Requirements Specification

2 Dec. 2019

Team Lora

Community Aware Networks and Information Systems Lab

Dr. Morgan Vigil-Hayes (Sponsor)

Scooter Nowak (Mentor)

Ryan Wallace; Benjamin Couey; Mohammed Alfouzan; Brandon Salter

Accepted as baseline requirements for the project:

For the client: _____ Date: _____

For the team: _____ Date: _____

1 Introduction

As the world becomes increasingly reliant on the internet, providing ubiquitous connectivity also becomes vital. Current technology that connects devices over a large area relies on very expensive cell towers or satellites. Due to the cost, these technologies are rarely set up to service rural communities, cutting these people off from the information and opportunities provided by the internet. A new technology, Long Range Wide Area Networks (LoRaWAN), has the potential to change this by providing connectivity that is both far reaching and inexpensive.

Our client, Dr. Vigil-Hayes and her research lab CANIS, have been working with this technology for about a year. Their intention is to take advantage of LoRaWAN's long range in order to increase connectivity in rural areas and support mobile crowdsensing endeavors. In these cases, LoRaWAN will be able to provide the services of a cell tower or satellite connection at a fraction of the cost and power consumption.

Both of theses applications require mobile devices, such as smartphones, be able to connect over LoRaWAN. Traditional network technologies, such as WiFi and broadband transmissions, differ greatly from the underlying technology of LoRaWAN. There is presently no generic interface that would allow a smartphone or similar device to transmit messages over LoRaWAN, making it impossible to interact with any web applications. This project will make it possible for Android phones and, potentially, other devices to communicate over LoRaWAN.

To achieve this, we will be creating an interface for mobile developers that abstracts the process of transmitting a message over LoRaWAN. This interface will be comprised of a library for Android development and a proxy server. The library will encode messages and send them to the LoRaWAN network. These messages will then be received by the proxy server which will decode them and forward them to their intended destination.

This document will begin by examining the problem area in our client's research in greater detail and provide a full description of our solution. It will then specify a list of requirements for this project organized from the highest level requirements down to the lower level requirements. Next, we will consider potential risks to the project, the

impact of these risks on our process, and what we are doing to mitigate them. Finally, we will lay out our plans going forward to implement the aforementioned requirements.

2 Problem Statement

Rural areas experience a lack of network services such as cell service and internet access. The areas that do service experience much slower service and outages. This is due to the cost effectiveness for big companies installing cell towers or servers in these areas; it just isn't profitable for them to support these smaller communities. Thus, these rural areas and communities don't have the same network experience that those in an urban or city environment enjoy.

LoRaWAN is a cutting edge technology that is currently under utilized by both mobile and web developers. While a LoRaWAN network has a similar range of coverage to a cell tower, it is far cheaper. A LoRa Gateway and set of LoRa Nodes cost less than $1500 to configure and install whereas a cell tower costs around $175,000 to build. LoRaWAN is under utilized due to the lack of an easy to use framework that will allow the encapsulation of messages to be sent over LoRaWAN and when received, used in a useful way.

Our client, Dr. Vigil-Hayes envisions using LoRaWAN to enable mobile crowdsensing and provide better connectivity in rural areas. However, there currently isn't an easy-to-use framework or set of libraries that allow for communication from Android or iOS devices to a Lora Node or Gateway. Dr. Vigil-Hayes has tasked us with creating a framework that will allow the encapsulation of dynamic messages to be transmitted from an Android device to a LoRa node that is then sent to a LoRa Gateway to be used in a useful way. This will require us to create many supporting libraries for the envisioned LoRa-Android Framework and to document it so that future developers are able to implement and extend it.

