Smartphones in wireless communication without mobile networks

A study on Android smartphones and XBee radio modules

Dag Roger Stokland Rui

Thesis submitted for the degree of Master in Programming and Networks 60 credits

Institute of Informatics Faculty of Mathematics and Natural Sciences

UNIVERSITY OF OSLO

November 2016
Abstract

Smartphones are among the most popular communication devices there are today. They have the functionality, the flexibility, the prevalence and the economic and technological growth that makes them interesting for new applications. This thesis will focus on the use of smartphones as communication terminals without the use of mobile cellular networks. It explains how smartphones can be used for communication in situations when the mobile network is out of range or not working. These situations include emergency situations in case of hurricanes or flooding, but also less demanding situations where alternative communication may be useful. This thesis compares the current communication technologies available on smartphones and explores the alternatives in order to improve the use smartphones in communication without mobile networks. Finally, the thesis proposes an implementation where a radio module is used together with an Android smartphone to create a wireless messaging system.
Foreword

First of all I want to thank my supervisor Joar Martin Østby. Thank you for making it possible for me to work on this thesis, and thank you for supporting and helping me through these years. Writing this thesis would have been impossible without you.

I would also like to thank my family and my friends for supporting and helping me. Special thanks to Alexander Lorenzo, Hilde Stokland Rui and Odd Vidar Rui for helping me test the application.

Many thanks to everyone else who I has helped me, including Josef Noll, Frank Li, Amol Pophale and the people at Sonen, Institute of informatics, University of Oslo.

Through this thesis I have learned a lot, and I hope it can be of use to someone.
# Contents

1 INTRODUCTION .................................................................................................................................................. 1  
   1.1 MOTIVATION ................................................................................................................................................... 2  
      1.1.1 Situations without network coverage ............................................................................................. 3  
      1.1.2 Why smartphones?.......................................................................................................................... 4  
   1.2 DEVELOPMENT CHOICES .............................................................................................................................. 5  
   1.3 GOALS......................................................................................................................................................... 5  
   1.4 THESIS STRUCTURE ......................................................................................................................................... 6  

2 BACKGROUND ............................................................................................................................................... 7  
   2.1 ELECTRONIC COMMUNICATION ................................................................................................................ 7  
   2.2 WIRELESS COMMUNICATION .................................................................................................................... 8  
      2.2.1 Modulation ...................................................................................................................................... 9  
      2.2.2 Radio frequencies and bands ........................................................................................................ 10  
      2.2.3 Regulations .................................................................................................................................... 13  
      2.2.4 Systems and devices ...................................................................................................................... 14  
   2.3 MOBILE NETWORKS AND EMERGENCY COMMUNICATION SYSTEMS FAILURES ........................................................... 17  
   2.4 WIRELESS MULTI-HOP NETWORKS ........................................................................................................... 18  
   2.5 SMARTPHONE BASED ALTERNATIVE COMMUNICATION ................................................................................. 19  
      2.5.1 GoTenna ........................................................................................................................................ 20  
      2.5.2 Beartooth Walkie-Talkie ................................................................................................................ 20  
      2.5.3 Serval Mesh ................................................................................................................................... 21  
      2.5.4 FireChat ......................................................................................................................................... 21  
      2.5.5 Future applications ........................................................................................................................ 21  

3 TECHNOLOGICAL POSSIBILITIES AND LIMITATIONS .................................................................................. 23  
   3.1 ANDROID ................................................................................................................................................... 23  
      3.1.1 Connectivity................................................................................................................................... 24  
   3.2 ANDROID AD HOC WI-FI ............................................................................................................................... 29  
   3.3 EXTERNAL HARDWARE FOR COMMUNICATION ........................................................................................... 30  
      3.3.1 Challenges ..................................................................................................................................... 30  
      3.3.2 Considered hardware .................................................................................................................... 31  
   3.4 HARDWARE DECISION ................................................................................................................................... 38  
      3.4.1 XBee-PRO DigiMesh 2.4................................................................................................................. 38  
      3.4.2 Android smartphone as USB Host .................................................................................................. 39  
      3.4.3 Arduino as USB Host...................................................................................................................... 40  

4 DEVELOPMENT OF THE XBEE TERMINAL APP .......................................................................................... 41
Figures

Figure 2.1 modulated waveform of FM ................................................................. 10
Figure 3.1: Android USB Host and Accessory modes, from ........................................... 28
Figure 3.2: XBee hardware footprints ...................................................................... 34
Figure 3.3: Network topology in ZigBee ..................................................................... 35
Figure 3.4: Network topology in DigiMesh ................................................................. 36
Figure 3.5: XBee-PRO DigiMesh 2.4 radio module ...................................................... 39
Figure 3.6: SparkFun XBee Explorer Dongle ............................................................... 39
Figure 3.7: Arduino Due microcontroller ................................................................. 40
Figure 3.8: Arduino Wireless SD Shield ....................................................................... 40
Figure 4.1: Illustration of four Android devices connected through a DigiMesh network using the XBee-PRO DigiMesh 2.4 radios. ................................................................. 41
Figure 4.2: Solution for connecting an Android device via the USB port to the XBee-PRO DigiMesh 2.4........................................................................................ 43
Figure 4.3: XBee Terminal startup interface ............................................................... 45
Figure 4.4: XBee Terminal node list interface ............................................................. 46
Figure 4.5: XBee Terminal Chat interface with and without keyboard ....................... 47
Figure 4.6: Remote SMS interface .............................................................................. 47
Figure 4.7: Voice interface – prior to call .................................................................... 48
Figure 4.8: Voice interface - initiating call ................................................................. 48
Figure 4.9: Voice interface - receiving a call ............................................................... 49
Figure 4.10: Voice interface - in call ........................................................................... 49
Figure 4.11: USB to serial connection between an XBee radio and an Android device with the Android device using its USB Host controller ................................................... 50
Figure 4.12: USB to Serial connection between an XBee radio and an Android device using an Arduino Due as the USB Host controller ....................................................... 53
Figure 4.13: XBee Terminal message flow and classes .................................................. 65
Figure 4.14: Remote SMS message flow diagram ....................................................... 66
Figure 5.1: XBee Terminal app performance interface below the voice interface ........... 77
Figure 5.2: Single hop measurements location ........................................................... 79
Figure 5.3: Double hop measurements location ......................................................... 80
Figure 5.4: Single hop in non-optimal conditions distance measurement location. ............... 81
Figure 5.5: Rough overview of the components which contributes to latency in voice communication. ........................................................................................................................................... 83
Figure 7.1: Dual-SIM logger overview interface ........................................................................... 98
Figure 7.2: Dual-SIM logger logging interface ................................................................................ 98
Figure 7.3: Dual-SIM logger about interface .................................................................................. 98
Figure 7.4: Dual-SIM logger example output file imported into Excel ......................................... 98
Tables

Table 2.1: ITU radio bands, taken from the ITU radio regulations articles ............................. 11
Table 2.2: CDMA2000 Coverage comparison, from ........................................................................ 17
Table 4.1: XBee API frame basic format ................................................................................... 58
Table 4.2: Contents of the RF data field in a Transmit Request Frame when a remote SMS is sent ........................................................................................................................................... 67
Table 4.3: Remote SMS status message content ........................................................................ 68
Table 4.4: Speex quality versus bit-rate, from ........................................................................... 71
Table 5.1: Average measured latency in voice communication compared. ............................ 84
Code snippets

Code snippet 4.1: Opening a USB-to-serial connection using the UsbSerial library.................. 52
Code snippet 4.2: Arduino code used for exchanging data between the Android USB port and the XBee UART, using the Arduino Host library. ................................................................. 55
Code snippet 4.3: Pseudo code for buffering XBee frames .................................................. 62
Code snippet 4.4: Sending an SMS with the Android API .................................................... 67
Code snippet 7.1, getting the GSM signal strength on Android devices ............................... 90
Code snippet 7.2: instantiating a Class with Java Reflection example ................................. 91
Code snippet 7.3: Instantiating an object with Java Reflection example .............................. 92
Code snippet 7.4: Invoking methods with Java Reflection example .................................... 92
Code snippet 7.5: Accessing fields of an object with Java Reflection example ..................... 93
Code snippet 7.6: Instantiating the MultiSimTelephonyManager on Samsung GT-S7392 .... 95
Code snippet 7.7: Accessing the mSubscription field in order to listen to SIM card 2 signal strength on Huawei 4X Che2-L11 ......................................................................................... 96
Code snippet 7.8: Invoking the listen method through Java Reflection ............................... 96
1 Introduction

Today there are several communication systems which allows for quick and efficient communication. There are fixed wire networks, such as the telephone network, several mobile networks, satellite networks and the Internet. In addition to these networks, there are more simple systems such as Walkie-Talkies, which allow for two-way wireless voice communication over shorter distances. Compared to the networks mentioned, Walkie-Talkies have a more limited use, but at the same time does not require any sort of infrastructure. Such wireless communication devices are often used when it is necessary to have a way of communicating without, or in addition to the infrastructure based solutions, either for security reasons or for safety reasons. In cases of emergency, mobile base stations may stop working due to external reasons or they may get overloaded by too much traffic. In these cases it is absolutely necessary to have an alternative way of communicating in order to organize help and rescue for the people involved. In lesser demanding situations such as hiking or mountain climbing, an alternative way of communicating could also be a benefit. Some of the radios used in such situations are called CB (Citizen Band) radios, HAM (amateur) radios and Walkie-Talkies.

Communicating using wireless mobile phones is for the most part unproblematic in everyday life. However, in some rural, remote areas, there may be weak or no mobile reception at all. People experienced with weak mobile reception may use a more powerful mobile phone, with a larger antenna and with a signal amplifier to get better mobile reception. In addition to this solution, mobile repeaters can be used to increase the mobile reception in rural areas. Mobile repeaters can be placed within an area with mobile reception, often on hilltops. From there, they can give mobile reception in an area around itself by relaying mobile signals between a mobile phone and a mobile base station. Both of these solutions have their own use cases, but in areas completely isolated from mobile networks and where quick communication is needed, they have no function.

Another solution supported by a few mobile phones is the ability to connect to a more powerful external radio. This radio would not necessarily communicate with a mobile base station, but could communicate with other types of base stations or directly to other radios
of the same type. For many of the newer mobile phones a solution is to connect to an external radio system through the technologies known as USB (Universal Serial Bus) Bluetooth or Wi-Fi. An example of an external radio system is a mobile broadband system \[^1\]. This system offers a local short range wireless network using Wi-Fi and connects to the outer world using mobile technology.

A mobile broadband system would only work in areas with mobile coverage. The idea of this thesis is to use a simple radio which would be able to communicate with similar radios in areas isolated from mobile coverage. This radio would connect to a smartphone and the smartphone would then act as a terminal for communication. By using a smartphone as the terminal for communication it is possible to implement many interesting features by utilizing the hardware the smartphone already has. For example voice communication, by using the speaker and microphone on the smartphone. A few such radio devices have already been developed, such as the Beartooth \[^2\] and the GoTenna \[^3\]. The thesis focuses on how such a radio device would connect and work together with a smartphone.

### 1.1 Motivation

Communication is something we are getting more and more dependent on. We expect our mobile phones to be able to communicate with the rest of the world quickly, reliably and from everywhere. In some situations this presumption may not always be true. A phone call or just a text message could make a huge difference in a dangerous situation.

The needs for communication can be categorized into different groups where we have everything from everyday communication with friends and family, to monitoring systems, to emergency communication. Several of these groups are under development and one of the bigger topics now is the Internet of Things (IoT). IoT consists of sensors, vehicles, buildings and devices interconnected through the Internet. The goal of IoT is to create more efficient and economical environments in for example manufacturing, transportation, medical facilities and homes.
Another system which is common in most countries is an Emergency Communication System (ECS)[4]. These systems are built to alert people of dangerous situations and to efficiently contact help. These systems often support two-way and one-to-many communication and are often integrated between different communication technologies. These technologies can be telephone networks, mobile networks, satellite networks and emergency radios. In addition, emergency calls through mobile phones can often be made through mobile network providers which the user is not subscribed to.

1.1.1 Situations without network coverage

In systems such as an ECS, the IoT or in mobile networks it is important to have a reliable network and an infrastructure that has alternative network routes in case parts of the system stops working. There are however places where it is not economically viable to build communication channels, and there are places isolated from the infrastructure based wireless technologies. This is when we need easy to deploy, infrastructure-less communication systems, such as Walkie-Talkies. This type of communication system may be useful:

- Inside buildings or cellars with dense or reflective walls where there is weak or no mobile reception.
- In rural areas without mobile coverage. This may be in mountains, forests or on the sea.
- In natural disaster situations where infrastructure may break or when the increased communication needs cannot be satisfied by standard communication channels.
- When mobile networks are down due to technical problems.
- In cases with high demands of data that needs alternative channels. This may be the case for mobile computation or video streaming.
- In crowded areas, such as concerts or football stadiums where existing communication channels may get overloaded.

Cases such as natural disasters or mobile network downtime often happen sudden and unexpected. When these situations happen, people do not normally carry radio devices such as Walkie-Talkies. However, on a daily basis, people often carry their smartphones. This thesis proposes to use a radio device which can be connected to a smartphone. This radio
could potentially have a better chance at establishing communication to someone else when there is no mobile coverage.

1.1.2 Why smartphones?
The two motivating aspects of the thesis is the need for communication, together with the possibilities of a smartphone. Looking at mobile communication we see that it is a dominating form of communication with more than 3.6 billion unique subscribers, where 60% of these were counted as smartphone users in 2015\(^5\). From this huge user-base I assume that there is a demand for different ways of communicating with a smartphone. For some people it could be handy to have an alternative communication channel with their smartphone. The benefits of using smartphones as a platform in alternative wireless communication are many:

- Smartphones are very versatile and they are something people will carry with them in everyday life. This makes them excellent when there are sudden emergencies which needs external help.
- They are programmable and can run several custom apps (applications). This allows for access to different hardware components on the phone. Most smartphones can also be connected to external hardware through the technologies know as Wi-Fi, USB (Universal Serial Bus), Bluetooth and NFC (Near Field Communication). Access to the smartphone hardware and any other connected hardware through custom applications makes the usability of the smartphone much better.
- Smartphones has a huge technological and economical momentum. The mobile business keeps adapting to its users by expanding with new and better equipment, and the relationship between functionality and price is getting better and better. Smartphones today often include a GPS (Global Positioning System), a camera, a flashlight and sensors for light, gravity, and motion. Smartphone apps can also be used for generating, processing and displaying images, video, audio and documents.
- Smartphone users are familiar with the user interface of their smartphone. This is an advantage that makes it easier for people to take use of new apps as opposed to having a new device with a different and unknown user interface.
1.2 Development choices
The thesis is based on alternative wireless communication using smartphones. This is a pretty broad subject so some preliminary choices have been made in order to narrow the thesis down:

- Android based smartphones is the platform to develop for. I will explain some of the benefits with Android in chapter 3.
- I will not create a radio. A radio will be bought and my task from there is to connect, program and configure it to be compatible with Android smartphones.

1.3 Goals
The initial goals for the thesis were worked out together with my supervisor Joar Martin Østby. While some of them were possible to achieve, others were not possible and have been adjusted. This is mainly because of legal limitations when it comes to radio frequency usage in Norway, but also because of my limited knowledge of radio technology.

Goals:

- Get a certain understanding of the different situations where alternative communication can be useful, and map the existing technologies. Here alternative communication is anything besides the mobile cellular network and the different technologies can be anything from satellite to Wi-Fi to other types of wireless technology.
- Develop a signal strength logger app for dual-SIM smartphones in order to better be able to map mobile reception coverage.
- Get an overview and compare the different communication technologies in common smartphones. I will look at communication technologies which can be used to communicate directly between smartphones and technologies which can be used to communicate via an external radio.
- The communication technology should be able to offer communication when a mobile network is not present and is required to:
Implement multi-hop communication between connected devices. Examples of such networks are ad hoc networks and mesh networks. These types of networks are explained in the next chapter.

Be able to transfer text messages and speech.

- Develop an Android application which uses the chosen communication technology. The app must have an interface for sending text messages and allow for communication via speech. It should also implement a mechanism for sending messages between connected devices and out on the mobile network.

1.4 Thesis structure
Chapter 2 contains background information on the relevant subjects for the thesis.

Chapter 3 discusses the technical possibilities and problems for smartphones and connected radio hardware.

Chapter 4 is the main part of the thesis where the implementation of the Android app with an external radio is explained.

Chapter 5 contains the measurements and tests done concerning the result of the implementation.

Chapter 6 contains a conclusion to the implementation and measurements. It also contains ideas for future work on the Android app. This chapter ends the first part of the thesis.

Chapter 7 explains the implementation of the dual-SIM signal strength logger app. This chapter is a little separated from the rest of the thesis and is a small project in itself. This chapter has its own discussion and conclusion section.
Chapter 2: Background

2 Background

This chapter gives information on some of the relevant subjects for the thesis. A short history of electronic communication will be presented together with the Public Switched Telephone Network (PSTN) and the Internet. It presents some of the major wireless communication devices and systems there are today and some of the problems with the existing systems. The problems with these systems lead to a need for alternative communication, which the thesis proposes smartphones can help with. Lastly, this chapter will go through some of the existing alternative communication solutions for smartphones and look at possibilities in the future.

2.1 Electronic communication

Through human history, ways of communicating over long distances has been, for the most part, a slow and unreliable process. Messages were for a long time transported by hand, by horse or by ship. In the early 18th century experimentation with electronic communication began, and almost a hundred years later, in the 1830s, the first practical telegraph systems were introduced. These systems used Morse-code to transfer information which required a trained operator. Telegraph lines were laid across large distances all over the world and even across the Atlantic Ocean. Later this technology was developed into something more advanced, the telephone. The Bell telephone company advertised in May 1877, claiming superiority of the telephone over the telegraph:

1. No skilled operator is required, but direct communication may be had by speech without the intervention of a third person.
2. The communication is much more rapid, the average number of words transmitted in a minute by the Morse sounder being from fifteen to twenty, by telephone from one to two hundred.
3. No expense is required, either for its operation or repair. It needs no battery and has no complicated machinery. It is unsurpassed for economy and simplicity.

The invention of the telephone made communication over large distances faster than ever, and with the quick development of the telephone network, it soon became something most people could afford and use in everyday life.\[6\].
Wireless communication using electromagnetic waves was experimented with around the same time in the 1900s. This also started out with communication by Morse-code, but was later developed into voice capable radios. The first commercial radio broadcast was transmitted in Pittsburg 1920 and this technology later developed into modern radio technology such as radio broadcasting and mobile networks\(^7\).

Today we have a worldwide telephone network called the Public Switched Telephone Network (PSTN), which emerged from the many smaller telephone networks built over the years. This is the traditional telephone network most of us use together with a telephone, though today much telecommunication has been taken over by wireless mobile phones. Mobile phones connect to mobile networks and these mobile networks connect to the PSTN, allowing for cross communication between the two. Further, we have the Internet which is also a worldwide network consisting of many smaller networks based on the Internet Protocol (IP). The Internet allows for all kind of communication, such as IP telephony (Voice over IP), text messaging, video streaming and file sharing. The Internet and the PSTN is connected allowing users of IP telephony to make voice calls to telephones and mobile phones and vice versa. Mobile networks are also often connected to the Internet, giving mobile phones access to the Internet. The combined interconnection and accessibility of these technologies has a huge impact on communication today.

