilst allowing the developer to declare the interface in an imperative language.

3. Design and Implementation

In this section we will briefly discuss the design of the SignalMapper app's overall architecture and functionality, along with its implementation. We will first begin by discussing the apps functionality from a non-detail-oriented point of view to get a proper understanding of the app before getting into specifics. After this, we'll discuss, in order, how the app collects its measurements, how it processes them, and how they are displayed to the user.

3.1 Overview

Before discussing how the app accomplishes what it does, lets first briefly go over what the app sets out to do. The following is a list of requirements set for the development of the app to meet prior to development:

1. Record signal strength at a fixed interval and map it to Google map.
2. Do so automatically and in the background.
3. Show a list of measurements with extra data for analysis.
4. Nearby measurements should be grouped. This allows for aggregate statistics when movement of device is idle.
5. Allow for exporting the measurements.
6. Allow for loading previous measurements.

With these requirements in mind, a brief overview of the app's functionality can be stated as follows. The SignalMapper app, at a regular interval of 10 seconds, takes measurements of the host phone's current signal strength and GPS coordinates, of which the methods for doing so are discussed in section 3.2. It does this in the background. On Android, an app always has a MainActivity, which can be thought of as the entry point of the app like "main" in Java or C. Apps can on top of this have "services", which can be thought of as separate threads for computation, although this is not technically the most accurate comparison in practice. The SignalMapper has a separate service along with the MainActivity, which takes these measurements. This allows the measurements to be taken even when the app is not in focus, or the phone is locked. Nearby measurements (within 10 meters) are grouped, and aggregate statistics including mean, mode, and standard variation are performed on these groups of measurements. This enables measurements to become more accurate as a phone idles at said location, and this functionality is further discussed in section 3.3. These groups of measurements, along with a Google Map displaying the locations of the measurements, are displayed to the user once the measurement service sends the measurements to the MainActivity. Finally, measurements can be exported and imported as JSON objects, further discussed in section 3.5.

3.2 Collection of Measurements

As discussed in section 3.1, measurements are taken in a background service called the "SignalMapperService". This allows for measurements to be taken without the user needing to have the app running in the foreground, for example, when the phone is locked. Measurements can be broken down into two components, the phone's current signal strength in dbm and the user's current GPS coordinates. The first is taken by using the "TelephonyManager" class from the Android API, for which the supporting code is as follows:

```
val cellData = (getSystemService(TELEPHONY_SERVICE) as TelephonyManager).allCellInfo
val dbm = when (cellData[0]) {
        is CellInfoGsm -> (cellData[0] as CellInfoGsm).cellSignalStrength.dbm
        is CellInfoLte -> (cellData[0] as CellInfoLte).cellSignalStrength.dbm
         else -> 0
}
```

The GPS coordinates are obtained at an interval half as long as the measurement interval, via a callback the service registers with the Android API "FusedLocationProvider", which stores the updated location as a global service variable. Once both measurements are taken and the measurement interval has elapsed, measurements are sent to the MainActivity for processing, which we will detail in the following section.