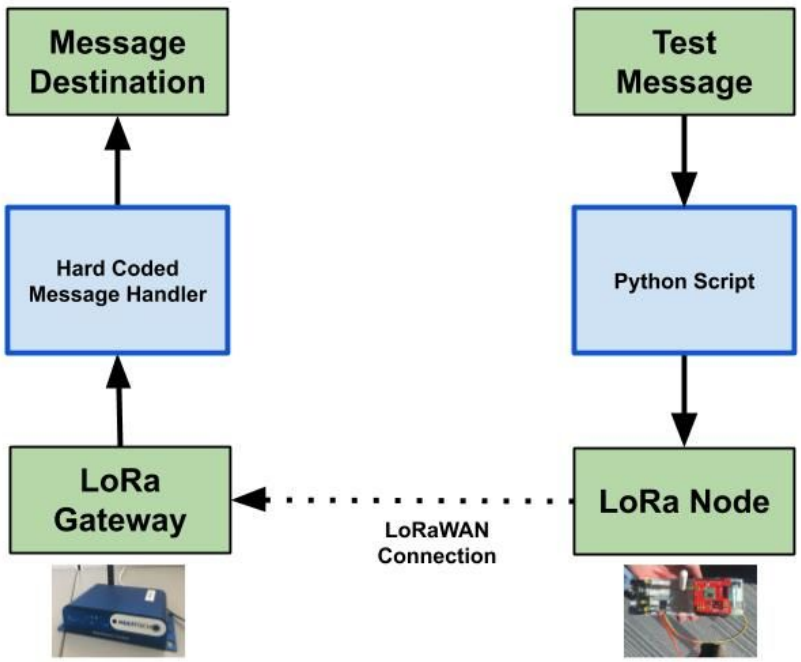

Figure 1: This shows the current workflow of the CANIS lab's basic test architecture.

3 Solution Vision

To solve our client's problem, we have envisioned creating an extensible Android library that allow smartphone users to send web requests to a LoRa Node. The request will then be sent to a configurable proxy server that can process the web request. Due to LoRaWAN's low throughput, our library will encode a message in such a way that they can fit on the LoRaWAN connection. After the encoded message is received by the LoRa Gateway, the message will be sent to the proxy server. This proxy server will decode the message and then forward it to the intended destination. To prove the efficacy of our solution, we will be creating a proof-of-concept application which extends the iNaturalist or OpenCellID apps with our library, and configures our proxy server to handle the application's messages.

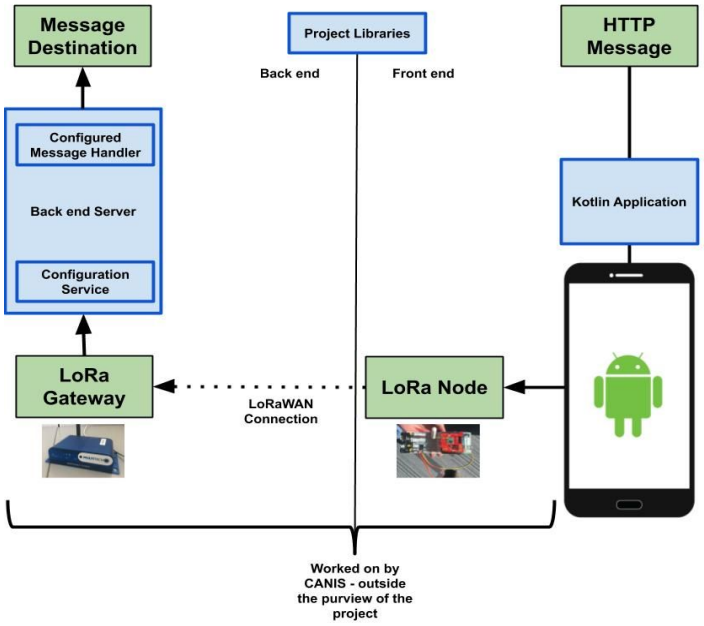

Figure 2: The journey of a message through our implementation

As is shown above, the journey of a message through our implementation starts from an Android smartphone user that sends a message over a LoRaWAN network to the intended destination. The journey begins on:

Android Smartphone:

- A message gets received on a Kotlin application. On the Kotlin application, we will be providing a simple interface for general users along with extensible libraries to facilitate and encode the message in a specific form. The message will be transferred over WiFi to the ESP-32 chip on the LoRa Node.