### 2.2 Wireless communication

In the context of telecommunication, wireless is descriptive of a network or terminal that uses electromagnetic waves (including radio frequency, infrared, laser, visible light—and acoustic energy) rather than wire conductors\(^8\). Using electromagnetic waves within the radio frequency spectrum is the most common way to communicate wirelessly. Systems using these waves for communication are commonly called radios and are found in everyday devices such as mobile phones, RC cars or TV remote controls.

The very basics of a radio consist of an antenna and at least either a transmitter, a receiver or a transceiver. The transmitter produces an oscillating electrical current of some frequency. It also encodes a message such as voice, image or another medium into the current through a process called modulation. Then the current is transferred to the antenna
which transforms the current into electromagnetic waves. The receiving radio also has an antenna which transforms the electromagnetic waves back into electric current and a demodulator which transforms the current into its original format. A transceiver is a system which can both transmit and receive.

The communication range, data rate and otherwise usability of a radio depend on an array of different things. Some of them include the radio frequencies used, the modulation method used, the power output of the sending radio, the sensitivity of the receiving radio and the environment where the radio waves travel. When looking at radio technology which can be used in this thesis, I have paid special attention to the legal use, range and data rate of each radio. In Norway it can be hard to find a radio optimized in these aspects and the radio would most likely have to be imported from another country. This section will go through different bands and frequencies in radio communication since this is highly connected to range and data rate. It will also go through the legal use of radios and some of the existing systems we have today.

### 2.2.1 Modulation

In radio communication, modulation is a method of varying properties in the waveform of electromagnetic waves in order to make them carry information. Modulation and demodulation, which is the method of extracting information back from modulated waves, can be done in either software, hardware or in a combination of these two. The sending radio radiates a modulated carrier signal which the receiving radio transforms back into its original format through demodulation. An example of a modulation method can be to simply switch the signal on or off each second. The receiving radio can then demodulate the signal by checking if the signal is on or off. This simple example can be used to transfer a string of bits which can be further used to communicate via text. Modern modulation methods are usually much more advanced.

Electromagnetic waves can be modulated on frequency, amplitude, phase or a combination of these. The range, communication quality and data rate of the communicating radios is highly connected to the modulation method. Amplitude modulation (AM) and frequency modulation (FM) is known from its use in radio broadcasting. They are examples of analog
modulation methods. In analog modulation, a baseband signal, for example a Television signal or an audio signal is transformed into a passband signal with different properties which can be transferred wirelessly with an antenna.

In digital modulation, a string or stream of bits are transformed into a passband signal which can be transferred wirelessly. This type of modulation is important in modern wireless systems since error correction and encryption is easier to implement. Digital modulation also tend to have better spectral efficiency which allows for more channels and more users within a radio band\[9\].

![Figure 2.1 modulated waveform of FM \[9\]](image)

### 2.2.2 Radio frequencies and bands

Radio technology is based on the transmission and reception of electromagnetic waves. These electromagnetic waves exist at different frequencies and the frequency spectrum for radio waves ranges from 1Hz to 3000 GHz\[10\]. This spectrum is separated into several radio bands. The bands separate the frequency usage from different services to prevent interference between radios. For example to prevent interference between mobile communication and radio broadcasting. Each of the bands also has a bandplan which defines how each band can be used in order to further avoid interference. An example here is how different mobile network providers allocate their frequencies between each other in the same areas.
2.2.2.1 Radio bands

The bands in the table below are the ITU radio bands described by the International Telecommunication Union (ITU, 2008)\(^{[10]}\). The table displays the frequencies within each radio band and some of its typical uses.

<table>
<thead>
<tr>
<th>Band Number</th>
<th>Symbols</th>
<th>Frequency range</th>
<th>Typical uses</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>VLF</td>
<td>3 to 30 kHz</td>
<td>Radio navigation, submarine communication, radio clock synchronization.</td>
</tr>
<tr>
<td>5</td>
<td>LF</td>
<td>30 to 300 kHz</td>
<td>AM longwave broadcasting, radio frequency identification, maritime mobile.</td>
</tr>
<tr>
<td>6</td>
<td>MF</td>
<td>300 to 3000 kHz</td>
<td>Aeronautical radio navigation, broadcasting, radiolocation.</td>
</tr>
<tr>
<td>7</td>
<td>HF</td>
<td>3 to 30 MHz</td>
<td>Aeronautical and marine mobile, AM shortwave broadcasting.</td>
</tr>
<tr>
<td>8</td>
<td>VHF</td>
<td>30 to 300 MHz</td>
<td>FM broadcasting, television broadcasting.</td>
</tr>
<tr>
<td>9</td>
<td>UHF</td>
<td>300 to 3000 MHz</td>
<td>Television broadcasting, two-way mobile communication, microwaves, Wi-Fi.</td>
</tr>
<tr>
<td>10</td>
<td>SHF</td>
<td>3 to 30 GHz</td>
<td>Satellite communication – mobile, broadcasting, radio astronomy.</td>
</tr>
<tr>
<td>11</td>
<td>EHF</td>
<td>30 to 300 GHz</td>
<td>High speed data links, radar systems.</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>300 to 3000 GHz</td>
<td>Scientific and experimental uses in spectroscopy and high speed data communication.</td>
</tr>
</tbody>
</table>

Table 2.1: ITU radio bands, taken from the ITU radio regulations articles\(^{[10]}\)

The radio bands from ITU are guidelines to what they should be used for. Most often it is the government in each country or area which regulates the specifics of the frequency spectrum usage. The frequency allocation of Norway is provided in Appendix A.
2.2.2.2 Frequencies

The usage of a radio can to some degree be determined by the frequencies it uses. The examples from the ITU radio bands shows that the lower frequency bands are often used for long range communication such as AM and FM broadcasting while the higher frequency bands are often used for high data rate communication such as mobile communication. There are several reasons for why long range is connected to low frequencies and high data rate is connected to high frequencies, but most importantly:

- Lower frequencies tend to be more able to pass through objects such as walls with less attenuation, making the range better[^11].
- Radio waves with a frequency up to 2 MHz has the ability to follow the curvature of the earth (Ground wave propagation) and radio waves with a frequency between 2MHz and 30MHz have the ability to be reflected by the ionosphere (Sky wave propagation), making them able to travel even longer distances[^12].
- For radios to transfer information they need to use a certain frequency bandwidth within the band. Looking at the bands in Table 2.1, there are fewer frequencies in the lower bands. These frequencies are therefore often reserved for special purposes (for example in military, police and medical) which decrease the allowed bandwidth for other purposes. For example, radios operating in the VLF band may only have a few kHz of spectrum available to use. The general idea is that radio bands at lower frequencies are narrower which makes the data rate lower. Higher frequency bands can take use of a larger spectrum which makes the data rate higher.

There are upsides and downsides to whatever frequencies are used in radio communication, though some frequencies are generally more favored. Mobile telephony usually operates in the UHF band and uses frequencies around 900MHz, though the newer mobile 4G technologies can use frequencies around 1800MHz and 2600MHz. These technologies can reach data rates up to 150Mbit/s[^13]. The range is dependent on the surrounding area and the cell towers may be as close as 1 km to each other in urban areas or 30 km apart in rural flat terrain.

Mobile telephony uses a broad spectrum of frequencies to achieve these data rates in comparison to FM broadcasting. FM broadcasting usually operates in the VHF band at
around 100MHz and the data rates in FM broadcasting are less than what mobile technologies can perform. The range of FM broadcasting also depends on the surrounding area and how much energy the radio towers radiate, but these radio waves are intended to travel longer than that of mobile technology.

For this thesis, a radio with the data rate of mobile technology is not needed at all, and thus lower frequencies are preferred in order to get better range. Using frequencies close to the range of FM broadcasting may be ideal since it would allow for voice communication in addition to good range. There are however problems with using these frequencies too, as the next section will explain.

2.2.3 Regulations

Transmitting radio waves when using mobile phones or Walkie-Talkies is usually unproblematic when it comes to legal issues. This is because legal bought radio devices are licensed for its intended use. Altering how the radio works and using it for anything else than its intended purpose is strictly forbidden. Each country has its own laws about frequency usage and this causes some radios to be illegal in other countries. Today, most of the radio spectrum is either licensed for use by government or by companies such as mobile operators and broadcasting channels.

However, most countries also have license free radio bands called the Industrial, Scientific and Medical (ISM) bands.

**Worldwide ISM bands:**

- 13553 kHz to 13567 kHz
- 26957 kHz to 27283 kHz
- 40.66 MHz to 40.7 MHz
- 433.05 MHz to 434.79 MHz. This is only for region 1 defined by ITU and includes Europe, Africa, the Middle East, the former Soviet Union and Mongolia
- 2400 MHz to 2500 MHz
- 5725 MHz to 5875 MHz
- 24 GHz to 24.25 GHz

[10, Chapter 2, Article 5, p. 60 and 65]
These bands can differ in what and how they can be used between each country. For example, there are often limits to the radiated effect, how much energy that can be radiated on a certain frequency within a time period and to the modulation method. In addition, many of the lower frequency bands are narrow-band, meaning they have a small bandwidth which results in a lower data rate. The legal radiated effect on ISM bands tend to be low and since the bands are free to use, interference is likely. Lower radiated effect and interference can mean worse quality of communication, lower data rate and lower range. In other words, there are limitations to what kind of communication can be achieved by using the ISM bands.

Another set of bands that are available to some people are the amateur radio bands. Using these bands require an amateur radio license, which is obtained by showing the necessary skills by passing an examination. The amateur radios has a much higher limit for the energy radiated, up to 1000 Watt in Norway, making them able to communicate over much longer distances than radios using ISM bands. The amateur radio bands available in Norway can be found at[14].

The radio technology required by this thesis would most likely use an ISM band, since these bands are easily available to use. Chapter 3 presents some of the radios within the ISM band considered for use in this thesis.

### 2.2.4 Systems and devices

This section covers some of the major wireless communication systems there are today. It explains what each system is typically used for, including its radio frequencies and radiated effect.

#### 2.2.4.1 Citizen band radio

Citizen band radios (CB radios) are short distance two-way radio communication systems that often utilize the HF band. They became popular in the 1970s and were often used in work applications such as trucking. Today, mobile phone communication has taken over and CB radios are mostly used by hobbyists. Band allocation, channel usage and license
requirements differ from country to country, though they are often similar. For Norway and many other countries, these properties apply to a CB radio:

- Usage of the 27MHz band. In Norway, this band is defined as the range from 26,960 MHz to 27,410 MHz, separated into 40 channels.
- License free to use with a maximum ERP (Effective Radiated Power) of 4 Watt when using AM or FM modulation, and a maximum of 12 Watt ERP when using SSB (Single-Sideband Modulation).\(^\text{[15 p. 2]}\)
- Depending on the line of sight between two communicating CB radios and the quality of the CB radios, they can typically communicate over a distance of 3-15 km.

The channels of CB radios are shared, and only one user can transmit on a channel at a time. In emergency situations where there are many people transmitting, the channels may become crowded and hard to communicate through.

### 2.2.4.2 Walkie-Talkie

Walkie-talkie is a term used for hand-held, portable two-way radios able to send and receive wireless messages. Some common characteristics are two-way voice capabilities, group communication, channel selection, and a push-to-talk button which starts transmission. Over the years, there have been many different implementations of the Walkie-Talkie with features such as text messaging and wide-band radios which can transmit and receive on a broader spectrum of frequencies. It has been used much in military and marine communication, but also by hobbyists and for personal use. Compared to CB radios, the communication can be private, by using encryption. This makes them better where privacy is needed, but at the same time can make it harder to use in emergency situations compared to the CB radios where anyone can listen. The radiated power and frequency used for Walkie-Talkies vary between usage area and country. In Europe, Walkie-Talkies often use the PMR446 (Personal Mobile Radio 446MHz) band with FM modulation.

### 2.2.4.3 Satellite phone

A satellite phone is a good alternative when communication is needed in a remote area. They usually offer text messages and two-way voice communication, in addition to connection to the Internet. In comparison to mobile phones which connect to mobile base stations, these devices connect to satellites in space. Communication through satellites can
go to other satellite phones, to mobile phones and to the PSTN. Satellite phones are ideal in situations where cell towers are not working or in situations without mobile coverage. However, they can be expensive and requires one to buy a service for specific areas or for a global coverage. Satellite reception can also be interfered by weather conditions and the satellite networks can congest in cases of large amounts of communication. The satellites can either have a fixed position in relation to Earth (Geostationary) or moving in orbits (Low Orbit Satellites). The geostationary satellites are located about 36 000 km above Equator and their coverage in areas close to the North and South pole are typically bad. The orbiting satellites are much closer to Earth and have a South to North directed orbit resulting in good coverage of the poles.

### 2.2.4.4 Mobile Networks

A mobile or a cellular network is a network consisting of several interconnected base stations. The base stations provide a wireless network link to surrounding mobile phones, in an area called a cell cite. When a mobile phone is connected to the wireless link it can access the underlying wired network which can be used to access telephone calls, SMS, or the Internet. The base stations are distributed on land to give mobile coverage to large portions of the Earth.

Today, there are several different mobile technologies used, both for the wireless link to the mobile phone, and for the underlying wired network link. Some wireless technologies gives better range, while others gives better data rate. For example, the newer 4G technology LTE (Long-Term Evolution) vastly outperforms the older 3G technologies such as GSM (Global System for Mobile Communications) and CDMA (Code Division Multiple Access) in terms of data rate.

The frequency used in mobile technology may also vary, though frequencies around 900MHz are often used. This makes a huge difference in the mobile coverage of each cell cite site as shown in Table 2.2.
<table>
<thead>
<tr>
<th>Frequency (MHz)</th>
<th>Cell radius (km)</th>
<th>Cell Area (km$^2$)</th>
<th>Relative Cell Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>450</td>
<td>48.9</td>
<td>7521</td>
<td>1</td>
</tr>
<tr>
<td>850</td>
<td>29.4</td>
<td>2712</td>
<td>2.8</td>
</tr>
<tr>
<td>950</td>
<td>26.9</td>
<td>2269</td>
<td>3.3</td>
</tr>
<tr>
<td>1800</td>
<td>14.0</td>
<td>618</td>
<td>12.2</td>
</tr>
<tr>
<td>1900</td>
<td>13.3</td>
<td>553</td>
<td>13.6</td>
</tr>
<tr>
<td>2100</td>
<td>12.0</td>
<td>449</td>
<td>16.2</td>
</tr>
</tbody>
</table>

*Table 2.2: CDMA2000 Coverage comparison, from [16]*

2.3 Mobile networks and Emergency Communication Systems failures

This section quotes and discusses two disaster situations where the mobile networks and emergency communication systems failed.

**Hurricane Katrina**

When Hurricane Katrina, a Category 5 hurricane, hit New Orleans, the emergency communications systems were completely destroyed, including power stations, internet servers, mobile phone towers, and 911 services. The Federal relief workers' satellite phones weren't interoperable, even when they did work. A few AM radio stations were able to continue broadcasting throughout the storm, notably WWL Radio, which remained on the air by broadcasting from a closet. Amateur radio was instrumental in the rescue process and maintained signals when 911 communications were damaged or overloaded. [17]

**New York World Trade Center attack**

During the September 11 attack in 2001, traditional telecommunications were stretched and overloaded. Phone networks along the entire East Coast were congested into uselessness. 911 operators were overwhelmed with calls and could do little more than offer encouragement because of the confusing information they were receiving. Communications between emergency services personnel were limited by a lack of interoperability between departments. Many fire-fighters died when the towers collapsed because they couldn't receive the warning that the police officers received from the New York City Police Department (NYPD) helicopters.
Amateur radio played a large role in facilitating communications between the various emergency departments, which operated on different frequencies and protocols.\footnote{18}

In these cases the ECSs and mobile networks had problems and failed. Communication systems were overloaded or destroyed and people resorted to use amateur radios. Some broadcasting stations where still able to operate, but the lack of two-way communication between people had major consequences.

I believe that if non-infrastructure based radio equipment were more available, some damage could have been prevented. Emergency communication systems could also have benefitted from incorporating more wireless channels and satellite communication should have been more available.

\section*{2.4 Wireless multi-hop networks}

As mentioned in the Goals section of chapter 1, one of the goals is to enable multi-hop networking for smartphones. In traditional wireless networks such as mobile networks and Wi-Fi networks, wireless communication only occurs on a single connection between a base station and an end device. In multi-hop wireless networks, each connected node (end device) can cooperate in relaying messages. This eliminates the need for a fixed base station and can improve network usability. Multi-hop networks can extend further and gives better connectivity than traditional single link networks by giving each node the ability to communicate with any other node connected to the wireless network.

Networks that implement multi-hop connections are often called wireless mesh networks (WMNs) or wireless ad hoc networks. There has been researched and developed several protocols for both network types, but the major difference between them is the usage area. WMNs have a relatively static network topology where most of the relay nodes are in fixed positions. They can be used for sensor networks or for quick deployment of communication networks for larger areas. Ad hoc networks on the other hand, have a dynamic network topology, are designed for high mobility nodes and routing is typically on-demand. Wireless ad hoc networks are typically used in isolated areas, without infrastructure and are often deployed by mobile devices only\footnote{19, p. 6,7}.
WMNs and ad hoc networks suffer from issues such as network scalability and data throughput. In a random topology network, the data throughput becomes lower when the number of nodes in the network increases. Other problems specific to ad hoc networks and WMNs are resource management, reliability, robustness and throughput fairness\(^{[19, p. 8]}\).

These problems will not be discussed in this thesis and I will rather be discussing how one can take use of existing wireless ad hoc network or WMN technologies with smartphones. Ad hoc networks are more relevant for my thesis as smartphones users are often mobile. Ad hoc networks that utilize the existing hardware in smartphones are often called Smartphone Ad Hoc Networks (SPANs), but the general abbreviation used for Mobile Ad Hoc Networks is MANETs.

The next section includes some of the existing MANET or SPAN solutions for smartphones today.

### 2.5 Smartphone based alternative communication

A smartphone is a mobile phone able to perform many of the functions of a personal computer. Compared to a standard mobile phone, the smartphone has an operating system, a touchscreen, a Global Positioning System (GPS), a camera, Internet access capability and much more. Communication is typically done by voice calls or text messages through mobile networks, but can also be done through other wireless technologies such as Wi-Fi and Bluetooth.

There are several apps (applications) for smartphones that can be used to communicate over shorter distances using Wi-Fi or Bluetooth. One example is the Wi-Fi Talkie Free app\(^{[20]}\). Most of these apps works by creating wireless local area networks where one smartphone is the access point of communication. Every other device can connect to the access point and communicates with other connected smartphones through the access point. This means that the range of the network is limited by the range of the connection to the access point.
A few apps such as the Serval Mesh app, allows any smartphone in range of any other smartphone to communicate without a specified access point. Together with a multi-hop protocol these apps can create networks which spans over much larger distances.

There are also solutions which uses external radio hardware. These radios typically connect to a smartphone using Bluetooth, and have point-to-point communication with similar radios. These radios have much greater range than any of the communication technologies in smartphones (except for mobile technology), but I have not been able to find such a radio device which implements multi-hop networking.

2.5.1 GoTenna
Recently a device called GoTenna\[^3\] came on the market which allows smartphones to send and receive messages without the use of the mobile network. It connects to a smartphone via Bluetooth-LE (Low Energy) technology and can be used to communicate via text messages or voice with other GoTenna connected smartphones. The GoTenna is a 2 Watt radio working in the frequency range of 151 MHz to 154 MHz. In perfect conditions the GoTenna can communicate over a 6 km range. The GoTenna is currently only available in the Unites States.