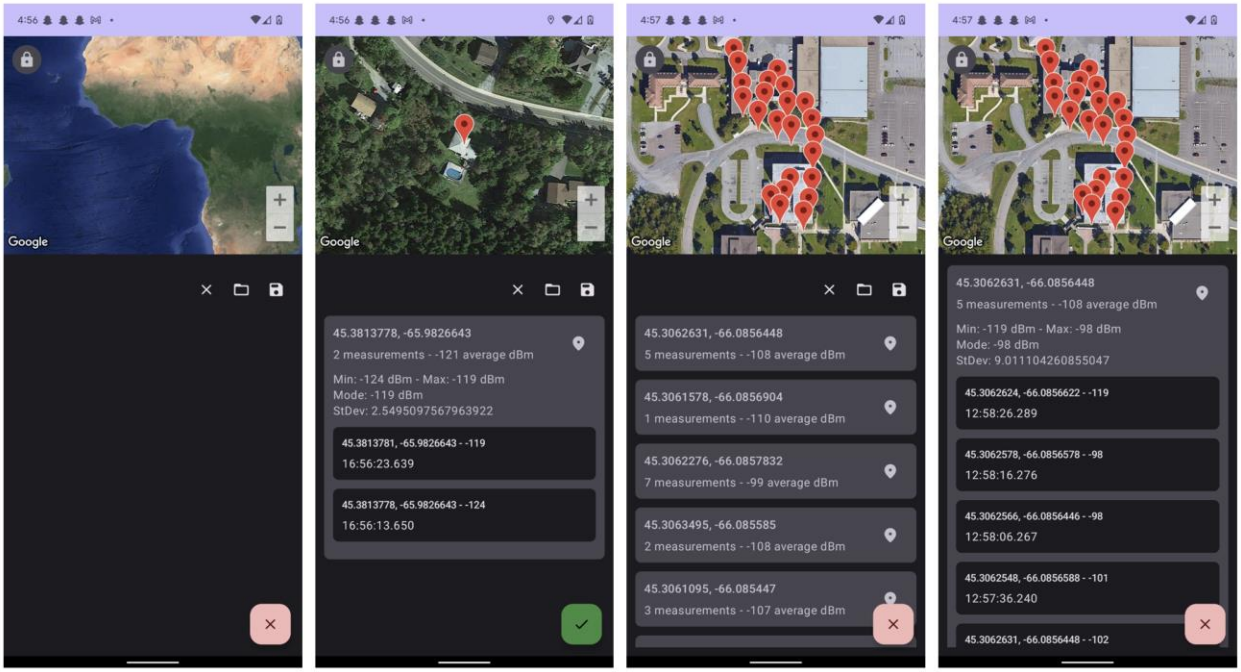
3.3 Data Structure Overview

When the main activity receives a measurement from the background service, it begins to process the new data with respect to the old data in order to determine if the measurement should be stored as a new group of measurements or placed into a pre-existing group. This is done by comparing the new measurement's coordinates to all previous coordinates using the Haversine method of computing the distance between two pairs of coordinates. If the distance is less than or equal to 10 meters away from another pre-existing group, the new measurement is added to this group. By grouping nearby measurements, more accurate measurements can be achieved and

aggregate statistics can be computed. Groups are defined in a data class named "AccumulativeDBMMeasurement", which stores its origin coordinates and a list of independent measurements defined as a data class called "DBMMeasurement". These independent measurements too store coordinates, along with the date and time they were captured and their dbm measurement (an integer).

3.4 User Interface

The user interface of the app exposes controls for taking, analyzing, and viewing measurements. The controls exposed allow the user to start or stop the background measurement service and allow the user to import or export current measurements as JSON objects. The user interface boasts a Google Map with measurements plotted on it as pins, and a list of accumulative measurements. By tapping on an accumulative measurement card, the card expands to show aggregate statistics and a sub list of all independent measurements stored in the group. The following figure shows the various states of the user interface.



3.5 Serialization

Finally, the app allows users to export measurements as JSON objects. This is done by serializing list of all accumulative measurement objects into a JSON file and prompting the user to name the file and store it with a file picker user interface. The reason why JSON was chosen as an output format was that JSON is extraordinarily both human and machine readable, making the resulting data very easy to parse with scripts, such as a Python script, from which it can be easily turned into whatever format the user would like, such as an excel spreadsheet. This data can also be simply loaded by the app at a later date, using the same file picker user interface as before.

4. Testing

In early development, the app was tested using an Android Emulator from Android Studio. This enabled testing the overall service architecture of the app along with the user interface, but due to the emulator having no real GPS or LTE functionality, measurements would feature only “dummy” data. Once the app’s UI and architecture was finalized, testing begun on a real Android Device (Google Pixel 6 with Android 12L) with Rogers connectivity. Testing on the real device included driving to and from the University of New Brunswick campus with the phone in a pocket, which was done to test the GPS functionality and variability of measurements captured over a large stretch of land. Further testing involved having the phone capture measurements at an idle location over night while the user was sleeping. The purposes of this test were two-fold – determine impact on battery life and determine the ability for the app to capture variability in measurements while at an idle location. All tests were successful.

5. Conclusion and Further Work

In this report we discussed the design and implementation of the SignalMapper app, an app which enables users to take in depth and complex measurements of the signal quality of both small and large areas of land, which are automatically taken and plotted on a map using GPS information. Such a tool is very useful for anyone who wishes to survey a geographic location for both overall network quality of service and for discrete fluctuations in signal strength over a period of time or over a geographic plot of land. Such a tool could be improved in the future, though, and the following are recommendation for further development. At the moment, the app takes and processes measurements in such a way that treats signal fluctuations as a function of geographic location. While this is a valid approach, it does not shed much information regarding signal strength fluctuations as a function of time, which in some locations may have a significant effect on quality. Further work should take into consideration the possibility and viability of processing measurements as a time series, with a user interface to match.

6. References

[1] Science Direct. Signal Strength - an overview | ScienceDirect Topics [Accessed: 23-Apr-2022]. https://www.sciencedirect.com/topics/computer-science/signal-strength

[2] Anderson M. Apple Toolbox. iPhone: Viewing Detailed Signal Strength in dBm [Accessed: 23-Apr-2022]. https://appletoolbox.com/iphone-viewing-detailed-signal-strength-in-dbm/

[3] Google. Why Compose | Jetpack Compose. Android Developers. 2022 Mar 11 [Accessed: 23-Apr-2022]. https://developer.android.com/jetpack/compose/why-adopt

[4] Facebook. React – A JavaScript library for building user interfaces. Reactjs.org. 2019 [Accessed: 24-Apr-2022]. https://reactjs.org/

[5] You E. Vue.js. Vue JS. 2000 [Accessed: 24-Apr-2022]. https://vuejs.org/