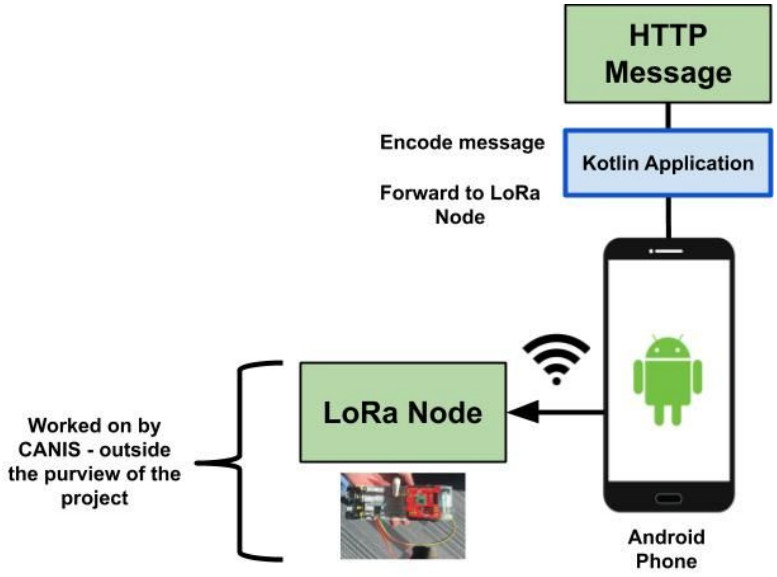

Figure 3: The front end half of the project, where an end-user device, in our case an Android phone, uses the interface of our library to encode a message and then pass it on to the LoRa Node.

CANIS lab responsibilities:

- After the message gets transferred to the LoRa Node, The CANIS lab will be responsible for sending the message from the LoRa Node to the LoRa Gateway over a LoRaWAN connection.

A configurable proxy server that connects to the LoRa gateway:

- The LoRa Gateway will receive the encoded message from the LoRa Node and forward it to the proxy server.
- The configurable proxy server that decodes messages and handles the authentication tokens which are supplied by developers during configuration. The proxy server will then forward the decoded message to its intended destination.

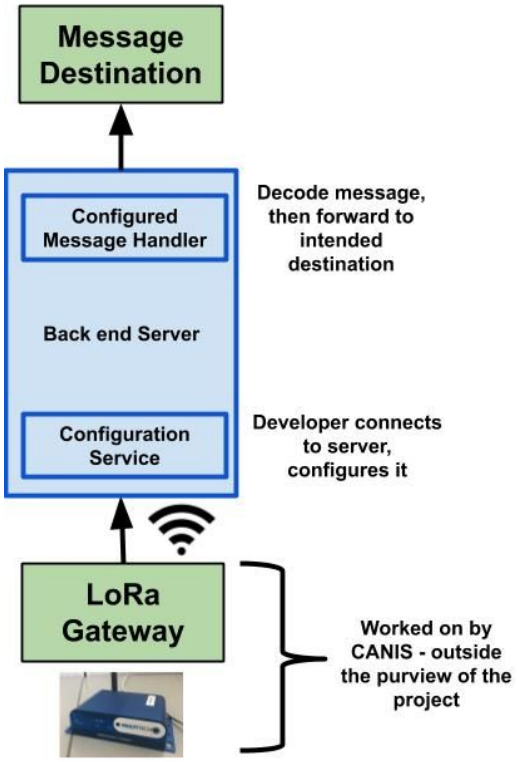

Figure 4: The back end half of the project, where a proxy server receives encoded messages from the LoRa Gateway, decodes them, and forwards them to their intended destination. This proxy server is configured to handle types of messages by a developer.