2.5.2 Beartooth Walkie-Talkie
The Beartooth Walkie-Talkie\[^2\] is a wireless communication device similar to the GoTenna. It pairs with a smartphone using Bluetooth, but can be used to charge a smartphone using its internal battery through USB. The radio operates in the frequency range between 902MHz and 928MHz with 1 Watt transmit power. In the United States this is a license free band, but in Norway this band is not available. In optimal conditions the range can be up to 8 km for voice and 16 km for text messages. Compared to the GoTenna, which has higher power output and uses lower frequencies, this device has a higher advertised range. This may be because of the receiver sensitivity or the modulation method for transmission.
2.5.3 Serval Mesh

Serval Mesh\textsuperscript{[21]} is an application for Android devices, developed by The Serval Project team and community\textsuperscript{[22]}. It allows for wireless voice and text communication in addition to file sharing using the Wi-Fi on the Android device. Compared to the Wi-Fi Talkie Free app, Serval Mesh allows for multi-hop communication through Wi-Fi. Serval Mesh makes it possible to deploy expanding mesh networks between smartphones in remote areas without connection to the Internet or to mobile networks.

The range of Wi-Fi on common smartphones is relatively low (I measured about 300 meters), compared to external radios such as the GoTenna or Beartooth. To achieve longer range there has been developed prototypes for a Mesh Extender\textsuperscript{[23]}. The recent prototype device connects to an Android smartphone through USB and can connect to another device in a range of 10 kilometers using a radio operating in the 900-928 MHz band.

The Serval Mesh app and most of the other components such as the assembly guide for the Serval Mesh Extender is open source and available through GitHub\textsuperscript{[24]} and the Wiki\textsuperscript{[25]}.

2.5.4 FireChat

FireChat is an app available for both Android and iPhone, developed by OpenGarden \textsuperscript{[26]}. It uses Wi-Fi and Bluetooth to create multi-hop networks and can transmit text messages or pictures. It became popular in Iraq in 2014 when the government put restrictions on Internet use\textsuperscript{[27]}. Similar to Serval Mesh, the distance between communicating devices is limited by the range of Wi-Fi, but also by the range of Bluetooth. FireChat is not open source, but there is a partner program for the MeshKit\textsuperscript{[28]} module which allows developers to add features from the FireChat app into their own app.

2.5.5 Future applications

Currently, there are already a lot of choices when it comes to alternative communication on smartphones. The alternatives using the internal Wi-Fi chip is limited by low range and the alternatives using an external radio is limited by single-hop communication. Using the mobile radio for direct communication between smartphones is also very restricted, though this is most likely not a technical restriction.
LTE Direct\textsuperscript{[29]} is a device-to-device technology which uses the mobile radio to connect mobile phones. It has an estimated range of 500 meters and is focused around low battery usage, and communication in high density areas. In the future, it would be interesting to see if direct communication between mobile phones using the mobile radio could improve. With improved range in an emergency situation, this technology could make a huge difference.

The range of Wi-Fi on smartphones is quite low due to the low power consumption required by smartphones. Currently, the only real benefit with Wi-Fi in communication in isolated areas is the multi-hop ability as seen in the Serval Mesh app. In order to benefit more from this, the range of smartphone Wi-Fi must increase substantially.

As for external radio hardware, the current limitations are for the most part legal radio spectrum usage. Both the Beartooth Walkie-Talkie and the GoTenna uses lower frequency bands which is better for achieving longer range communication compared to the 2.4GHz used by Wi-Fi. The frequencies used by the Beartooth Walkie-Talkie and GoTenna are for the most part limited to America. Worldwide ISM bands around 433MHz and 40MHz does not currently have any solution as an external radio device for smartphones, so using these bands could be an idea for the future.
Chapter 3: Technological possibilities and limitations

The goal of this chapter is to explain my process and findings in the search of a way to communicate between Android smartphones without the use of mobile networks. To solve this I have looked into the two primary approaches:

- Communication directly between smartphones by using any of its internal radio hardware, and
- by using external radio hardware which connects to the smartphone.

This chapter will explain the communication technologies that are available for Android smartphones and discuss some of the ways that makes the two approaches possible. Using internal hardware on smartphones, such as Wi-Fi, for direct communication is something that is easily achievable today, and my thesis will rather focus on how one would implement multi-hop networks for smartphones. Section 3.1 will give a brief explanation on the relevant communication technologies in standard Android smartphones. Section 3.2 will go through possible implementations using internal the internal Wi-Fi chip on Android smartphones and section 3.3 will go through possible implementations using external hardware.

3.1 Android

The Android operating system is developed by Google and is the most popular operating system for mobile devices. Android has an open source project\(^{[30]}\) which can freely be downloaded and customized to suit different needs for different devices. This has caused Android to be a popular operating system choice for smartphones and other mobile devices.

Android is based on the Linux kernel and has an API (application programming interface) for most of the functionality found in the Linux kernel, in addition to the official Android API. When developing apps (applications) for Android, access to some functionality is restricted. This is mainly to protect users from breaking their phone and to protect against malicious software. For example it is normally not possible to access the low level settings of the Wi-Fi chipset through an app. These settings are instead configured when the device is manufactured or can be handled through a system app. In order to access this functionality in a standard app, system privileges are needed. Parts of my thesis require some of this
functionality and section 3.2 and chapter 7 will explain why and how. The method of gaining system privileges is often called rooting in Android.

Despite restrictions to some functionality I chose to get familiar and develop for Android early on. Android has the benefit of making it possible to freely develop an app using the USB port, contrary to the iOS operating system where a license is needed. The popularity, community and available information made me choose Android as the platform to develop for.

### 3.1.1 Connectivity

In addition to mobile network connection, Android smartphones usually have communication technologies for Wi-Fi, Bluetooth, NFC (Near Field Communication) and USB. NFC is limited to a few centimeters of wireless range, and is not supported on all devices. NFC is therefore not further discussed in this thesis.

Bluetooth, Wi-Fi and USB can be used to directly connect smartphones to share files and send messages using apps. USB is a wired connection with limited use in this kind of direct communication and Bluetooth range is relatively short. However, all of the above technologies can also be used to connect to an external radio. This radio would be dedicated towards long range communication to similar radios. Bluetooth and USB are more relevant in this case since the external radio would be close to the smartphone. This section will discuss if and to what degree these technologies can be used for the thesis.

#### 3.1.1.1 Wi-Fi

Wi-Fi is a Wireless Local Area Network (WLAN) technology which is present on most Android smartphones. It is often used to connect to a Wi-Fi access point (AP), often called a Wi-Fi hotspot, which gives access to a local area network and often to the Internet.

Wi-Fi is great when high data speed is needed for voice or video transmission. It has the highest range between the wireless communication technologies on Android (except mobile technology). Wi-Fi is also very standardized, meaning it can communicate with other types of Wi-Fi enabled devices. Wi-Fi would allow for connection to external radio hardware through
easily available Wi-Fi chips (for example\textsuperscript{[31]}), but can also be used to communicate directly between smartphones.

Wi-Fi is based on the Institute of Electrical and Electronics Engineers (IEEE) 802.11 standards. The IEEE 802.11 standards have two basic modes of operation: infrastructure mode and ad hoc mode.

- Infrastructure mode is most commonly used, as it is more secure than ad hoc mode. In Infrastructure mode, clients connect wirelessly to an access point giving access to other connected clients and often to the Internet. A wireless AP supporting one or multiple clients is known as a Basic Service Set (BSS)\textsuperscript{[32]}.

- In ad hoc mode there is no need for an AP and clients can communicate directly to each other. There is no central control and clients communicate peer-to-peer. Ad hoc mode can therefore be utilized to create multi-hop networks between devices. Ad hoc mode can be enabled using the Independent Basic Service Set (IBSS)\textsuperscript{[32]}.

Going further into Infrastructure mode, Wi-Fi communication on Android devices can be done in two different ways:

- Some Android devices can act as a Wi-Fi access point allowing Wi-Fi devices in client mode to connect. The Wi-Fi access point creates a local area network where clients can communicate to the access point and through the access point to each other. Wi-Fi access point is supported on Android 2.2 and higher and client mode is support on Android version 1.0 and higher. However, there are a few Android smartphones of version 2.2 and higher which not have the hardware to support Wi-Fi access point mode.

- Android devices of version 4.0 and higher supports the Wi-Fi Peer-to-Peer (P2P) framework. Wi-Fi P2P complies with the Wi-Fi Direct\textsuperscript{[33]} standard and enables all Wi-Fi Direct devices to connect to each other without a dedicated access point. Instead of a dedicated access point, devices who want to form a network cooperate in choosing a group owner. The group owner has similar responsibilities compared to an access point, as the group owner also routes data and maintains the network. The Wi-Fi P2P group owner can be described as a software access point\textsuperscript{[34]}.
Both Wi-Fi AP mode and Wi-Fi P2P essentially gives a developer the same type of networks to work with, where there is a central access point routing all data. Clients connecting to an access point or a group owner cannot connect to other access points or group owners simultaneously, which limits the network to be local with no multi-hop capabilities.

Currently, only infrastructure mode is enabled on Android smartphones. There are some WLAN chipsets that supports both modes, but Android only provides an API for infrastructure mode. Section 3.2 presents how ad hoc mode can be enabled and used to create multi-hop networks.

The maximum communication distance using Wi-Fi on Android smartphones depends on several variables such as the quality of the WLAN chipset, its ERP (Effective Radiated Power) and the environment. I measured the Wi-Fi line-of-sight connection range between two newer Android devices (Samsung Neo S5 and Huawei Honor 4X) to be about 290 to 310 meters.

### 3.1.1.2 Bluetooth

Bluetooth is another wireless technology integrated in most Android smartphones. Bluetooth can be used for point-to-point and point-to-multipoint communication to other enabled Bluetooth devices, similar to Wi-Fi. It is commonly used for wireless file sharing, hands-free voice calls and wireless music streaming.

Bluetooth range is short compared to Wi-Fi. Simple measurements show an average range of 10-12 meters before disconnecting. This makes it unsuitable for direct communication to other mobile devices. On the other hand, it can be used to communicate with an external radio.

Android of version 4.3 and higher also supports Bluetooth Low Energy (BLE) which in comparison to classic Bluetooth is designed to use significantly less power. BLE is often used in applications where it is important to send small amounts of data periodically. This makes Bluetooth very relevant in communication to an external radio as both the smartphone and the external Bluetooth chip could use less power in standby mode.
Bluetooth communication is very standardized, it works between an array of devices and is easy to program and take use of on Android devices. Bluetooth chips are also easily available\[35] to use with an external radio.

### 3.1.1.3 USB

The USB port on Android smartphones is commonly used for charging the device and for transferring files to and from a PC. It also supports for interaction with other types of hardware such as cameras, mice and headsets. USB is a wired technology and this section will therefore only discuss its usability in communication with an external radio.

A standard in USB is that the interaction between two USB devices must always be between a master and a slave. When an Android device acts as an USB mass storage device or as a multimedia device connected to a PC, the Android device will act as the slave and the PC will act as the master. In custom communication to other types of hardware, communication will also be between a master and a slave. In Android these two modes are respectively called USB Host mode and USB Accessory mode. Section 3.3 will review hardware that enables USB communication between an Android and an external radio. The rest of this section will briefly explain how the two modes work on Android devices.

**USB accessory mode**

When the Android device is connected to an USB master device, the Android device will act as in USB Accessory mode\[36]. In this mode the connected USB master device initiates and controls communication and must power the Android device with up to 500mA at 5V. The USB host device must also implement the Android Open Accessory (AOA) protocol according to the Accessory Development Kit (ADK) \[37], in order to communicate with an Android device in USB Accessory mode. USB Accessory mode is supported on Android devices of version 2.3.4 and higher.

**USB Host mode**

Android devices which supports USB Host mode\[36] can initiate and control communication with other USB slave devices. Similar to a USB port on PCs, Android devices with USB Host can connect to USB mass storage devices, USB speakers and to HID(s) (Human Interface
Devices) such as a mouse or a keyboard. As in standard USB, the Android USB Host device powers the connected USB slave device with up to 500 mA at 5V. Host mode is supported on Android of version 3.1 and higher, but requires an USB Host controller (hardware) which is often not included on Android smartphones. An Android device with USB Host functionality is often labeled by retailers to have USB OTG (On-The-Go) enabled.

Whether each USB mode is available on a device depends on the implementation of the Android smartphone. Host mode is only included on a few smartphones and these are generally more expensive than other smartphones. Accessory mode should always be included, but I have found that some Android devices have connection problems in Accessory mode. These problems are shortly explained in the next chapter (Section 4.10).

![Android USB Host and Accessory modes](image)

3.1.1.4 Discussion

Compared to Wi-Fi and Bluetooth, USB has a major advantage when using the USB Host mode. Android devices with USB Host mode can power an external radio and requires generally less and smaller external hardware when connecting to an external radio. USB Accessory mode is comparable to both Wi-Fi and Bluetooth as it requires an external battery and more external hardware for communication between the Android device and the external radio. Data rates for USB are generally higher than both Bluetooth and Wi-Fi, but
Chapter 3: Technological possibilities and limitations

this depends on the hardware of the Android device and also the external hardware (USB controller, Wi-Fi chip or Bluetooth chip) used between the external radio and the Android device.

3.2 Android ad hoc Wi-Fi

As explained in the Wi-Fi section (Section 3.1.1.1), direct communication between two Android devices using Wi-Fi can easily be achieved by having one device in AP mode and another device in Client mode. A more interesting approach is to use the Wi-Fi ad hoc mode which is available on some Android WLAN chipsets. By default, ad hoc functionality is not enabled on Android devices and there is no API to access and enable ad hoc mode. This issue has been discussed for a long time and a petition to include an API for ad hoc Wi-Fi started in 2008\cite{38}. The petition is now closed and I have not been able to find more information about including an ad hoc Wi-Fi API in Android. Despite this, there are several apps that allows for some sort of multi-hop networking.

Serval Mesh has two solutions to enable multi-hop Wi-Fi networking. The first method is by enabling ad hoc mode on the WLAN chipset through an Edify script\footnote{Android can install over-the-air (OTA) updates to the system and application software through the Internet. This updater contains an interpreter for the extensible scripting language Edify, which in this case is used to enable Wi-Fi ad hoc mode.} \cite{39}. This only works for devices with WLAN chipsets that already has functionality for ad hoc mode. Further, these scripts are reserved for system applications and require system privileges to be used from standard applications. Enabling Wi-Fi ad-hoc mode has also been done using Java Reflection\footnote{A popular modification of the Android OS which enables ad-hoc mode through Java Reflection is called CyanogenMod\cite{41}, but again, installing this modification requires system privileges.}. A popular modification of the Android OS which enables ad-hoc mode through Java Reflection is called CyanogenMod\cite{41}, but again, installing this modification requires system privileges.

Rooting an Android device in order to obtain system privileges is not something everyone would do, since it breaks the device warranty. Serval Mesh has therefore introduced a second solution. Multi-hop network capabilities are achieved by cycling between Wi-Fi Access point mode and Client mode, combined with a store-and-forward protocol\footnote{This solution has a few downsides such as high power consumption, lack of real-time}. This solution has a few downsides such as high power consumption, lack of real-time
Chapter 3: Technological possibilities and limitations

communication, and no end-to-end connection over many hops. They have however implemented text messaging and file sharing using this method.

The FireChat application does not need a rooted phone in order to do multi-hop networking. There is no available information to how they achieved this. However, they are using both Bluetooth and Wi-Fi. A personal guess would be that the smartphones sets up a network by alternating between AP mode and client mode for both Bluetooth and Wi-Fi.

These are the only methods for multi-hop communication using Wi-Fi for Android devices I have been able to find. There is however a few other implementations for multi-hop networking using these methods mentioned.

3.3 External hardware for communication

Early in the thesis I decided to go for an implementation using external radio hardware which connects to an Android device through a USB cable. Important properties required by the radio are good range, multi-hop networking capabilities and small size. This section will go through the different hardware modules which were considered, together with some options which are not available in Norway.

3.3.1 Challenges

- Finding a radio with a data rate high enough to be capable of communicating via speech.
- The radio must be legal to import and use in Norway concerning radio spectrum usage.
- The radio must, in some way, have multi-hop networking capabilities. This can be implemented on a radio which does not natively have multi-hop capabilities but it should be preferred to find a radio which natively supports multi-hop.
- I choose to use USB as the communication medium between the Android device and the radio. This requires the radio to be able to communicate with the Android device through USB.
3.3.2 Considered hardware

This section and its subsections will go through the different hardware I considered to use with a radio module. Interaction between a radio module and a smartphone was a difficult task, and to make this easier I decided to use a programmable microcontroller to act as the bridge between the radio module and the smartphone.

3.3.2.1 Microcontroller

A microcontroller can be described as a small, programmable computer with a processor, memory and input and output peripherals. Usually, microcontrollers do not run an operating system and are instead used to control and manage simple tasks. They are often used in small electronic devices such as toys, remote controls, engine control systems and other embedded systems. For this thesis, it is ideal to have a small microcontroller, able to be programmed for USB communication with an Android device and for communication with a radio module. This microcontroller would act as the bridge between the radio module and the Android device.

3.3.2.1.1 Arduino

Arduino\(^{[43]}\) is an open-source prototyping platform focused on easy-to-use hardware and software. Arduino microcontrollers are able to interact with an array of different hardware components such as LEDs, motors, PCs and radios. To use more advanced hardware with the Arduino, a so called shield can be used as an adapter between the Arduino and other hardware, such as a radio module. The shields can be mounted directly on top of an Arduino and makes it easy to connect the two hardware pieces together. Arduino shields include USB Host Shields, Servo Shields and Wireless Shields for Wi-Fi and Bluetooth chips.

Most of the Arduino boards can be powered by a USB connection with 5V or by a 2.1mm power plug with a recommended voltage between 7V to 12V. The boards can therefore either be powered by a smartphone with USB host or by a standard nine-volt battery. Input and output on Arduino boards can be through both digital and analog pins. There are also more specialized pins like a reset pin, an interrupt pin and serial communication pins, known as UART (Universal Asynchronous Receiver Transmitter) ports.
Arduino has several boards where some are more capable than others. This is short summary and comparison between the boards which were tested and considered:

- **Arduino Uno**

  Arduino Uno[^44] is one of the smaller boards from Arduino and is based on the ATMega328P microcontroller from Atmel. It has one USB port which can be connected to a computer for programming or connected to an Android USB port for serial communication (USB-to-Serial communication). USB master capability is not supported on the Arduino Uno, but it can act as a USB slave. It has one input and one output pin for serial communication (one UART port) and several analog and digital pins which can be connected to a radio module.

- **Arduino Due**

  Arduino Due[^45] is based on the Atmel SAM3X8E ARM Cortex-M3 CPU. It has more pins and memory than the Arduino Uno and unlike other boards it runs at 3,3V. It can be powered at up to 12V from the power plug or 5V from the USB port. It has two USB ports: the programming port and the native port. The programming port can be used for programming or for serial communication in USB slave mode. The native port can also be used for programming and has USB master functionality. This means it can act as a USB Host to a connected Android smartphone. The Arduino Due has 4 UART ports.

- **Arduino Mega ADK**

  The Arduino Mega ADK[^46] board is based on the ATMega2560 microcontroller. This board also has built in USB Host capabilities and was specifically designed to communicate with Android devices over USB. Similar to the Arduino Due it has two USB ports, one for programming and serial communication (USB slave) and one USB master port. It can easily be connected to a radio and a smartphone at the same time through 4 UART ports and several analog and digital pins.
3.3.2.1.2 PandaBoard

PandaBoard is a small single-board computer, able to run operating systems such as Ubuntu or Android. Similar to Arduino, it has input and output peripherals for interaction with other hardware, but has more processing power and memory than the Arduino microcontrollers mentioned. For my thesis, this computer is a slight overkill. However, for more advanced applications where for example a routing protocol for a large network is needed, the PandaBoard may be ideal. Other benefits with the board are built-in Wi-Fi, Bluetooth, an Ethernet port and USB Host capabilities with Linux drivers. The PandaBoard was recommended by Professor Frank Li at the University of Agder in Norway, but as it is larger and more expensive than an Arduino microcontroller I choose to not use it.

3.3.2.1.3 Raspberry PI

Raspberry PI is a series of single-board computers similar to the PandaBoard. Raspberry PI Zero is the smallest in the series and is smaller than the Arduino Uno. It can run the Raspbian operating system, has USB Host capability, has more powerful hardware and is cheaper than the Arduino Uno. Without testing the Raspberry Pi Zero it may have been the ideal microcontroller to use in this thesis, but for the term of the thesis I was not able to acquire one.

3.3.2.2 Radio modules

This section describes the radio modules I considered to use for the thesis. Because of the relatively few license free radio bands available, I have only been able to test radios on the 2.4 GHz ISM band.

3.3.2.2.1 XBee

XBee is a brand name for radio modules developed by Digi International. The XBee radio modules are simple to use, and can connect to a microcontroller through its UART port. Most XBee modules can be used with a minimum connection of a 3.3V power supply, ground outlet, and the UART port. XBee modules come in two different hardware footprints (Figure 3.2), and the format of serial communication between XBee modules is often similar or identical. This makes them easy to switch out with different modules with different properties in range, data rate, modulation scheme and networking protocol.
Different XBee radio modules support different radio frequencies and different networking protocols. The frequencies are in the ISM bands: 868 MHz, 900 MHz and 2.4 GHz. A few protocols are proprietary, but the ones listed are:\(^{52}\):

- IEEE 802.11 for Wi-Fi connection in infrastructure mode.
- IEEE 802.15.4 for point-to-multipoint network topology, similar to Wi-Fi.
- ZigBee for mesh networking based on the IEEE 802.15.4 specification.
- DigiMesh for mesh networking. This is a proprietary protocol.

For this thesis the ZigBee and DigiMesh protocols are the most relevant as they implement mesh networking.

**XBee-PRO ZigBee 2.4**

The XBee-PRO ZigBee 2.4\(^{53}\) is the first radio module I tested. It uses the same 2.4GHz ISM band as Wi-Fi, but has substantially higher range. The maximum transmit power is 63 mW and the advertised range is 3.2 km in optimal conditions and 90m indoors. The maximum transmission data rate 250kbps.

This module uses the mesh protocol ZigBee which is designed for low data rate, low power applications such as sensor systems. The ZigBee protocol uses 3 types of network nodes: Coordinator, Router and End Device. Each node has a different function in the ZigBee network.
ZigBee node types:

- The Coordinator is the most capable node in the network. It can relay messages between other nodes and is the node which establishes the network. There must be exactly one coordinator node in each ZigBee network.
- As the name implies, the Router nodes relay information between other nodes. Routers can also send and receive messages.
- End Device nodes can only communicate through a Coordinator or a Router and does not relay information for other nodes. Their reduced functionality allows them to use less power.

This type of topology is a problem for use in this thesis, as it is impossible for two Coordinator nodes to communicate. Configuring a node to work as a Coordinator prior to communication is not preferred. If two Coordinator nodes were to communicate, one node would have to switch to End Device or to Router before any kind of communication could happen. After testing the modules it was apparent that it would not suit the thesis since switching node type needs to be done by uploading firmware from a PC. In ZigBee every node needs to be programmed for its intended purpose (as a Coordinator, Router or End Device) and is better for networks where the network topology is fixed and the Coordinator is always present. My thesis required a network where every node can connect to any other node without the tree structure of ZigBee.
XBee-PRO DigiMesh 2.4

The XBee-PRO DigiMesh 2.4 radio module\(^{[55]}\) is similar to the XBee-PRO ZigBee 2.4 in that it uses the 2.4 GHz ISM band, has 63 mW power output and an indoor communication range of 90m. Line of sight range is advertised to 1.6 km. The lower range is one disadvantage, but the major advantage is that it runs the DigiMesh mesh protocol for wireless transmission.

In DigiMesh there is only one node type. Every node can relay messages between other nodes and can also send, receive and process messages itself.

![DigiMesh Nodes](image)

*Figure 3.4: Network topology in DigiMesh\(^{[54]}\)*

The benefit with this homogenous mesh topology is the simple network setup. No configuration or firmware update prior to connection is needed and any node can establish and join a DigiMesh network at any time.

XBee-Pro 900HP

Similar to XBee-Pro Digimesh 2.4, the XBee-PRO 900HP module\(^{[56]}\) can also use the DigiMesh protocol. This is a high powered module which can transmit at a range of up to 14 km at 10 Kbps, and 6.5 km at 2000 Kbps. It uses the frequencies from 902-928 MHz and has a transmit power of 250 mW. This is one of the most powerful radios in the XBee series and would be the best choice of radio module for the thesis if it was available and legal to use in Norway.
3.3.2.2.2 *HamShield for Arduino*

The HamShield\(^{[57]}\) is a wireless UHF/VHF radio that can be mounted on an Arduino like a shield. It can use the frequencies: 134-174 MHz, 200-260 MHz and 400-520 MHz. Within these bands are the Amateur Radio bands 144-148 MHz, 220-225 MHz and 420-450 MHz, in addition to the ISM band near 433 MHz. It uses Frequency Modulation (FM) to transfer audio, but can also transmit Morse-code to encode text messages. This radio does not support any sort of multi-hop networking like the XBee counterparts, but has the benefit of being able to transmit and receive on a much larger frequency spectrum. Multi-hop networking can also be achieved by implementing a mesh or ad hoc protocol on top of the existing radio transmission protocol. In addition, since it uses FM modulation, it can communicate with other types of Amateur Radios.

I have not been able to find any range measurements for this radio, but with a transmit power of 250 mW and the fact that it uses lower frequencies than any of the XBee modules, the range should be comparable to the XBee 900HP. HamShield started shipping in March 2016 and since I do not have a Radio Amateur license, I was not able to get this device in time. It could however, be a very good alternative for a future application.

3.3.2.2.3 *RockBLOCK mk2*

The RockBlock mk2\(^{[58]}\) is a satellite modem for the Iridium satellite network. Compared to the other radios described, this device is infrastructure based, with a global coverage. The downsides are slower transmission times (up to 20 seconds for a single message), larger size and has a running cost for messages received or sent.

Messages sent or received from RockBLOCK are transferred through the Internet, via an email-address or with HTTP (Hyper Text Transfer Protocol). It can be connected to an Arduino through its UART port.

The RockBLOCK may be interesting for applications requiring global coverage, but for general satellite communication using smartphones, I believe there are better alternatives. For example the Iridium GO\(^{[59]}\) satellite modem which creates a Wi-Fi hotspot smartphones can connect to.
3.3.2.2.4 Software Defined Radios

A Software Defined Radio (SDR) is a general term for a radio system where some of the hardware components (modulators, amplifiers etc.), that are typical to common radios are instead replaced by software. SDRs can then be programmed for different communication protocols, modulation schemes, frequency spectrum usage and more. They have the major benefit of being able to be programmed to be able to communicate with other types of radios. The downsides are that they are generally large or expensive. The smaller radios tend to only be a receiver and the power output of the transceivers is often lower than other types of radios. As any radio its legal use must comply with the rules and laws in each country.

A comprehensive list of SDRs from 2014 can be found here[^60]. The cheaper SDR transceivers cost around 300 US dollars and are generally larger than other radios. For this thesis, using an SDR would make it possible to communicate with other radios, but the price and the size would be too much for use in this thesis.

3.3 Hardware decision

First of all, I wanted to use USB as the communication medium between the radio and the Android device. The main benefit with USB is when the Android device has a built-in USB Host controller. This eliminates the need for an external USB Host controller and the external radio can be powered by the Android device. Since not all Android smartphones has a USB Host controller, I have implemented a second solution for Android devices using USB accessory mode. This resulted in two different hardware solutions and additional programming for each USB mode.

3.3.1 XBee-PRO DigiMesh 2.4

The radio was the most important choice for the thesis. An XBee module seemed like a good choice since they are easy to replace with other XBee modules with different wireless properties. The XBee-PRO DigiMesh 2.4 was cheap and easy to acquire in Norway. It has built-in mesh networking and the data rate is high enough for voice transmission.
3.3.2 **Android smartphone as USB Host**

With the Android smartphone as USB Host, the only hardware needed for communication to the XBee module is a USB-to-Serial converter. For this purpose I used a SparkFun XBee Explorer Dongle[61]. The XBee radio can be attached directly to the SparkFun XBee Explorer Dongle which connects to the Android smartphone via an USB OTG (On-The-Go) cable.
3.3.3 Arduino as USB Host

Since I decided to use USB as the communication medium between Android and XBee, there were not many options for the microcontroller. USB communication with Android devices in USB accessory mode needs to connect to a USB Host controller which implements the AOA (Android Open Accessory) protocol. Arduino is one of the few microcontrollers which have an implementation for AOA. Both Raspberry PI and PandaBoard have a few implementations for the AOA protocol, though they are unofficial and I have not been able to find much information on either of these. The Arduino Due is the most recent microcontroller from Arduino which has a USB Host controller together with a software library for AOA. Therefore, I decided to use an Arduino Due (Figure 3.7) together with an Arduino Wireless SD Shield (Figure 3.8) as the bridge between the Android device and the XBee module.

Figure 3.7: Arduino Due microcontroller

Figure 3.8: Arduino Wireless SD Shield
Chapter 4: Development of the XBee Terminal app

This chapter presents and explains some of the key-points in developing the XBee Terminal app for Android devices. The main concept of the XBee Terminal app is to utilize the XBee-PRO DigiMesh 2.4 radio modules for wireless transmission of text messages and voice.

4.1 The XBee Terminal app

Wireless communication is done using an XBee radio module connected to the Android device via the USB port. This XBee-Android pair is referred to as a node in this chapter. The app allows two or more nodes to:

- Send broadcast or unicast messages to each other.
- Communicate via text messages and via voice.
- View other nodes and XBee radios connected to the network.
- Send an SMS through the network and out on a mobile network.

The XBee radio itself forms a so called DigiMesh network with other nearby XBee modules. The XBee Terminal app interacts with the radio module by giving a user interface for text messages and speech. Figure 4.1 illustrates how the app is used with an Android device to communicate through a DigiMesh network.

Figure 4.1: Illustration of four Android devices connected through a DigiMesh network using the XBee-PRO DigiMesh 2.4 radios. Each Android-XBee pair can send, receive and relay messages through the network. The XBee radios relay messages independent from a connected Android device (as seen in the center XBee radio).
The next section gives an overview of hardware and software in the system. Section 4.3 explains some basic concepts in Android programming, taking Java programming as the standpoint. Section 4.4 displays the user-interface and explains some details in the implementation of the user-interface. The following sections 4.5 - 4.6 explain implementation concerning communication between XBee radio and Android device. Sections 4.7 – 4.9 explains implementation details about general data management and implementation for voice communication and remote SMS.

4.2 System overview

The XBee Terminal system consists of 3 parts or layers of hardware and software. Each part will be explained in detail in the following sections. However, I first want to give an overview of how each part interacts with each other.

1. At the bottom layer is an XBee-PRO DigiMesh 2.4 radio. This is a digital packet-based radio with mesh networking capabilities. The radio is ad hoc, meaning it can relay packets received to effectively create a DigiMesh network. Messages can be sent through the DigiMesh network by providing, among other parameters, the destination address of the receiving radio and the message itself. Providing these parameters to the XBee radio is done via its UART (Universal asynchronous receiver/transmitter) port.

2. At the next layer is a USB-to-serial converter used to connect the UART of the XBee radio to the USB port of the Android device. This connection has two solutions as seen in Figure 4.2: One where the Android device is in USB host mode (left) and one where the Android device is in USB accessory mode (right).

3. At the top layer is the Android application which accesses the USB port and communicates with the XBee radio via the USB-to-serial converter. The app has a user interface for interacting with the XBee radio in various ways.
4.3 Android programming

The XBee Terminal app is programmed using the Android SDK (Software development kit) and the Android Studio IDE (Integrated development environment). The application is written in the Java programming language, but the Android API (Application Programming Interface) is in many ways different from the standard Java API. This section explains some of the differences between the two.

4.3.1 Activities and Fragments

A core aspect in Android programming is the use of Activity and Fragment classes. The Activity class is the entry-point (similar to the main method in Java) to the app and contains lifecycle methods which are invoked at certain events. These events can be for example when the application is started, when the application pauses (application goes to the background), when it resumes, when the user interface is loaded or when the user closes the application. Fragments also contain these lifecycle methods, and Fragments are always a level below the Activity that controls it. They are often used to control parts of the user interface or to separate the functionality of an Activity into smaller pieces. In this application there is one Activity and several Fragments used to switch between different user interfaces.
4.3.2 AndroidManifest.xml

Android apps also need an AndroidManifest.xml file which contains essential information about the app. Most importantly it contains a declaration for the application and a declaration for one or more Activities. Usually one Activity is declared as the main Activity which should launch when the user launches the application. An activity may also be launched in different ways, such as when a USB device is connected or when another application launches it. In addition, the AndroidManifest.xml needs to declare what kind of features and permissions the app needs from the Android operating system. In the XBee Terminal the features used are USB host functionality and USB accessory functionality. It also needs permission to send SMS and to record audio. Playing audio does not require a permission.

4.3.3 Intents and intent-filters

Intents are used to send messages between the Android operating system and Activities. A common example is when an Android application launches. This is normally done by clicking on the icon for the application (on the Android desktop or menu) which will launch the application and send the MAIN intent to the declared main Activity. In order for the application to receive this Intent, it needs an Intent-Filter. The Intent-Filter can be defined in the AndroidManifest.xml which makes it a global receiver for a specific intent, meaning the Android operating system or another application can start the application. Intent-Filters can also be defined in Java code by using the IntentFilter class together with the BroadcastReceiver class. This XBee-Terminal uses Intent-Filters and Intents to receive messages from Android about USB connection state and SMS responses. How Intents were used in the app is further explained in section 4.5 and 4.8.

4.3.4 The Android Framework

The Android Framework has a very high level API, which gives easy access to most of the functionality on an Android device. Input and output such as keyboard input or drawing text on the screen can be done with a few lines of code. Support libraries make it possible to use the same code for Android devices with different hardware and with different versions of the Android Operating System. Hardware modules such as the WLAN chip and the USB port can easily be used in an application without much knowledge about the technology behind
The API also includes most of the standard Java classes and extends it with several Android specific classes. The Android operating system uses processes and threads similar to other operating systems, but Android is designed to be a mobile operating system for devices with fewer resources. As such there are a few extra considerations when developing Android applications.

### 4.4 XBee Terminal user interface

The Android framework gives a lot of options when it comes to creating a user interface. There are components for text fields, buttons, keyboard and much more. The user interface is often structured by using XML (Extensible Markup Language) layout files. These XML files contain information about each component on the screen, such as component type, position, size, color and much more. The XML files are then used by an Activity or a Fragment to generate the user interface and to draw on the screen. The Activity and Fragment may also dynamically add, remove or alter the user interface components at runtime.

I wanted the app to be user friendly, and some effort has been given to make it look similar to other chat applications. The user interface consists of the `MainActivity` class and 5 Fragment classes. The `MainActivity` handles USB communication and displays to the user whether the USB is connected or not (Figure 4.3, top of the screen). It will also switch between the different fragments to give access to different functionality. One Fragment is always displayed on top of the `MainActivity`. When the app first starts it will display the `StartFragment` on top of the `MainActivity` (Figure 4.3).

![XBee Terminal startup interface](image-url)
The \textit{StartFragment} checks if the XBee radio is properly connected through the USB port by reading some of its settings (Hardware version, firmware version, serial number, operating channel and network ID) through the USB connection. These settings are displayed on the bottom of the \textit{StartFragment}. When the XBee radio is connected, the “START” button may be pressed to open up a list of nodes connected the DigiMesh network. Navigating back from a Fragment can be done by using the on-screen back button (top left) or the hardware back key.

4.4.1 Node list interface

The \textit{NodeListFragment} class has an interface for displaying a list of nodes connected to the DigiMesh network. It has a refresh button which is used to issue a command to update the node list, in addition to an entry for each node on the network. Each entry displays the Node Identifier (an arbitrary string stored in the XBee radio) and the network address for the XBee radio (corresponds to the serial number in Figure 4.3). The network addresses are used internally by the XBee radio to send, relay and receive wireless packets. Each entry in the user interface has three buttons which allows the user to either open the chat interface, the remote SMS interface or the voice call interface.

4.4.2 Chat interface

When the chat button is pressed, the \textit{ChatFragment} will enter the screen and display a history of messages sent and received from the selected node. It displays who you are chatting with, a list of messages and an input text field which may be written to by the on-screen keyboard. When the send button is pressed it tries to send the message to the selected node and notifies with a “Sent” message if wireless transmission was successful.
4.4.3 Remote SMS interface

The remote SMS interface (Figure 4.6) can be used to send an SMS through the DigiMesh network and out on a mobile network. This function may be useful if one of the nodes does not have mobile network coverage, but is connected to the DigiMesh network. This screen is controlled by the SmsFragment which has input fields for the target phone number and the message itself. When the “SEND” button is pressed, the phone number and message is sent to a remote node which tries to send an actual SMS. As seen on bottom of the figure, the user also receives status information about the SMS sent. Implementation details are in section 4.8.
4.4.4 Voice interface
The voice interface allows for voice communication between two nodes. This interface is controlled by the VoiceFragment and is displayed in Figure 4.7 – 4.10. Before voice communication can begin, one of the nodes has to initiate a voice call by pressing the “CALL” button (Figure 4.7). Figure 4.8 displays how the interface looks like while waiting for the receiving node to accept the call. The receiving node will play a short ringing sound and prompt the user with the VoiceFragment showing the “ACCEPT” and “DECLINE” buttons (Figure 4.9). The decline button cancels the call, while the accept button opens up the in-call interface (Figure 4.10). From this final interface, the two nodes can communicate by pressing the “PUSH TO TALK” button. The quality and data-rate of voice transmitted can be adjusted by using the slider bar (0 is low quality, low data-rate and 10 is high quality, high data-rate). Implementation details are in section 4.9.

Figure 4.7: Voice interface – prior to call
Figure 4.8: Voice interface - initiating call
4.5 USB-to-serial communication

Communication between the XBee-PRO DigiMesh 2.4 radio and the Android device is done by using the USB port on the Android device and the UART on the XBee radio. The USB port connects to an external hardware piece which converts the USB connection to a serial connection which further connects to the UART of the XBee. The external hardware used for this conversion depends on the capabilities of the internal hardware on Android device. Some Android devices have a USB Host controller (USB Host mode). Other Android devices do not have this feature, but can act as a USB slave (USB Accessory mode). In order to support more Android devices I have implemented a solution for both USB modes.

This section explains the key points in the code written and some details about the hardware used.
Chapter 4: Development of the XBee Terminal app

4.5.1 Android as USB Host

Figure 4.11 displays the hardware solution for Android devices with a built-in USB Host controller. It uses an USB OTG cable and a Sparkfun XBee Explorer dongle to connect the XBee radio module to the Android smartphone. The Sparkfun XBee Explorer dongle converts the 5V provided by the Android device to the 3.3V required by the XBee module and has a FTDI chip (FT231X5) which is the USB-to-Serial converter. When the FTDI chip is written to through the USB, it will convert the USB data to serial data and write it to the UART of the XBee module. Vice versa, it will convert serial data received from the XBees UART and write it to the USB port of the Android device.