4 Project Requirements

4.1 Introduction

In order to obtain the requirements for this project, we met with our client on a weekly basis to obtain an understanding of what they wanted us to build. From this, we developed a rough requirements list which was passed back to the client for feedback, and then updated. We repeated this cycle a number of times to further refine our requirements list. Finally, we met with the CANIS lab to gain a deeper understanding of the LoRaWAN technology and the limitations it imposed upon our project. From this, we have determined the following domain level requirements:

- An Android library which allows an application to submit data to LoRaWAN.
- A flexible proxy server which abstracts the process of receiving data on the LoRa Gateway and forwarding it to its intended destination.

- These aforementioned libraries and server will be easy for a developer to use or extend for their mobile projects.
- A proof of concept Android application which implements the aforementioned library and server to connect the iNaturalist and/or OpenCellID applications to LoRaWAN.

4.2 Functional Requirements

Our functional requirements are based upon the requirements list we obtained from meeting with our client. This section is separated into the three main parts of our project: the Android library, the proxy server, and the proof-of-concept application which implements the library and server.

Android Library

1 An Android library which abstracts the process of encoding data and sending it from the phone to the LoRa Node. This will entail;

1.2 Functions which allow an application on the phone to submit data to be transferred over LoRaWAN.

The goal of these functions is to be used in/by services on the Android device that will allow developers to expand on them and implement their own API hooks for applications they wish to tie in to. The functions will be designed with generics to begin with which will make it easier for future developers to implement their own objects in future applications.

1.3 Functions which take this submitted data and encode it by mapping the message's information to bytes in a LoRaWAN packet. This information includes:

The CANIS lab already has a method for encoding messages to fit onto the low-throughput connection of LoRaWAN. We will be using a similar method for how we encode our messages. The encoded message can only be 13 bytes.

1.3.1 The application which submitted the message.

The first byte will show which application is submitting the message. To begin with, we will only support the iNaturalist or

OpenCellID applications. All other applications will be set to same byte ID.

1.3.2 The service which the message is performing.

The second byte will represent which service the application is trying to perform. These services will be API calls for functions supported by the application that sent the message which is represented by the first byte. If the sending application is not iNaturalist or OpenCellID, then this byte block will be ignored for the time being as we are only developing with the two aforementioned applications for this project but this can be expanded on by future developers.

1.3.3 Any arguments passed along with the message.

Depending on the service called by the second byte, some number of the remaining bytes can be used for arguments. This will give future developers plenty of room to expand as needed.

1.4 The above functions must be able to service the basic API calls made by the iNaturalist or OpenCellID application.

As mentioned above, we will be designing our encoded messages to work with iNaturalist or OpenCellID's APIs and all other applications will be ignored for the time being. Our encoded message will be able to use a few API hooks from the aforementioned applications when we are finished.

1.5 If a message is too large to be encoded into a single packet, our library does not need to handle fragmenting the message into multiple packets and can just drop it.

Similar to how the first byte will represent the application that is sending the message, if the library finds that an unsupported application or an application that is supported is trying to send a message that is to large, it will drop the message in instead will report an error to the node which will then be passed along to the gateway.

1.6 Functions which establish a WiFi connection to the LoRa Node and transmit the encoded messages.

Similar to how the first byte will represent the application that is sending the message, if the library finds that an unsupported application or an

application that is supported is trying to send a message that is to large, it will drop the message in instead will report an error to the node which will then be passed along to the gateway.

Proxy Server

2 A flexible proxy server which abstracts the process of receiving data on the LoRa Gateway and forwarding it to its intended destination. This will entail:

2.1 A utility service running on the server which provides an interface that allows developers to connect to the proxy server and configure it. This interface must have:

2.1.1 The ability to accept a secure remote connection from a developer

The proxy server will need to be configured by a developer to recognize the encoding pattern used on messages. To accomplish this, the developer will remotely connect to the proxy server and issue it commands.

2.1.2 An interface to define a type of message to be received. This definition must include:

2.1.2.1 The application which submitted the message.

This information is purely to identify the message as well as its destination. The messages will be passed off to handlers based upon the application which submitted them.