Figure 4.11: USB to serial connection between an XBee radio and an Android device with the Android device using its USB Host controller

4.5.1.1 Android

The Android framework makes it possible to create apps utilizing the USB port in USB Host mode. It does all the lower level operations of initializing the USB port, sending and receiving data on the USB port. However, to properly communicate with the FTDI chip requires an FTDI driver. The driver in the XBee Terminal comes from the open-source library UsbSerial\(^\text{[63]}\). Among other libraries I tested, this library is the most stable one and makes it possible to use several different USB-to-Serial converters (most FTDI chips, CP210X, PL2303 and more). The library has an API for writing serial data to the FTDI chip and it has a callback method for when data is received.
In Android, connections and disconnections to USB devices are handled through events in the Android operating system. These events can be listened to and received by using an Intent-Filter. And as shortly mentioned, these Intent-filters can be defined in both the AndroidManifest.xml file and in code with the IntentFilter class. In this app, there is an intent-filter in the AndroidManifest.xml file for new USB device connections. This means that whenever a USB device is connected, the Android OS will notify the app that a USB device has connected by sending an Intent to the MainActivity. Since the Intent-Filter is defined in the AndroidManifest.xml file and not in code, the XBee-Terminal will automatically start whenever the USB device is connected. The Intent received contains a UsbDevice class object which is further used to open a USB connection and to read and write to the USB port. The app also defines in code an Intent-Filter for USB disconnects and for permission to use the USB port. These two Intent-Filters are used together with a BroadcastReceiver class, which is a receiver for all Intents defined by Intent-Filters in code. This means that when a USB disconnect happens, the app will receive an Intent through the BroadcastReceiver and further notify the user about a possibly bad USB connection.

Permission for the USB port also needs to be handled before data can be read or written to the USB port. This permission is a requirement by the Android operating system and requires the user to accept a pop-up window. This pop-up is requested through the UsbManager class and the permission can then be received using the BroadcastReceiver and IntentFilter classes.

When the USB device is connected and USB permission is given, the UsbDevice object is used to open the USB-to-Serial connection using the UsbSerial library[^63]. This serial connection is configured with the baud rate (related to data transmission speed), data bits (number of bits per character), stop bits (number of bits sent after a character) and the parity (an error detection method). The XBee radio module is also configured with the same values for serial communication. An example snippet for opening a USB-to-Serial connection (without acquiring USB permission) using the UsbSerial library is displayed in Code snippet 4.1.
Chapter 4: Development of the XBee Terminal app

4.5.2 Arduino as USB Host

The second solution for USB communication with the XBee radio consists of an Arduino Due and a Wireless SD Shield. The Arduino Due acts as the USB Host and is connected to the Android device via a USB OTG cable and a standard USB A to micro-B cable. The Wireless SD shield is mounted on top of the Arduino Due and the XBee radio is mounted on top of the wireless SD shield (see Figure 4.2 - right). In this solution, the Android device cannot supply power which means that a power source such as a 9V battery is required. The Arduino Due converts the 9V to the 3.3V it requires itself and to the 3.3V required by the XBee module. See Figure 4.12 for an overview of the hardware and software for this solution.
4.5.2.1 Arduino

The Arduino Due uses two serial interfaces: One to communicate over USB and one to communicate with the XBee radio. USB communication with an Android device uses the first serial interface with the pins 0 for input and 1 for output. The second serial interface on the Arduino Due consists of the pins 19 for input and 18 for output. The second serial interface is connected to the UART on the XBee radio through the wireless SD shield.

The Arduino Due is programmed with a simple loop which transmits data received on the USB port to the XBee radio, and vice versa. It is programmed in the Arduino language using the Arduino IDE.

Arduino code has its own syntax which is in some sense similar to the C++ programming language. One important difference is that Arduino has a setup function and a loop function. The setup function is invoked once when the Arduino powers on and the loop function will thereafter repeatedly invoke. The Serial objects are also important as they are used to either read or write from a serial interface (an UART port). These are initialized in the setup function and used in the loop function to read and write serial data to the XBee module. Communication via USB is implemented using the Arduino USB Host library\(^{[64]}\) which implements the AOA (Android Open Accessory) protocol. This library has the functionality to connect and communicate with an Android device using the USB and ADK classes. See Code
snippet 4.2 below for a shortened version of the code for the Arduino Due. Appendix C contains the complete Arduino code for both the Arduino Due and for the Arduino Uno with a USB Host Shield.

```c
/*
Created 20 Mars 2016
by Dag Roger Stokland Rui
*/

#include <adk.h>

#ifdef dobogusinclude
#include <spi4teensy3.h>
#include <SPI.h>
#endif

//The LED on Arduino Due is connected to pin 13
#define LED 13

#define XBEE_BUFFER_SIZE 64
#define ADK_BUFFER_SIZE 64

//USB class used communicate via USB
USBHost usb;

//Initialize ADK (object used to communicate with an Android device)
ADK adk(&usb, "Test Manufacturer", // Manufacturer Name
        "Test Model", // Model Name
        "XBee Terminal", // Description (Visible on the Android device when USB is connected)
        "1.0", // Version
        "," // URL to app if Android device doesn’t have a compatible app
        "123456789"); // Serial Number

bool connected;

//Buffers for messages (Android to XBee and XBee to Android)
uint8_t adkBuffer[ADK_BUFFER_SIZE];
uint8_t xBeeBuffer[XBEE_BUFFER_SIZE];

void setup() {
    //Initialize serial port 0 (pins 0 and 1)
    Serial.begin(57600);

    //Initialize serial port 1 (pins 18 and 19)
    Serial1.begin(57600);

    //Delay 200ms to wait for the serial ports to initialize
    delay(200);
}

void loop() {
    //USB task needs to run once every iteration.
    //This function handles lower level USB connection and communication
    usb.Task();

    //Turn on LED if Android device is connected (indicates if USB is
Chapter 4: Development of the XBee Terminal app

4.5.2.2 Android

Using the Android USB Accessory mode API is very similar to the USB Host mode API. USB connections, disconnections and permissions are all handled similarly to the USB Host mode implementation. A major difference is that USB Accessory mode uses a special file required by the Android AOA protocol in order to do a handshake (to connect) with the connected USB device. This file is called accessoryFilter.xml and needs to contain information corresponding to the information defined in the ADK object on the Arduino side (see ADK object initialization in Code snippet 4.2). The information that needs to correspond is the manufacturer name, the model name and the version strings.

Another difference compared to USB Host mode is that USB Accessory mode functionality is accessed through the UsbAccessory class. This class provides methods for reading and writing to the USB accessory and requires no additional driver or library in order to communicate with the Arduino Due. The UsbAccessory class object is obtained in a similar way to USB Host mode where an Intent-Filter is defined in the AndroidManifest.xml and received in the MainActivity. USB permission is requested through the UsbManager class, and disconnects and permission is also received by using the IntentFilter and BroadcastReceiver classes.
Chapter 4: Development of the XBee Terminal app

4.6 Serial communication with the XBee-PRO 2.4 DigiMesh radio

This section covers how the XBee Terminal app communicates with the XBee-PRO DigiMesh 2.4 radio through the USB-to-Serial converter. I will start by explaining some basic usage and functionality of the XBee radio module and then go into detail on the Android-side implementation for communication with the XBee module.

4.6.1 XBee-PRO 2.4 DigiMesh serial communication

The XBee radio is a packet based radio with its own networking protocol called DigiMesh. In other words, there is no need to implement any sort of networking protocol. The XBee radio processes received packets, checks for errors and automatically tries to relay packets for other radios. It also incorporates an acknowledgment (ACK) scheme, by sending ACK packets back to the original sender on a received packet.

The XBee Terminal app simply sends and receives messages from the XBee radio using the USB-to-Serial converter. The format of these messages is related to the operating mode of the XBee radio. There are a total of 3 operating modes which can be used: Transparent mode (also called AT mode), API mode and API mode with escaped characters. Transparent mode and API mode were the ones I considered to use, with API mode being the most beneficial to the application.

4.6.1.1 Transparent mode

Transparent mode is the default operation mode for XBee modules. In this mode, any data received on the XBee UART is transmitted wirelessly to a remote module identified by the destination address register (The registers are covered in section 4.6.3). For example, if a series of bytes containing the string “hello” is received on the UART, the XBee will wirelessly transmit a packet to another XBee which will transmit “hello” on its UART. This makes Transparent mode easy to use with other hardware and is ideal in applications where the destination address is static.
The destination address can be configured at any time in order to switch the receiving radio. However, changing the destination address in Transparent mode takes some time and interferes with other messages, which otherwise should be received or transmitted. Transparent mode gives no option to disable acknowledgment packets and does not give an indicator to which XBee radio a message was sent from. Transparent mode leaves out some of the functionality of the XBee module which is beneficial to the XBee Terminal app. The XBee Terminal does therefore not support Transparent mode.

4.6.1.2 API mode

In API mode, a message sent to or received from the UART of the XBee has a special format called a frame. A frame sent to the UART contains additional information used in wireless transmission of messages. When sending a wireless packet, the destination address registers are ignored and instead the frames must contain a destination address together with several other parameters. The XBee module will process the frame, and depending on its contents transmit a wireless packet to another XBee through the DigiMesh network. The receiving XBee module will then generate a frame and transmit a frame on its UART, specifying the address of the source XBee module.

API mode is more useful for unicast transmitting between XBee modules, since it does not need to change the destination address register before transmitting. On the other hand, since frames contain more data than AT mode messages, more data needs to be transmitted between the XBee UART and the Android USB. This can be an issue if the USB-to-Serial converter has a slower data rate than wireless transmission. In addition, the frames needs to be generated before they are sent to the XBee and processed when they are received. Frames have a basic structure which is common between all frames. This structure and a short explanation for each field are displayed in Table 4.1.
4.6.2 API frames implementation

I wanted the app to be able to transfer unicast and broadcast messages, and for the receiver of a message to be able to see which radio sent the message. API mode was therefore the natural choice. The XBee user-guide\(^{[65]}\) describes an array of frame types with different purposes. The XBee Terminal app does not have an implementation for all of these frames, but implements 5 Java classed used to interpret and generate certain frames. This is a short summary of the frames used in the application:

- **Transmit Request Frame:** This frame is a *sending frame*, meaning it can only be transmitted to the UART of the XBee module. It is used for transmitting a wireless packet containing a message to another XBee radio. The radio receiving the wireless packet will acknowledge this packet by automatically sending acknowledgment packets (ACKs) back to the source radio.
• Transmit Status Frame: This frame is a receiving frame, meaning it can only be sent from the UART of the XBee. This frame is transmitted on the UART when a wireless packet is acknowledged or when there is an error in transmitting. It is associated with a Transmit Request Frame through an identifier.

• Receive Packet Frame: This frame is a receiving frame and is transmitted on the UART when the XBee radio receives a wireless packet. It contains the message sent by a Transmit Request Frame and contains among other parameters, the source address of the packet.

• AT Command Frame: This frame is a sending frame, used to issue a specific command to the XBee module. There are several such commands (XBee manual [65]), but in this app, the AT Command Frame is only used to read configuration parameters (such as the serial number) from the XBee radio, and to issue the Network Discovery (ND) command. The Network Discovery command issues the XBee radio to scan the DigiMesh network for connected XBee radios and to return an AT Command Response Frame for each radio.

• AT Command Response Frame: This is a receiving frame and contains the response of an AT Command Frame. When an AT Command Frame issues to read a configuration parameter from the XBee radio, this frame will return the result. When the Network Discovery command is issued, one AT Command Response Frame will be returned for each node on the network. Each frame returned from a Network Discovery command contains a Network Discovery Response which contains the address and Node Identifier of the node (as seen in the node list user interface).

The exact format and an explanation for each of these frames are compiled in Appendix B. The appendix also includes an example of a Transmit Request Frame and the format of the Network Discovery Response.

4.6.2.1 Frame buffer

The XBee Terminal also contains a class called XBeeFrameBuffer which buffers data received on the USB into frames. This is needed because data is continually read from the USB as it is written to by the USB-to-Serial converter. Normally, a part of a frame or in some cases more than one frame is read from the USB. Incoming USB data is therefore buffered by the
XBeeFrameBuffer class. Outgoing USB data is buffered by the XBee radio itself. Considering the basic frame structure (Table 4.1) and that a random number of bytes can be received, the buffer needs to do several things:

- Insert data into the buffer every time data is received from the USB.
- Check if the start delimiter is correct. If not, the USB-to-serial connection may have failed to transfer a frame. This happens rarely, but should be considered. The solution is to search the buffer for the next occurrence of the start delimiter and thus drop parts of the buffered data.
- The next step is to check that the whole length of a frame has been received in the buffer. This is done using the length field of the frame to determine the total number of bytes needed to parse the frame. If the number of bytes in the buffer is equal to or larger than the size of the frame, the frame can be parsed and used.
- This implementation does not check if the checksum is correct, as the XBee radio only uses wireless packets that are acknowledged. The XBee radio automatically processes the checksum when packets are acknowledged. Packets with wrong checksum are discarded by the XBee radio. Alternatively, packets do not need to be acknowledged, giving more options for implementing error checking methods on the Android side.
- Lastly the frame type field is used to determine what the frame data field should be used for. At this point the frame can be parsed and further processed by the rest of the application.

Code snippet 4.1 contains Java-like pseudo-code for the XBeeFrameBuffer class. It leaves out some minor error checking code and does not show the createFrame method which extracts each field from the frame into a class object. The implementation is not optimal and several improvements to efficiency can be made, such as using a ring buffer or another data structure for the buffer.
// Constants
int FRAME_BASE_SIZE = 4;  // Start delimiter + length + checksum
int START_DELIMITER = 0x7E;
int BUFFER_SIZE = 1000;

// Counter for the total number of bytes in the buffer
int byteCount = 0;

// Position of the start of the frame currently being parsed
int position = 0;

byte[] buffer = new byte[bufferSize];

/*
 * The putAndCheck method is invoked whenever data is received from the USB.
 * It returns a list of frames parsed from the buffer.
 * The createFrame(buffer, offset, length) method parses each field into a
 * class object (See appendix B for the format of these frames)
 */
List putAndCheck(byte[] bytes) {
    /*
    * Copy bytes to the end of the buffer. The arraycopy method is native in
    * Java: arraycopy(source, sourceOffset, destination, destOffset, length)
    */
    arraycopy(bytes, 0, buffer, byteCount, bytes.length);
    byteCount += bytes.length;

    if (byteCount < FRAME_BASE_SIZE)
        return null;

    // This loop tries to parse frames until the buffer is empty
    while (byteCount > position) {
        // Check if the start delimiter is incorrect
        if (buffer[position] != START_DELIMITER) {
            /*
            * Search for 0x7E in the buffer and set position to the
            * position of 0x7E. If not found, reset position and
            * byteCount to 0 and return (discard buffer).
            */
        }

        // Extract the length of the data field
        int length = buffer[position + 1] << 8 | buffer[position + 2];

        // Find where the frame ends in the buffer
        int frameEnd = position + length + FRAME_BASE_SIZE;

        // If there is exactly one frame in the buffer, parse it and reset
        // the buffer
        if (frameEnd == byteCount) {
            createFrame(buffer, position, length + FRAME_BASE_SIZE);
            // add the created frame to a list of frames

            position = 0;
            byteCount = 0;
        }
    }
}
//If there are more than one frame in the buffer, parse it and 
//continue 
else if (frameEnd < byteCount) {
    createFrame(buffer, position, length + FRAME_BASE_SIZE);
    position = length + FRAME_BASE_SIZE;

    //add the created frame to a list of frames
}

//If there is no frame in the buffer, copy any bytes to the start 
//of the buffer and return (wait for more data)
else if (frameEnd > byteCount) {
    arraycopy(buffer, position, buffer, 0, byteCount - position);
    byteCount = byteCount - position;
    position = 0;
    break;
}

//Return a list of frames created by the createFrame method 
return list;

4.6.3 XBee Configuration

The XBee-PRO DigiMesh 2.4 has several registers which may be configured to alter the way 
the radio works. Configuring an XBee radio can easily be done using the XCTU software [66] 
with a USB-to-Serial converter, such as the SparkFun XBee Explorer Dongle with on a PC .
XBee configuration includes networking, diagnostics, addressing, encryption, serial 
communication and sleep commands. Most of these settings were kept default or were not 
used for this application. However, some configuration is needed to make XBee-PRO 
DigiMesh 2.4 compatible with the XBee Terminal. Below is a list of the important changes, 
but also some minor ones which should be taken note of.

- Operating channel – Used to configure the frequency band used for wireless 
  transmission. There are a total of 16 channels working within the range of 2.405GHz 
  to 2.480GHz. All modules must be configured to use the same operating channel.

- Network ID – A network identifier on top of the operating channel. Only modules 
  with the same Network ID may communicate. All XBee radios must also be 
  configured with the same Network ID.

- API enable – This register determines the operating mode for the XBee (AT mode, API 
  mode or API mode with escaped characters). The XBee Terminal only supports API 
  mode.
• Baud Rate – This is the baud rate used for serial communication via the UART. It can be configured for up to 115200 baud. However, communication via the serial-to-USB converter is more stable at lower rate and data rate is already limited by the wireless transmission rate. Baud rate is configured to 57500 in the XBee Terminal app and the XBee radio must also be configured to use 57500 as the baud rate.

• Unicast MAC Retries – The maximum number of packet delivery attempts for unicasts. This is kept at the default value of 10 which means that the XBee radio can attempt to send a packet up to 10 times if it does not receive an ACK packet after a timeout.

• Transmit Power level – This is set to the maximum value which gives the best possible wireless range.

• Node Identifier – This register contains a string of up to 20 characters which is transmitted to an XBee node which performs the Network Discovery command. I have set each XBee radio with a unique Node Identifier: “Jens”, “Bob” and “Karl”.

• Destination Address – Determines the address where a packet should be sent to when the XBee radio is in AT mode.

4.7 Data management

Data management in the XBee Terminal app consists of keeping track of the XBee radios on the DigiMesh network and presenting the data received from each XBee radio. Each data type are all sent using the Transmit Request Frame and received using the Receive Packet Frame. Both chat messages also use the Transmit Status Frame to display to the user whether a wireless transmission was successful or not.
Common for all Receive Packet Frames carrying any data type is that they are processed by the XBeeService class. This class does two things:

- It creates a list of XBee radios (nodes) based on information in the frames received.
- It distributes the frames to each Fragment class based on their data type. The data type contained in a Receive Packet Frame is determined by a one byte field which is always sent together with the rest of the message. For example if a Receive Packet Frame contains a chat message, the message will contain the prefix “m” in ASCII (or 0x6d in hexadecimal). The different data types and their prefixes are:
  - Chat message – m
  - Voice message – v
  - Voice status message – u
  - Remote SMS message – s
  - Remote SMS status message – t

### 4.7.1 Node management

Creating a list of XBee nodes in the DigiMesh network is done using the information retrieved from Receive Packet Frames (carrying for example a chat message) and from Network Discovery Responses (see Appendix B for reference).

The data contained in Network Discovery Responses is parsed by the XBeeService and stored in a map data structure of Node objects. This map has a unique entry for each node, using the address of each XBee module as the unique key. Network Discovery Responses contain some information which is irrelevant to the application and only the source address and Node Identifier is actually used and displayed to the user. For Receive Packet Frames, the only obtainable information is the source address. This means that if a chat message is received, only the source address of the sender will be displayed in the node list interface and a Network Discovery is required to see the Node Identifier string. This Network Discovery command is issued by pressing the “REFRESH” button in the node list interface. The map is directly used by the node list interface to create a node entry and to display the Node Identifier and address of each node.
A similar type of map structure is also used in the ChatFragment and SmsFragment in order to filter messages depending on the address of the sender. The ChatFragment stores both incoming and outgoing messages. When a chat message is sent, it generates a Transmit Request Frame with a new frame ID which is stored in a list of un-acknowledged messages. After a message is sent and when a Transmit Status Frame is returned from the XBee radio, the un-acknowledged message is removed from the list and the user is notified with the status contained in the Transmit Status Frame.

4.7.2 Message flow

Figure 4.13 displays the MainActivity, XBeeFrameBuffer, XBeeService and Fragment classes and explain how they interact with each other when receiving and sending data via the USB port.

![Figure 4.13: XBee Terminal message flow and classes. Incoming USB data is buffered by the XBeeFrameBuffer before it is parsed in an XBeeFrame object and then distributed by the XBeeService.](image-url)
4.8 Remote SMS

Remote SMS is a function in the XBee Terminal app which utilizes the mobile radio of the Android smartphone together with the connected DigiMesh network. It allows the user to send a remote SMS message to another Android-XBee pair (node) which then uses its mobile radio to send an actual SMS. The idea behind this function is to allow for some communication to mobile networks even if there is no mobile network coverage. If one of the nodes within a local DigiMesh network has mobile coverage, every other node on the DigiMesh network would be able to send a remote SMS. Figure 4.14 displays a message flow diagram when sending a remote SMS.

![Remote SMS message flow diagram](image)

*Figure 4.14: Remote SMS message flow diagram. The source node sends the remote SMS. The remote node receives the remote SMS and sends the actual SMS.*

4.8.1 Sending a remote SMS message

The source node will use a *Transmit Request Frame* to transmit the phone number, the SMS message and an SMS identifier to the remote node. The phone number and SMS message is then used by the remote Android smartphone to send an SMS. A complete overview of the data sent in a remote SMS is shown in Table 4.2.
### Data Field | Byte Position | Explanation
--- | --- | ---
**Data type** | 1 | The data type specifies the contents of the rest of the message. Remote SMS has the identifier “s” (0x73 in hexadecimal).
**SMS ID** | 2 | This field is used together with the address of the sender to create a unique entry for each remote SMS. It is generated and stored by the sender so that the sender can receive a response to the remote SMS (if SMS failed or succeeded).
**Phone number length** | 3 | The length of the phone number field (Phone numbers can be 3 to 13 characters long).
**Phone number** | 4 to n | The phone number is stored in a numerical format.
**SMS message** | n to m | The SMS message in ASCII.

*Table 4.2: Contents of the RF data field in a Transmit Request Frame when a remote SMS is sent*

The receiving smartphone will parse the *Receive Packet Frame* containing the message received. It will then use the Phone number and SMS message fields to send an SMS. Both the sending and receiving node will also store each of these fields in a *SmsMessage* object which is used to send a response back to the sender.

### 4.8.2 Sending an SMS

The Android API makes it very simple to send SMS from within an application:

```java
1. SmsManager smsManager = SmsManager.getDefault();
2. smsManager.sendTextMessage(destinationNumber, serviceNumber, message, sentIntent, deliveredIntent);
```

*Code snippet 4.4: Sending an SMS with the Android API*

The first three parameters of the *sendTextMessage* method are pretty straight forward. They take a string with the destination telephone number, the service center address (null to use the default service center) and the SMS message. The *sentIntent* and *deliveredIntent* parameters can be used to obtain transmission status of the SMS. The *sentIntent* is used to
obtain information about whether or not the phone was able to transmit the SMS to the mobile network. The `deliveredIntent` is used to obtain information about whether or not the SMS was received by the target phone. Both of these intents are implemented to send a status message back to the node that originally sent the remote SMS (as seen in the interface of Figure 4.6). The content of the status message is displayed in Table 4.3.

<table>
<thead>
<tr>
<th>Data Field</th>
<th>Byte Position</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data type</td>
<td>1</td>
<td>The data type specifies the contents of the rest of the message. Remote SMS status has the identifier “t” (0x74 in hexadecimal).</td>
</tr>
<tr>
<td>SMS ID</td>
<td>2</td>
<td>Specifies which remote SMS has a status update.</td>
</tr>
<tr>
<td>SMS Status</td>
<td>3</td>
<td>Specifies the status of the SMS according to the information obtained from the <code>sentIntent</code> and <code>deliveredIntent</code>. The possible statuses are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>SMS_SENT_TO_REMOTE = 0</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>SMS_RECEIVED_BY_REMOTE = 1</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>REMOTE_MOBILE_SENT_SMS = 2</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>TARGET_MOBILE_RECEIVED_SMS = 3</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>REMOTE_MOBILE_ERROR_GENERIC_FAILURE = 4</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>REMOTE_MOBILE_RESULT_ERROR_NO_SERVICE = 5</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>REMOTE_MOBILE_RESULT_ERROR_NULL_PDU = 6</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>REMOTE_MOBILE_RESULT_ERROR_RADIO_OFF = 7</code></td>
</tr>
</tbody>
</table>

*Table 4.3: Remote SMS status message content*

### 4.8.3 Remarks

Before a remote SMS message is sent, the app does not check to see if the remote node has mobile coverage. Instead, when a remote SMS is sent, the app will display what happened according to the SMS status message. Alternatively, to increase chances of a successful SMS, a remote SMS can be broadcast on the DigiMesh network, making every other node send an SMS to the target phone. In addition, this implementation does not split up a remote SMS message when it sends it through a `Transmit Request Frame`. The length of the SMS message is therefore limited by the maximum size of the `Transmit Request Frame RF data` field (which is 73 bytes long on the XBee-PRO Digimesh 2.4).
4.9 Voice communication

Transmitting voice over the DigiMesh network has a few extra considerations compared to transmitting text messages. First of all, audio (or voice) recorded by an Android device has a rather high data rate. In order to wirelessly transmit the recorded audio data, it needs to be compressed by an audio codec. This audio codec needs to be able to encode (compress) and decode (de-compress) audio fast enough for the Android device to handle real-time recording, encoding, decoding and playback simultaneously. In addition, the codec needs to be able to compress the audio to a data rate low enough should the DigiMesh network be able to transfer it. In a DigiMesh network it is always better to have a lower data rate transmitted as it reduces packet errors, which in turn reduce delay and increases the distance that two nodes can communicate. I chose to use the Speex open-source voice codec\[67\] as it has a high compression rate compared to other alternatives\(^2\).

Further, packets transferred over a DigiMesh network may arrive in a different order than they were sent. This is due to errors in transmission which causes a retransmission of the same packet at a later time. When roaming this becomes an even bigger issue and may also cause jitter in packets received. These two issues have not been considered in the implementation.

4.9.1 The Speex voice codec

The Android API has several native voice codecs such as the AMR (Adaptive Multi-Rate) codec and the GSM Full-rate codec, which are often used in mobile wireless communication. Both of these codecs have high compression rates and are in other ways ideal for real-time voice communication. However, these codecs has to be used through the \texttt{MediaRecorder} and \texttt{MediaPlayer} classes when used from within an application. These classes have a restriction in that they can only read audio from a file, or write audio to a file (when recording and playing audio). The problem here is that reading from a file before wireless transmission and then writing to a file on reception would cause additional latency. Because of this restriction, I choose to not use any of the native Android audio codecs.

\(^2\)A good comparison of audio codecs can be found at: https://en.wikipedia.org/wiki/Comparison_of_audio_coding_formats
The alternative is to use the *AudioRecord* and *AudioTrack* classes in the Android API. These classes support reading and writing audio data into a buffer in memory. They do not support audio compression and use PCM (Pulse-Code Modulation) to play and record audio. PCM audio data has a rather high data rate which cannot reliably be transferred over a DigiMesh network. The Speex open-source voice codec[^67] is therefore used to compress audio data.

Speex is a real-time lossy (quality decreases when encoding and decoding) voice codec for the PCM format. Speex has several settings, such as a compression versus quality setting and a setting for the audio sample rate. The audio sample rates are divided into three modes: Narrowband (8000Hz), wideband (16000Hz) and ultra-wideband (32000Hz) mode. The XBee Terminal uses the narrowband mode, and allows the user to adjust the compression setting with the slider bar on the user interface. The quality level in Speex is a value from 0 to 10 (inclusive). Table 4.4 shows a comparison between the quality and data rate for each setting. The table displays the bit-rate, the FLOPS (FLoating-point Operations Per Second) and a quality description from the creator of Speex. In my opinion, the quality of setting 2 has very little noise and voice is easy to understand. Quality setting 2 is set as default in the XBee Terminal.

Speex also includes features for echo cancellation, packet reordering and a jitter buffer, though these features were not utilized in the XBee Terminal. Speex is written in the C++ programming language which does not natively compile in a Java Android application. I have therefore used a JNI (Java Native Interface) wrapper library[^68] to link the Speex library to the XBee Terminal app.
Table 4.4: Speex quality versus bit-rate

<table>
<thead>
<tr>
<th>Quality setting</th>
<th>Bit-rate (bps)</th>
<th>Estimated Mega FLOPS</th>
<th>Quality/Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2150</td>
<td>6</td>
<td>Vocoder (mostly for comfort noise)</td>
</tr>
<tr>
<td>1</td>
<td>3950</td>
<td>10.5</td>
<td>Very noticeable artifacts/noise, good intelligibility</td>
</tr>
<tr>
<td>2</td>
<td>5950</td>
<td>9</td>
<td>Very noticeable artifacts/noise, good intelligibility</td>
</tr>
<tr>
<td>3 – 4</td>
<td>8000</td>
<td>10</td>
<td>Artifacts/noise sometimes noticeable</td>
</tr>
<tr>
<td>5 – 6</td>
<td>11000</td>
<td>14</td>
<td>Artifacts usually noticeable only with headphones</td>
</tr>
<tr>
<td>7 – 8</td>
<td>15000</td>
<td>11</td>
<td>Need good headphones to tell the difference</td>
</tr>
<tr>
<td>9</td>
<td>18200</td>
<td>17.5</td>
<td>Hard to tell the difference even with good headphones</td>
</tr>
<tr>
<td>10</td>
<td>24600</td>
<td>14.5</td>
<td>Completely transparent for voice, good quality music</td>
</tr>
</tbody>
</table>

4.9.2 Initiating a voice call

Before any kind of voice data is transmitted between two nodes, one of the nodes needs to initiate a call and the second node needs to accept the call. The idea behind the initiating stage is to allow the receiver to either accept or decline the call to eliminate collision (if for example two nodes sends voice data to a single node), and also to notify the receiver of an incoming voice call. When the receiving node accepts the call it can immediately start transmitting voice data to the initiating node. It will also send an “accepted” message to the initiating node, telling the initiating node that voice data can be sent. These messages are called voice status messages to separate them from voice data messages. Voice status messages use the prefix “t” and include the following messages:

- Call – initiate a voice call.
- Accept – accept a voice call.
- Decline – decline a voice call.
- Hang up – cancel an ongoing voice call.
4.9.3 Audio recording and encoding

Audio recording is done using the `AudioRecord` class from the Android API. This class is used to read analog audio signals into digital audio buffers in the 16-bit PCM (Pulse-Code Modulation) format at 8000 samples each second. According to the documentation, this format and sample rate may not be supported on all devices as they depend on the hardware of the device, but should be supported on most phones. Including my test phones Samsung S5 Neo and Huawei Honor 4X.

The `AudioRecord` is linked to the `AcousticEchoCanceler` in the Android API. The `AcousticEchoCanceler` reduces the echo caused by playing and recording audio at the same time. However, whether the `AcousticEchoCanceler` is supported or not is also dependent on the capabilities of the Android device, as it is not supported on my Huawei Honor 4X.

Further, the PCM audio buffers are processed by the Speex encoder in narrowband mode. This mode encodes audio frames of 160 16-bit PCM samples (320 bytes) into an encoded audio frame. This means that 160 samples of PCM data is recorded and buffered before encoded, and causes a constant delay of 20ms (50 frames a second). The data size of the encoder output depends on the quality setting used. At setting 2, the Speex encoded audio frames varies between a size of 14 and 15 bytes at an average of 14,875 bytes.

The encoded audio frames are then wirelessly transferred via the XBee radio to another node.

4.9.4 Wireless transmission

Each Speex encoded audio frame is transferred using a `Transmit Request Frame` together with the prefix for voice data “v”. The average amount of data transmitted in each wireless packet is therefore 14,875 + 1 bytes for Speex quality setting 2. This transmission happens 50 times each second which translates to 6350 bps.

Sending one audio frame using one `Transmit Request Frame` causes the least delay as opposed to buffering several audio frames and sending them in one `Transmit Request Frame`. It is also suggested by the XBee manual to send several smaller packets rather than fewer larger packets as it reduces the chance of errors in each packet transmitted. On the
other hand, larger packets means less overall data transmitted as each packet also contains a header.

Similarly to text messages, voice communication also use acknowledged packets, but Transmit Status Frames are disabled to reduce USB-to-serial communication. This is done by setting the Frame ID field of the Transmit Request Frame to 0.

### 4.9.5 Voice decoding and playback

At the receiving end, the Speex encoded audio is decoded into audio frames of 160 samples of 16-bit PCM audio data. The audio is then played using the `AudioTrack` class in streaming mode. Streaming mode essentially allows audio to be written to the internal Android audio buffer before it is played sequentially. Similar to recording, audio is played at 8000 samples each second. Encoded audio frames are decoded and played as they are received, and there is no buffering implemented except for the buffering provided by the `AudioTrack` class.

### 4.9.6 Remarks

The echo cancellation provided by the `AcousticEchoCanceler` may not be good enough for good quality two-way communication. The Speex library does provide functionality to further remove echo, but at the cost of more processing time and an extra delay in buffering the audio. The JNI wrapper for Speex does not implement this functionality and I have not been able to implement them myself.

As seen from the quality level setting, better sound quality means a higher data rate for wireless transmission. Lower level setting should then provide more reliable wireless transmission than a higher level setting, especially over longer distances and when doing multiple hops over the DigiMesh network.

### 4.10 Technical Challenges and problems

This project was to a large degree dependent on open-source libraries, on both the Android and the Arduino side. Most of these were easy to understand and use. However I have had major problems with Arduino USB Host library when the Arduino Due acts as the USB Host. Specifically, the Arduino Due has problems in maintaining USB connection to the Android
device. Furthermore, some Android devices are more stable in communication with the Arduino and some Android devices would not establish a connection at all. A Samsung GT-I19195 has a more stable USB connection than an LKD F5 and a Huawei 4X che2-L11 would not establish communication at all with the Arduino Due.

Debugging the XBee Terminal was also a problem since the USB cable cannot connect to the XBee module and a PC at the same time. The solution to this problem is to use ADB (Android Debug Bridge) over Wi-Fi\(^{[73]}\). With ADB over Wi-Fi and the Android device in USB host mode and connected to the XBee radio, debugging worked perfectly. But when the Arduino acted as USB Host, the Android devices would randomly disconnect from the debugger. This caused the development process to take much longer time than it should have. Further, the Arduino Due USB connection is much slower than an Android device in USB Host mode. In fact, the USB connection was so slow it was not able to transfer real-time one-way Speex encoded voice. Direct voice communication is therefore not possible using the Arduino Due as USB Host. In hindsight I would have liked to have spent less time trying to solve the problems with Arduino and only implemented the app for Android devices with USB Host.

### 4.11 Distribution

Currently, the XBee Terminal app can be viewed as in a testing phase. All the functionality explained in this chapter works, but I have not had the time or resources to solve and find minor bugs. There may be user interface bugs and there are likely bugs with USB Accessory connectivity. Appendix D contains a complete list and a short explanation of each of the Java classes in the XBee Terminal.

#### 4.11.1 Source code

The source code for the XBee Terminal app and the Arduino is open and free for everyone at:

- [https://github.com/Daffern/XBee-Terminal](https://github.com/Daffern/XBee-Terminal)
The Android source code is not commented. It contains a precompiled version of the Speex library together with the source code of the UsbSerial library. It also includes build files and other files used by Android Studio. The folder structure of the source code is:

- XBee-Terminal/app/src/main/
  - java/
    - com/
      - fehlr/ (contains the source code for the UsbSerial Library)
      - purplefrog/speexjni/ (contains the Java wrapper code for the Speex library)
        - no/daffern/xbeecomunication/ (contains the source code written in this thesis)
    - jniLibs/ (contains the precompiled Speex library)
  - res/ (contains default and custom resource files used by the app)
    - layout/ (contains the layout files written in this thesis)
    - xml/ (contains the USB filter files written in this thesis)
    - menu/ (contains a menu file written in this thesis)
  - Arduino code/ (contains the Arduino code written in this thesis)

Appendix C also contains the source code, in addition to the compiled APK file which can be installed on an Android device.

### 4.11.2 Google play store

The XBee Terminal app can be downloaded and installed from the beta section at the Google Play store using this link:

5 Tests and measurements

Using the XBee-PRO DigiMesh 2.4 module together with the XBee Terminal there are two
groups of measurements done in this chapter:

- Range and quality measurements of voice communication, and
- Range measurements of chat messages.

Measurements have been done on field at Bygdøy in Oslo. The field is open and flat and is
about 950 meters long at its longest. This field is close to optimal conditions where the XBee
radios have a line of sight to each other. Some measurements have also been done in less
optimal conditions, inside a dense forest.

5.1 Background for wireless transmission on the XBee-PRO DigiMesh 2.4

Measurements with the XBee-PRO DigiMesh 2.4 have only been done using unicast
transmission. In unicast transmission, the receiving radio will transmit ACK packets back to
the sender when it successfully receives a packet. This packet indicates to the sender that a
packet has been successfully received. If an ACK is not received, the sender will retransmit
the original packet after a timeout. Retransmitting a packet can happen up to 10 times
before the XBee radio finally gives up. This is the default transmit retry value, which can be
configured with the RR (Unicast MAC Retries) command.

In a mesh topology network (more than two nodes), a relay node can retransmit a received
packet an additional 10 times (specified by the RR command). This means that when a relay
node is routing packets between two nodes, the maximum number of transmit retries is 20.
Additionally, there is a MR (Mesh Unicast Retries) setting on the XBee module. This value
determines how many unicast transmit attempts should be made on top of the Unicast MAC
Retries. For example: With one relay node (double-hop), Unicast MAC Retries set to 10 and
Mesh Unicast Retries set to 2, the total maximum transmit retries would be 40. I have
specified these two settings in each of the measurements done.
Chapter 5: Tests and measurements

The closer the communicating radios are to each other, the better are chances for successful transmission. At longer distances, the environment and signal loss causes more packets to fail. Packets transmitted over longer distances are therefore more often retransmitted before a successful transmission. This retransmission has two side effects: Packets may be delayed and packets may be received in a different order than they were scheduled. This has a huge effect on the quality of voice communication.

5.2 Voice communication latency measurement method

In order to properly measure the latency in voice communication, an additional interface can be opened from the menu to show the PerformanceFragment below the VoiceFragment (see Figure 5.1). This interface measures the audio amplitude recorded on both the sending node and the receiving node. It also displays when there is a peak in the audio amplitude together with a timestamp of the peak. This timestamp is compared between two nodes to measure the latency in voice communication. The clock used for the timestamp is synchronized between nodes by using the GPS on each Android device.

Figure 5.1: XBee Terminal app performance interface below the voice interface. The bottom right fields displays parameters read from the XBee radio, such as the RSSI (Received Signal Strength Indicator). These are not used in this paper, but can give some indicators to how good a network link is when using the app. The bottom right fields display the GPS synchronized clock, the last recorded audio amplitude peak and the time of the amplitude peak.
An amplitude peak is generated by one of the nodes making a loud noise (clapping hands or snapping nails on the phone). The amplitude peak together with the timestamp is then stored at the sender before the audio is transmitted to the receiver. The receiver will receive and play the sound, and then record the same sound. Then the amplitude of the recorded sound is measured and the amplitude peak and timestamp is stored. This method of measuring the latency excludes two parameters compared to the latency experienced by a human:

- The measurements excludes the time it takes for the sender to record the sound (From when the sound is generated to when the sound is recorded and the amplitude is measured).
- The measurements includes an additional latency caused by the time it takes to record the sound at the receiver (From when the sound is played/generated to when it is recorded and the amplitude is measured).

If recording and playback has the same latency between two devices, these two parameters should even each other out. However, this was not the case as explained in the following measurements and in section 5.4.

### 5.3 Voice measurements

The measurement I have taken most notice of is the latency in voice communication, but also the distance one can communicate and the quality (from a subjective view). The distance is measured by using coordinates acquired by GPS.

Measurements were done between two nodes, with two network topologies: Single-hop (node to node) and double-hop (one relay node). All measurements are one-way, as two-way voice communication does not work properly. This is because the XBee Terminal has no proper solution for echo cancellation. The Speex quality setting is set to 2, which means data is transferred at 6350 bps. Further, the Unicast MAC Retries is set to 3 and Mesh Unicast Retries to 1. This is done to reduce jitter and wrong packet order. Lastly, all USB-to-Serial communication is done with the Android devices in USB Host mode using a Huawei 4X che2-L11 (Android 4.4.2) and a Samsung Galaxy S5 Neo SM-G903F (Android 5.1).
5.3.1 Single hop in optimal conditions

The first measurement was done at a distance of 470 meters. At this distance packets was reliably transferred and it was easy to understand and talk to each other. Distances longer than this (480 – 500m) had packet loss noticed by voice loss and sometimes incomprehensible voice. We also noticed it was important to face the radios toward each other and stand still as we were talking. Both radios were kept at shoulder height, about 1.5 meters above ground. Figure 5.2 displays a picture of the location where we did the single hop measurements.

![Figure 5.2: Single hop measurements location. Image taken from Google Maps](image)

At 470 meters distance, we did 5 amplitude peak measurements. These had a latency of:

- 321 ms
- 327 ms
- 762 ms
- 517 ms
- 494 ms
The numbers are quite varied, especially when comparing the top two against the bottom three measurements. The main reason for this gap is that the top two measurements were sent from the Huawei X4 smartphone to the Samsung Neo S5 smartphone, and the three bottom measurements were sent from the Samsung Neo S5 to the Huawei 4X. This variety is explained further in section 5.4.

There may also be other errors in these numbers, such as the 762ms measurement. Errors may be caused by badly synchronized clocks, audio artifacts caused by Speex, packet jitter or simply by human error. The latency experienced when talking to each other using the app was quite noticeable, and I imagine the latency to be in the range of 380 ms to 440 ms. Otherwise, there were no jitter, no packet loss and good quality sound.

5.3.2 Double hop in optimal conditions
The second measurements had a relay node between the two communicating nodes as seen in Figure 5.3. The total distance between the communicating nodes had a range of 664 meters and distances between each node were 400 meters and 264 meters. The communicating nodes were about 1.5 meters above ground and were both facing the relay node. The relay node was about 0.2 meters above ground.

Figure 5.3: Double hop measurements location. Image taken from Google Maps
The amplitude peak measurements had a latency of:

- 402 ms
- 255 ms
- 499 ms
- 497 ms
- 503 ms

The top two measurements were sent from the Huawei 4X to the Samsung Neo S5, and bottom three vice versa. Compared to the previous measurements, these numbers shows an increase in latency when excluding the 255ms measurement. Subjectively the voice quality was exactly the same as in the single hop measurements, and it was hard to notice an increase in latency.

5.3.3 Single hop in not optimal conditions

The final voice measurement was done inside a dense forest near Bygdøy. Here, we measured the voice communication range when there were several obstacles between each radio. The measurements gave us 186 meters and 190 meters (as seen in Figure 5.4) in two different locations. At these distances we could easily communicate (good quality), and the latency was comparable to the single hop in optimal conditions measurements. We specifically noticed that walking behind a small hilltop would completely block the signal, even at very low distances (90 meters).

![Figure 5.4: Single hop in non-optimal conditions distance measurement location. Image taken from Google Maps](image-url)
5.4 Chat message measurements

Latency and packet order does not affect the quality of chat messages. Therefore these measurements only include the distance at which two nodes can communicate with chat messages. As a note, these measurements are only affected by the environment and the capabilities of the XBee module, and have little to do with the XBee Terminal app.

Transmission parameters are:

- We did the measurements on an open field with both radios having line of sight to each other.
- These measurements were between two nodes (single hop).
- Packets were sent with a data size of 15 bytes.
- Transmit retries is kept at the default value of 10.
- Both radios were kept about 1.5 meters above ground and facing each other.

<table>
<thead>
<tr>
<th>Distance</th>
<th>Packets sent</th>
<th>Packets received</th>
<th>Transmission failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>800 m</td>
<td>50</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>850 m</td>
<td>50</td>
<td>39</td>
<td>11</td>
</tr>
<tr>
<td>900 m</td>
<td>50</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>950 m</td>
<td>50</td>
<td>6</td>
<td>44</td>
</tr>
<tr>
<td>1000 m</td>
<td>50</td>
<td>0</td>
<td>50</td>
</tr>
</tbody>
</table>

5.5 Voice communication latency analysis

The latency in voice communication using the XBee Terminal app is quite noticeable compared to the latency in mobile voice communication. In mobile voice communication, a latency of more than 200ms is often considered uncomfortable. This section summarizes and explains some of the causes for voice communication latency in the XBee Terminal app.

Figure 5.5 displays and explains the components which contribute to voice communication latency.
In order to get a better understanding of the latency caused by the transmission layer (Figure 5.5) and the latency caused by recording and playback, some simple tests were done by comparing the latency in voice communication against the latency in chat communication.

First, I measured the latency in voice communication at a 1 meter range (single hop), inside a building, using the same method as explained in section 5.2.1. The numbers shows an average latency of 20 peak amplitude measurements:

- Sending from the Samsung Neo S5 to the Huawei 4X had an average audio latency of 249ms. The minimum latency was 239ms and the maximum was 391ms.
- Sending from the Huawei 4X to the Samsung Neo S5 had an average audio latency of 535ms. The minimum latency was 516ms and the maximum was 544ms.
The second test measures the time it takes for a Transmit Request Frame to be sent from one Android device to when it is received as a Receive Packet Frame at another Android device (transmission layer and frame buffering in Figure 5.5). These measurements were also done at a 1 meter distance (single hop), inside a building and using GPS synchronized clocks. Packets contained 15 bytes of random data and were sent with a 20ms delay between each other to simulate how voice data is transferred.

- In these measurements the latency when sending and receiving between the devices (Huawei 4X and Samsung Neo S5) was approximately the same, with an average latency of 36ms. Maximum latency was 73ms and minimum latency was 29ms.

Table 5.1 compares the latency for the most important contributors to latency, for both smartphones. These numbers are on an average of the measurements I did, and they exclude some parameters (processing time outside of encoding/decoding). The figure shows that in a single hop over the DigiMesh network, most of the latency in voice communication is caused by playing and recording audio.

<table>
<thead>
<tr>
<th></th>
<th>Samsung Neo S5</th>
<th>Huawei Honor 4X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total latency</td>
<td>249ms</td>
<td>535ms</td>
</tr>
<tr>
<td>Transmission latency</td>
<td>36ms</td>
<td>36ms</td>
</tr>
<tr>
<td>Encoder + encoder buffering latency</td>
<td>20,46ms</td>
<td>20,56ms</td>
</tr>
<tr>
<td>Decoder latency</td>
<td>0,19ms</td>
<td>0,21ms</td>
</tr>
<tr>
<td>Recorder + Player latency</td>
<td>192ms</td>
<td>423ms</td>
</tr>
</tbody>
</table>

*Table 5.1: Average measured latency in voice communication compared.*
6 Discussion, conclusion and future work

This thesis started out with the goal of researching alternatives to mobile network communication for smartphones. It presents the different alternative communication technologies available for smartphones and briefly presents how they can be used to communicate with other smartphones. The thesis also briefly presents how the same technologies can be used with an external, more powerful radio. Lastly, the thesis focuses on how the USB port on Android devices can be used to connect to an XBee radio module and how the connected radio module can be used to communicate via speech and text messages.

6.1 Discussion

The main goal of the thesis was to create an app which would allow smartphone users to communicate over longer distances without using any infrastructure-based networks. The measurements show that the range of the XBee-PRO Digimesh 2.4 radio has a greater range than the Wi-Fi on Android smartphones. However, the range was shorter than expected given the advertised range of 1.6km, especially when used for voice communication.

When comparing the measurements done for range in voice and chat communication, chat could reach almost twice the range of voice. The most important factor for this difference is that the MAC Transmit Retries were set lower in voice communication. In order to get better range, the MAC Transmit Retries can be increased. However, having a high MAC Transmit Retries setting could result in worse quality of voice communication with the current implementation of the XBee Terminal app. Over longer distances, with more transmit retries, the problems would be:

- Packets would sometimes be received in a different order than they were scheduled when they were sent. This would cause the audio samples received to play in a different order which results in incomprehensible voice.
- The average latency would be higher and jitter would become a problem. This would result in worse quality of communication.
- XBee buffer overflow. This has not been discussed earlier in the thesis, but it means that if the XBee radio receives more data via its UART than it can transmit wirelessly,
it will drop frames. This essentially means that the signal received is too weak to transfer the amount of data needed for voice communication.

In addition, as seen in section 5.4, most of the latency in voice communication is caused by recording and playing the audio. Recording and playing audio in the XBee Terminal has been implemented using the `AudioRecord` and `AudioTrack` classes in the Android API which has a substantial latency. Instead of using these classes, OpenSL ES \[74\] can be used through the Android NDK (Native Development Kit) and should provide a lower latency when playing and recording audio. Furthermore, according to an article \[75\], Android devices of version 5.0 and higher provides a lower latency when recording and playing audio compared to lower versions of Android. This can also be noticed by comparing the measurements (Section 5) when the Samsung Neo S5 (Android 5.1) was the receiver and when the Huawei 4X (Android 4.4.2) was the receiver.

I was unable to do measurements for more than two hops over the Digimesh network, which makes it hard to say if voice communication is possible over several hops. The double-hop measurements does however, show a slight increase in latency.

### 6.2 Conclusion

The legal limitation to the radio spectrum made it harder to use and to develop for radios operating on frequencies outside of the ISM bands. ISM band based radios have drastically lower range compared to lower frequency radios with higher ERP. This is why I chose to implement the app for the relatively easy-to-use XBee radio module.

The communication range using the XBee-PRO Digimesh 2.4 is rather low, however, it should be easy to configure another XBee DigiMesh radio and use it with the XBee Terminal. The ideal radio to use with the XBee Terminal right now is the XBee 900HP.

The XBee Terminal app can be a useful tool for communicating in rural areas as a cheaper alternative to other solutions such as the GoTenna or Beartooth Walkie-Talkie. However, long range communication with the XBee Terminal using the XBee-PRO Digimesh 2.4 module is not possible. Compared to Walkie-Talkies, HAM radios GoTenna or Beartooth, the only
real benefit with this application is that allows the use of mesh networking. There may be some use cases for the XBee Terminal app, however its use is limited by the low communication range and its high latency in voice communication.

However, I believe that the research done in this thesis can be useful for others who are interested in making a similar type of application.

6.3 Future work
First and most importantly, recording and playing audio should be implemented using OpenSL ES in order to get acceptable latency in voice communication.

With lower audio recording and playback latency, the MAC transmit retry value could be increased on the XBee-PRO Digimesh 2.4 module in order to achieve longer range communication. Over longer ranges, wrong packet order and jitter would become a bigger problem in voice communication. Wrong packet order can be solved by numbering packets before they are scheduled for transmission, and then buffering and re-ordering packets on reception. Jitter can also be solved by buffering packets on reception. If the latency caused by playback and recording is substantially reduced, this buffer should be easy to implement and allow for better voice communication over longer distances.

The XBee-PRO Digimesh 2.4 radio module also comes with several features which are not used to its full potential in the XBee Terminal app. These are features for sleeping mode, to save power and network encryption for more secure and private communication.

Further, there are several other features which may be included in the XBee Terminal app. These are my ideas, ranging from most important to least important:

- Most of the logic (USB communication, frame buffering etc.) should be done in a background running Servic\textsuperscript{[76]}. The Service can then notify the user of an incoming chat message or voice call, even when the MainActivity is not open (when the screen is turned off or when the app is in the background).
• An interface where the user could read and write settings to the XBee radio. This includes the settings for MAC Transmit Retries, Mesh Transmit Retries and much more.

• An interface where the user can test the data rate between two nodes.

• Each node on a Digimesh network could broadcast its coordinates acquired by the GPS on the Android device. There could also be an interface for a map which displays the location of each node.

• In addition to the USB Accessory and USB Host mode solutions, the app could also be able to communicate with the XBee radio through Wi-Fi and Bluetooth. With this added functionality, the user could choose what connection method is more beneficial for his/her use case. It does however, require some sort of microcontroller and an extra battery (as in the USB accessory solution) to exchange data between the added Wi-Fi/Bluetooth chip and the XBee radio.

• The app could implement communication with other protocols than Digimesh, such as the ZigBee protocol. However, maybe a better idea for a future app would be to implement a more general radio terminal app which is not restricted to Digimesh
Chapter 7: Dual-SIM Signal Strength Logger

One of the goals of the thesis was to be able to map the coverage of different mobile providers in local areas. In order to do this more efficiently an idea was to use an Android phone with two SIM-cards from different network providers and log the signal strength of both providers at the same time.

This app is a project in itself and I dedicated this chapter for the dual-SIM logger. There was mainly one problem in developing the logger app and this chapter will go through the important points. This chapter only explains how I created the app as I did not have the time to actually map any mobile coverage.

7.1 Existing dual-SIM signal strength loggers

For Android phones there are several advanced apps with features for logging, analyzing and representing the mobile signal strength. These apps are, for the most part, limited to Android smartphones with one SIM card. The apps that allows for dual-SIM logging are also limited to Android of version 5.1 and newer. Older versions of Android only support retrieval of the signal strength of one SIM card through the standard Android API. One app that supports dual SIM logging is the app Mobile Signal\[^{14}\] on Android 5.1 and newer. The problem is that at the moment there are very few dual-SIM phones that runs Android 5.1 or newer.

7.2 Dual-SIM API

For Android versions older than 5.1 there is no built in dual-SIM or multi-SIM API. This means that smartphone developers using older than Android version 5.1 had to modify the Android API or create their own multi-SIM API. By modifying the API the developers created features for calling, sending text messages, accessing dual-SIM settings and all kinds of different features one expects from a dual-SIM phone. Since the developers had no standard Android API to implement they made their own API with their own ways of accessing these features. The multi-SIM functionality can therefore not be accessed through the standard Android API. In other words: classes, methods and variables for the dual-SIM API are not accessible without special developer tools, created and used by smartphone developers. This also
means that each of the classes, methods and variables in the dual-SIM are accessed differently on each device.

This section will explain how I used a decompiler to find this hidden dual-SIM API and how I used Java Reflection to access the hidden API on the dual-SIM phones: Huawei Honor 4X Che2-L11 running Android 4.4.2, and Samsung GT-S7392 running Android 4.2.2.

7.2.1 Android Signal Strength

In order to make a dual-SIM signal strength logger app I needed one important function, the function for getting the mobile signal strength. In standard Android programming, this is a very straightforward task. From any method inside an Activity this code may be used:

```java
TelephonyManager telephonyManager = (TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE);
telephonyManager.listen(new PhoneStateListener(){
    @Override
    public void onSignalStrengthsChanged(SignalStrength signal){
        int gsmSignalStrength = signal.getGsmSignalStrength();
    }
}, PhoneStateListener.LISTEN_SIGNAL_STRENGTHS);
```

This code snippet uses the `TelephonyManager` class[77] which gives access telephony functionality such as the `listen` method (line 4). The `listen` method takes a `PhoneStateListener` object which handles changes in signal strength. Inside the `PhoneStateListener` object there is a method called `onSignalStrengthsChanged` (line 6) which is invoked every time there is a change in signal strength by providing a `SignalStrength` object. This object provides information such as the GSM signal strength and the GSM bit error rate. Further, this information can be displayed on the screen or be logged to a file together with a timestamp, coordinates and other relevant parameters.

On dual-SIM phones this method can also be used, but will only provide the signal strength of the first SIM card. My task from here was to figure out how to get the signal strength from both SIM cards.
7.2.2 Java Reflection

Java Reflection\(^7^8\) is an advanced feature of the Java language which can be used to examine, access and modify classes, constructors, methods and fields at program runtime. Java Reflection enables an Android app to access features on an Android device which are not available through the standard Android API. The subsections below will explain and give examples on how Java Reflection can be used. The examples use standard Android classes (which is not a part of a dual-SIM API). Additionally they do not include try/catch statements and object initialization to make the examples shorter. Section 7.2.3 will explain how I used Java Reflection to get the signal strength of both SIM-cards on my dual-SIM phones.

7.2.2.1 Classes

Classes are known from Java and other object-oriented programming languages. In Java Reflection there is a special type structure called \texttt{Class} which is different from standard classes. The \texttt{Class} type provides methods for interacting with the methods, constructors and fields of any standard Java class. The code snippet below shows 3 methods for instantiating a \texttt{Class} type through Java Reflection.

\begin{verbatim}
1. Class cTelephonyManager1 = TelephonyManager.class;
2. Class cTelephonyManager2 = telephonyManager.getClass();
3. Class cTelephonyManager3 = Class.forName("android.telephony.TelephonyManager");
\end{verbatim}

\textit{Code snippet 7.2: instantiating a Class with Java Reflection example}

The first line gets the \texttt{Class} object directly from the \texttt{TelephonyManager} class, while the second method gets it from an object instance of the \texttt{TelephonyManager} class. The third method uses the complete class name (package and name) to get the \texttt{Class} object which represents the \texttt{TelephonyManager} class. The last method can be used to access classes which are not available through the standard Android API. It can find classes which are on the system of the Android device. The downside of this method is that it requires the complete class name, which cannot be found through Java Reflection. How I found these names is explained in section 7.2.3.
7.2.2.2 Objects and constructors

Based on the Class type, an Object type can be instantiated. This is done through a Constructor type. The snippet below shows how the Constructors of the Class PhoneStateListener is retrieved and how a Constructor is used to instantiate an object.