2.1.2.2 The service which the message is performing.

This is to determine how the handler will decode the message. Since messages with different purposes will require different encoding methods, they also require different decoding methods.

2.1.2.3 Authentication tokens for the message

Due to the limitations of LoRaWAN, it is currently infeasible to include an authentication token in the encoded packet sent from the LoRa Node to the LoRa Gateway. For this reason, authentication tokens for messages will be supplied by the developer during configuration of the proxy server.

2.2 A handler for an application which is generated based upon the definitions provided above. This handler must be able to:

2.2.1 Receive messages intended for its application.

Each handler will be responsible for decoding and forwarding all messages associated with an application. It will thus must be able to receive all messages from the LoRa Gateway which were sent by the application the handler was created to serve.

2.2.2 Decode the received message based upon the service the message is performing.

After the message has been received from the LoRa Gateway, the message will be decoded to get the message ready to be sent to its intended destination.

2.2.3 The handler must be able to handle and forward any message associated with its application.

Since the handler services all messages sent by its application, it must be able to service any type of message that application could send.

2.2.3.1 The handler must be able to service the basic API calls made by the iNaturalist or OpenCellID applications.

For now, our client only requires that we extend the iNaturalist or OpenCellID applications. Thus, for now, we need only make handlers for these specific applications.

2.2.4 Collect and manage the authentication tokens for the messages being handled.

Any message sent will require an authentication token which verifies to its recipient that the message was sent from a legitimate

source. These authentication tokens will be provided by the developer during configuration, and the handler will manage the process of assigning these tokens to outgoing messages.

2.2.5 Forward the decoded message to their intended destination.

The handler will be responsible for establishing a secure connection to the destination of a message and sending the message to its intended destination.

Proof of Concept Application

3 A "proof-of-concept" application which extends the iNaturalist or OpenCellID apps and demonstrates the use of the library and proxy server developed above.

We are primarily developing libraries for future developers to use. To prove these libraries efficacy, we will be creating a proof-of-concept application that uses our libraries to help show their functionality. Essentially a bear-bones application that will show of our libraries working with LoRaWAN.

3.1 This application will take messages sent by the iNaturalist or OpenCellID apps, intended to be uploaded to the web server, and, using the library, send these messages to the LoRa Node.

The main goal here is to take messages and data sent from iNaturalist and or OpenCellID and send those messages as packets to the LoRa Node. It is important that these two applications work because our client specifically request these two apps. The messages do not need to be broken up but be sent as one big packet.

3.2 The proxy server will be configured to handle the messages sent by this application.

Once the message has traveled through the LoraWAN we need to handle the sent message on the proxy server. Here we will develop on the back end so it can take these messages and configure them to the correct location.

3.3 Assuming that the messages arrive at the proxy server, it must be able to forward them to the appropriate server on the web.

The back end application will determine the final location for these messages. Once the location is acquired it will send them out to the correct appropriate server on the web.

4.3 Non-Functional Requirements

Our non-functional requirements are based on our client's desire for this project to serve as a tool for future development of mobile applications using LoRaWAN as a network. The section also includes the basic requirement of security common to all networked applications.

1. The interface will maintain the security of data entrusted to it

1.1 The wifi connection between the Android application and the LoRa Node will be secured with encryption

1.2 The wifi connection between the LoRa gateway and the proxy server will be secured with encryption

1.3 The proxy server will securely manage the authentication tokens for messages

Due to the limitations of LoRaWAN, we cannot include authentication tokens in the messages themselves. As such, the authentication tokens will be provided by the developer when they configure the proxy server. To maintain the security of this, developers will only be able to supply authentication tokens over a secure connection.

1.4 The connection between the proxy server and the wider internet will be secured with encryption

2. The interface will be easily usable by future developers

2.1 The codebase will be open source

Since the ultimate goal of this project is to enable further development of mobile applications which use LoRaWAN, we will make the codebase open source so that others may extend or copy our code for their own projects.