```java
1. Class cPhoneStateListener = PhoneStateListener.class;
2. Constructor[] constructors =
3.     cPhoneStateListener.getConstructors();
4. Constructor constructor = constructors[0];
5. Object phoneStateListener = constructor.newInstance();
```

The phoneStateListener Object on line 5 can then be used as any other Java object. The difference is in how one accesses the methods and fields.

7.2.2.3 Methods

A Method type can be instantiated in a similar manner to the Constructor type:

```java
1. Method[] methods = TelephonyManager.class.getDeclaredMethods();
2. for (Method method : methods) {
3.     if (method.getName().equals("isSmsCapable")) {
4.         boolean isSmsCapable = method.invoke(telephonyManager);
5.     }
6. }
```

This code iterates through the methods in the TelephonyManager class, looking for the method with the name: isSmsCapable. The first parameter in the invoke method is the object which should invoke the method. In this case the object telephonyManager invokes the method isSmsCapable. Any additional parameters provided to the invoke method are passed into the isSmsCapable method itself (in this example there are no parameters).
Chapter 7: Dual-SIM Signal Strength Logger

7.2.2.4 Fields

Lastly, the Field type can be used to access member variables of an object:

```java
1. Field[] fields = SignalStrength.class.getFields();
2. for (Field field : fields){
3.   if (field.getName().equals("mGsmSignalStrength")){
4.     field.setAccessible(true);
5.     field.setInt(signalStrength, 100);
6.     int iSignalStrength = field.getInt(signalStrength);
7.   }
8. }
```

Code snippet 7.5: Accessing fields of an object with Java Reflection example

This code snippet searches the SignalStrength class for the private field mGsmSignalStrength. The code on line 4 enables access to private and protected variables which are normally accessed through a getter and setter method. On line 5 and 6 the setInt and getInt methods are illustrated. Both of these methods use the signalStrength object as the target for field to access. The setInt method takes a parameter which specifies what the mGsmSignalStrength field should be set to.

7.2.3 Finding the dual-SIM API

Knowing how Java Reflection can be used to access device specific functionality, I needed to find the API for dual-SIM signal strength on my test devices. Since there is no available developer documentation for my devices, I tried to decompile the system files into something understandable. This step was the hardest part in making the app, and it was pretty much a trial and error process which turned out to work. Names for classes and methods used in the app are provided for each device in section 7.2.3.2 and section 7.2.3.3.

6.2.3.1 Retrieving framework files

In the standard Android API the signal strength is retrieved by using the classes: TelephonyManager, PhoneStateListener and SignalStrength. I found these classes to be located in the framework.odex and framework2.odex files3, which are located in the System/Framework directory on Android devices. On some devices these files may be

---

3 Dex files are executables and odex files are optimized executables for the Android virtual machine Dalvik. The odex format is typically used by system applications that require fast device startup.
named differently. For example my Huawei Honor 4X has files called *hwframework.odex* and *hwcustframework.odex*. Extracting system files from a device requires root access and is best done by using a computer with Android Debugging Bridge (ADB) [73]. Using a PC, the Android system files can be pulled from the device into the folder “destination”, using this command with ADB:

```
adb pull system destination
```

### 7.2.3.1 Decompiling

The next step is to decompile the *odex* files into readable Java code. The Java code obtained by this method is not identical to the original code, but can give some pointers to how the Android system works. Decompiling was done on a Windows computer with three tools: *smali/baksmali* [79], *dex2jar* [80] and *JD-GUI* [81].

To summarize how each tool was used:

- Smali/baksmali was used to convert the *odex* files into *dex* files. This process is often called *deodexing* by the Android community.
- Dex2jar was used to convert the *dex* files into *jar* (Java Archive) files.
- JD-GUI was used to decompile and view the source code from the converted *jar* files.

The next section is a short summary of what smali/baksmali is and what it can be used for.

### 7.2.3.1.1 Smali/baksmali

*Baksmali* is a program that can disassemble *dex* and *odex* files into the format *smali*. The *smali* format is somewhat humanly readable and can give pointers to certain functionality. Class, method and field names can easily be viewed, but the code within each method can be hard to understand. Further, the *smali* files can be reassembled into *dex* or *odex* files with the *smali* tool. People who modify certain functionality of their Android device often modify the *smali* files before reassembling them and installing the *odex* file back on their device. The app *DexDump* [82] uses the smali/baksmali library and can be used to display *odex*, *dex* and *apk* files on the Android device.
Chapter 7: Dual-SIM Signal Strength Logger

7.2.3.2 Samsung GT-S7392 Dual-SIM API

By decompiling the framework files on the Samsung GT-S7392 I found a class named MultiSimTelephonyManager. This class is used to access telephony specific functionality for each SIM card. In comparison to the standard Android class TelephonyManager, which is instantiated through the getSystemService method, the MultiSimTelephonyManager is instantiated through a constructor.

Code snippet 7.6 shows how Java Reflection is used to instantiate the MultiSimTelephonyManager Class and its Constructors. This class has two constructors: one which takes the Context (interface to global information about the application) as a parameter and one constructor which takes the context and an Integer parameter. The latter constructor is the one used to instantiate an object, as seen in the code snippet (line 9 and 12). This constructor is used to instantiate two objects, one for each SIM card, each defined by the integer parameter.

```
1. Class MultiSimTelephonyManagerClass =
2.    Class.forName("android.telephony.MultiSimTelephonyManager");
3. Constructor[] constructors =
4.    MultiSimTelephonyManagerClass.getDeclaredConstructors();
5. for (Constructor constructor : constructors) {
6.     if (constructor.getParameterTypes().length == 2){
7.         Object multiSimTelephonySIM1 =
8.             constructor.newInstance(context, 0);
9.         Object multiSimTelephonySIM2 =
10.            constructor.newInstance(context, 1);
11. }
12. }
```

Code snippet 7.6: Instantiating the MultiSimTelephonyManager on Samsung GT-S7392

These objects may then be used to retrieve the signal strength of each SIM card by using the listen method through Java Reflection with a standard PhoneStateListener object.

7.2.3.3 Hauwei Honor 4X Che2-L11 Dual-SIM API

After decompiling and looking at the source code of the Huawei Honor 4X che2-L11 I noticed that dual-SIM signal strength is accessed in a different way compared to the Samsung phone.
This method requires two specific classes which are accessed through Java Reflection. The complete class names are:

- “android.telephony.PhoneStateListener”, which is a modified version of the standard Android PhoneStateListener
- “android.telephony.MSimTelephonyManager”, which is the Dual-SIM equivalent of the standard Android TelephonyManager

The modified PhoneStateListener class has an additional private field called “mSubscription”. This field decides whether the PhoneStateListener should listen to the signal strength of the first or second SIM card by setting its value to 0 or 1. Since the modified PhoneStateListener overwrites the standard Android PhoneStateListener, it can be instantiated with standard Java code. However, in order to access the “mSubscription” field Java Reflection is needed:

```java
1. Class PSLClass =
2. Class.forName("android.telephony.PhoneStateListener");
3. Field field = PSLClass.getDeclaredField("mSubscription");
4. field.setAccessible(true);
5. field.setInt(phoneStateListener, 1);
```

Code snippet 7.7: Accessing the mSubscription field in order to listen to SIM card 2 signal strength on Huawei 4X Che2-L11

The MSimTelephonyManager object is instantiated in a similar manner to Code snippet 7.6. Lastly the listen method is invoked on the MSimTelephonyManager object:

```java
1. Method listenMethod;
2. Method[] methods =
3. MSimTelephonyManagerClass.getDeclaredMethods();
4. 
5. for (Method method : methods){
6. if (method.getName().equals("listen"))
7.   listenMethod = method;
8. }
9. 
10. listenMethod.invoke(mSimTelephonyManager, phoneStateListener,
11.   PhoneStateListener.LISTEN_SIGNAL_STRENGTHS);
```

Code snippet 7.8: Invoking the listen method through Java Reflection
7.3 Implementation

Most of the time developing the dual-SIM logger app consisted of figuring out how to retrieve the dual-SIM signal strength. Considering the time spent on this, I was unable to implement all the functionality one could expect from a logger app. However, the core functionalities for displaying and logging signal strength and coordinates are in place.

The app has a background running Service which takes care of all the data collection and logging. The Service has a connection to the user interface Activity which displays the data and gives the user some options on how the app should display or log data.

The app was written using the standard Android framework, specifically for the smartphones Huawei Honor 4X che2-L11 and Samsung GT-S7392. The app can be installed on other Android smartphones, but will then not be able to access the dual-SIM signal strength API.

7.3.1 User interface

The user interface consists of three interfaces. The overview interface displays the signal strengths (Figure 7.1), the logger interface is used to enable and disable logging (Figure 7.2) and the about interface shows some information about the app (Figure 7.3). Each of these screens is controlled by a Fragment, respectively the OverviewFragment, the LoggerFragment and the AboutFragment. The Fragments does the logic behind displaying data and gives the user options to enable or disable logging. The fragments are controlled by an Activity class called DualSimLoggerActivity, which is the main user-facing component. The DualSimLoggerActivity switches between the Fragments and does the initialization and communication with the background Service. The DualSimLoggerActivity receives changes in signal strength and coordinates from the Service and notifies the OverviewFragment. When logging parameters change or the user enables or disables logging, the DualSimLoggerActivity sends a message to the Service which acts accordingly.
7.3.2 Output format

The logger output file is stored on the SD card in a folder called “DualSimLogger”. The file contains a header for each field and is written in the comma separated format. Figure 7.4 shows an example output file imported in Microsoft Excel. This example displays the ASU (Arbitrary Strength Unit) levels and the dBm (decibel-milliwats) of both SIM cards. Notably, the first SIM card uses LTE technology and the second SIM card uses GSM technology. The time field displays the hour, minutes and seconds of the day.

<table>
<thead>
<tr>
<th>Time</th>
<th>Latitude</th>
<th>Longitude</th>
<th>Altitude</th>
<th>SIM1-GsmDbm</th>
<th>SIM2-GsmDbm</th>
<th>SIM1-LteDbm</th>
<th>SIM2-LteDbm</th>
<th>SIM1-AssLevel</th>
<th>SIM2-AssLevel</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:35:41</td>
<td>59.9541212505846</td>
<td>10.7301333292934</td>
<td>146.68762214207809</td>
<td>0</td>
<td>-97</td>
<td>0</td>
<td>51</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>15:35:51</td>
<td>59.9541212505846</td>
<td>10.7301333292934</td>
<td>146.68762214207809</td>
<td>0</td>
<td>-79</td>
<td>0</td>
<td>53</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>15:36:01</td>
<td>59.9541212505846</td>
<td>10.7301333292934</td>
<td>146.68762214207809</td>
<td>0</td>
<td>-70</td>
<td>0</td>
<td>53</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>15:36:12</td>
<td>59.9541212505846</td>
<td>10.7301333292934</td>
<td>146.68762214207809</td>
<td>0</td>
<td>-70</td>
<td>0</td>
<td>53</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>15:36:23</td>
<td>59.9541212505846</td>
<td>10.7301333292934</td>
<td>146.68762214207809</td>
<td>0</td>
<td>-70</td>
<td>0</td>
<td>53</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>15:36:34</td>
<td>59.9541212505846</td>
<td>10.7301333292934</td>
<td>146.68762214207809</td>
<td>0</td>
<td>-70</td>
<td>0</td>
<td>53</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>15:36:45</td>
<td>59.9541212505846</td>
<td>10.7301333292934</td>
<td>146.68762214207809</td>
<td>0</td>
<td>-70</td>
<td>0</td>
<td>53</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>15:36:56</td>
<td>59.9541212505846</td>
<td>10.7301333292934</td>
<td>146.68762214207809</td>
<td>0</td>
<td>-70</td>
<td>0</td>
<td>53</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>15:37:07</td>
<td>59.9541212505846</td>
<td>10.7301333292934</td>
<td>146.68762214207809</td>
<td>0</td>
<td>-70</td>
<td>0</td>
<td>53</td>
<td>21</td>
<td></td>
</tr>
</tbody>
</table>
7.3.3 Signal Strength data management

The Android API provides standard methods for retrieving the signal strength of the device. These methods include several fields, such as the dBm (decibel-milliwatts) of the radios received signal and also the *GSM signal strength* (on a scale from 0 to 100). See [83] for a reference to the *SignalStrength* API.

Most signal strength logger apps retrieve the *GSM signal strength* field when logging, but there are other fields such as the *error bit rate* which may be important in measuring the overall quality of the signal strength. Additionally, some phones such as my Huawei 4X can take use of LTE technology and thus provide additional information about the LTE signal strength. Since the data retrieved may differ from phone to phone, I decided to make each data field optional when logging data (as seen in Figure 7.1). This way the, user may choose what data is relevant and not. Since the Android API does not provide methods for all of these fields, I used Java Reflection to access the *getter* methods from the *SignalStrength* object. This resulted in a rather large list of fields, where most of them may not be relevant.

7.3.4 Background Service

In Android, a *Service* is a process which can continue to run when the screen is turned off, or when the *Activity* is in the background. This important for the app, as most of the logging would happen when the screen is off or when the user exits the user interface of the app.

The *Service* in the Dual-SIM logger app does most of the logic. This includes reading signal strength, reading coordinates and writing log files. The Service is connected to the Activity and sends messages about changes in signal strength and coordinates. It also receives and handles messages about when the user wants to start or stop logging and when the user changes the logging fields.
7.3.5 Location updates

In Android, the location of the device can be provided through the GPS or through the Network Location Provider. The latter one uses Wi-Fi signals and signals from mobile base stations to determine the location. The Network Location Provider tends to use less power, work better indoors and is quicker than the GPS. The GPS on the other hand, tend to be more accurate and works in places where there is no Wi-Fi or mobile signals.

This app would probably be used where there are less mobile signals and no Wi-Fi. The quality of the location from the Network Location Provider may then vary and GPS is therefore used as the default location provider. However, if the user decides to disable GPS, the app will try to use the Network Location Provider instead.

7.4 Summary and discussion

The original goal for this part of the thesis was to create a dual-SIM logger for Android smartphones and use it to map the mobile coverage in a few local areas. I did not have time to map mobile coverage, which means that this chapter only includes the implementation for the dual-SIM logger app.

As I did not have an Android with the version 5.1 API, I developed the app to work with my test phones Huawei Honor 4X and Samsung GT-S7932. Since these phones do not have an API for the dual-SIM signal strength, the focus became to figure out how to access the signal strength of both SIM cards. The app will to some sense become insignificant when newer 5.1 Dual-SIM Android smartphones comes on the market. It was however, for me, a very good learning experience in Android programming, Java Reflection and in decompiling Android files.

Java Reflection made the app possible to make, but since it accesses classes, methods and fields specific to these two phones, the app will not be able to access the dual or multi-SIM API of other smartphone types.

The app can be expanded to support more devices, but it requires information about how each device accesses the dual-SIM signal strength. I did look into ways the app itself could
try to find the methods required. For example by guessing class names to find the appropriate classes for dual-SIM API. However, in order to automate this process a much larger sample size of Android smartphones is needed. The smali/baksmali library could also be used to search framework files for the relevant class, method and field names.

7.5 Distribution
The source code for the Dual-SIM Signal Strength Logger app is included in Appendix E. It can also be found on GitHub using this link:

• [https://github.com/Daffern/DualSimLogger](https://github.com/Daffern/DualSimLogger)

The app can be downloaded and installed from the beta section at the Google Play Store:

## 8 Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3G</td>
<td>Third generation wireless mobile telecommunications technology</td>
</tr>
<tr>
<td>4G</td>
<td>Fourth generation wireless mobile telecommunications technology</td>
</tr>
<tr>
<td>ACK</td>
<td>Acknowledgement</td>
</tr>
<tr>
<td>ADB</td>
<td>Android Debug Bridge</td>
</tr>
<tr>
<td>AM</td>
<td>Amplitude Modulation</td>
</tr>
<tr>
<td>AOA</td>
<td>Android Open Accessory</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>AP</td>
<td>Access Point</td>
</tr>
<tr>
<td>BLE</td>
<td>Bluetooth Low Energy</td>
</tr>
<tr>
<td>BSS</td>
<td>Basic Service Set</td>
</tr>
<tr>
<td>CB</td>
<td>Citizen Band</td>
</tr>
<tr>
<td>CDMA</td>
<td>Code Division Multiple Access</td>
</tr>
<tr>
<td>dBm</td>
<td>Decibel milliWatt</td>
</tr>
<tr>
<td>ECS</td>
<td>Emergency Communication System</td>
</tr>
<tr>
<td>EHF</td>
<td>Extremely High Frequency</td>
</tr>
<tr>
<td>FLOPS</td>
<td>Floating-point Operations Per Second</td>
</tr>
<tr>
<td>FM</td>
<td>Frequency Modulation</td>
</tr>
<tr>
<td>FTDI</td>
<td>Future Technology Devices International</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>GSM</td>
<td>Global System for Mobile Communications</td>
</tr>
<tr>
<td>IBSS</td>
<td>Independent Basic Service Set</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>Acronym</td>
<td>Definition</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>IoT</td>
<td>The Internet of Things</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>ISM</td>
<td>The Industrial, scientific and medical radio bands</td>
</tr>
<tr>
<td>ITU</td>
<td>International Telecommunication Union</td>
</tr>
<tr>
<td>JNI</td>
<td>Java Native Interface</td>
</tr>
<tr>
<td>LF</td>
<td>Low Frequency</td>
</tr>
<tr>
<td>LTE</td>
<td>Long Term Evolution</td>
</tr>
<tr>
<td>MAC</td>
<td>Media Access Control</td>
</tr>
<tr>
<td>MANET</td>
<td>Mobile Ad Hoc Network</td>
</tr>
<tr>
<td>MF</td>
<td>Medium Frequency</td>
</tr>
<tr>
<td>ND</td>
<td>Network Discovery (A command for XBee modules)</td>
</tr>
<tr>
<td>NFC</td>
<td>Near-Field Communication</td>
</tr>
<tr>
<td>P2P</td>
<td>Peer-to-Peer</td>
</tr>
<tr>
<td>PCM</td>
<td>Pulse-Code Modulation</td>
</tr>
<tr>
<td>PSTN</td>
<td>Public Switched Telephone Network</td>
</tr>
<tr>
<td>RC</td>
<td>Remote Control</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SHF</td>
<td>Super High Frequency</td>
</tr>
<tr>
<td>SIM</td>
<td>Subscriber Identity/Identification Module</td>
</tr>
<tr>
<td>SMS</td>
<td>Short Message Service</td>
</tr>
<tr>
<td>SPAN</td>
<td>Smartphone Ad hoc Network</td>
</tr>
<tr>
<td>SSB</td>
<td>Single Sideband modulation</td>
</tr>
<tr>
<td>UART</td>
<td>Universal Asynchronous Receiver/Transmitter</td>
</tr>
<tr>
<td>UHF</td>
<td>Ultra High Frequency</td>
</tr>
<tr>
<td>USB</td>
<td>Universal Serial Bus</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>VHF</td>
<td>Very High Frequency</td>
</tr>
<tr>
<td>VLF</td>
<td>Very Low Frequency</td>
</tr>
<tr>
<td>VoIP</td>
<td>Voice over IP</td>
</tr>
<tr>
<td>Wi-Fi</td>
<td>Name for a Wireless Local Area Network technology</td>
</tr>
<tr>
<td>WLAN</td>
<td>Wireless Local Area Network</td>
</tr>
<tr>
<td>WMN</td>
<td>Wireless Mesh Network</td>
</tr>
<tr>
<td>XCTU</td>
<td>XBee Configuration Test Utility</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
Chapter 9: References

9 References


Stephen/publication/237091710_Serval_mesh_software-WiFi_multi_model_management/links/0f317537ca0e4be0a2000000.pdf cited


Appendixes

- Appendix A - Norway Frequency Allocations
- Appendix B - XBee API Frames used in the XBee Terminal app
- Appendix C – XBee Terminal app source code and Arduino Source code
- Appendix D – XBee Terminal package and class overview
- Appendix E – Dual-SIM Signal Strength Logger app source code