2.2 The codebase will use markdown to create a robust reference document hosted on the Github repository

For the same reasons stated above, we will endeavor to make our work easy to understand and use. The reference document created with markdown will be a major component to this.

2.3 The CANIS lab will be able to implement the interface without outside assistance

To verify that our interface is easily usable by future developers, we will be passing the finished project with documentation to the CANIS lab. The expectation is that they should be able to implement our interface to send a message over LoRaWAN. Being experienced with LoRaWAN technology, the CANIS lab is representative of the developers who might want to use our library in the future.

3. The interface will be easily extensible by future developers

3.1 The codebase will be heavily commented and documented to explain design decisions

Many of our basic design decisions will be based upon the work and experience of the CANIS lab. These will be explained in the documentation to provide future developers with the context that led to the finished interface.

3.2 The codebase will avoid an Android-specific implementation wherever possible

While initially the client only wants to prove the viability of this project on an Android platform, later on they want to expand the project to support other platform. By avoiding an implementation which leans heavily on Android, we can make this future goal easier.

4.4 Environment Requirements

When formulating our functional and nonfunctional requirements, we also needed to take into consideration the CANIS Lab and their technology. Environmental

requirements deal with the domain requirements and their interactions with the system and the final product will run on. They are as follows:

4.4.1 Compatibility with CANIS lab LoRa Node:

The CANIS lab is working with the LoRa Node and will be the primary users of our project. With this in mind it is important that our Android library is compatible with their technology.

4.4.2 Compatibility with CANIS lab LoRa Gateway:

Another compatibility requirement we need to keep in mind is having the CANIS lab's LoRa Gateway work with our proxy server. We will endeavor to make the proxy server work with any LoRa Gateway but must make sure our system works with the CANIS lab's hardware.

4.4.3 Proof of concept application will extend iNaturalist or OpenCellID

To demonstrate our working libraries we will be creating a proof-of-concept application. Our client requested this test application would use data from iNaturalist or OpenCellID. This will be a great way to demonstrate our working libraries at the end of the project.

5 Potential Risks

In this section we will discuss the potential risks that our project faces. For each risk, we will also address the likelihood of it occurring, the damage to the project and client it could cause, and what plans we have to mitigate the risk. As this project involves transmitting data over a network, security vulnerabilities are the primary risk we must consider. In many cases, our mitigation strategy will be to employ existing security strategies as ensuring the integrity of network communication is a common challenge.

- User attempts to send a message to the LoRa Node over an unsecured WiFi network
    - If a user connects to an unsecured WiFi network to transmit encoded messages to the LoRa Node, those messages will be vulnerable to bad actors intercepting and reading them. This clearly

jeopardize the integrity of the system as that user's information is now vulnerable, as is potentially the encoding scheme of the library.

    - Likelihood - Moderate - We cannot guarantee that users will avoid using our interface on an unsecured WiFi network. Some will inevitably attempt to use our interface in this dangerous manner.
    - Severity - Low to High - The severity of this risk depends on the sensitivity of the data the user is sending. If the data is just anonymous crowdsensing data (such as network measurements for OpenCellID), the damage done is relatively low. If the data contains the user's personal information, the damage done is quite high.
    - Mitigation Strategy - To mitigate this risk, our library will warn the user about the risks of sending data over an unsecured WiFi network should they attempt to do so.

- Data sent over LoRaWAN becomes corrupted
    - As with all network transmissions, it is possible for the LoRaWAN signal to be disrupted in such a way that it corrupts the data sent over the network. If this occurs, the corrupted packet wi
    - Probability - Moderate - While LoRaWAN technologies are still being developed and researched to determine their exact reliability, it is known that packet corruption is an issue. Moreover, since the client intends to service rural areas with this technology, it is likely they will be operating in more extreme environments where a signal is more likely to be disrupted by weather or the like.
    - Severity - Low to Moderate - The severity of this risk depends on the importance of the data corrupted. If the data is a single sensor reading that is part of a crowdsensing effort, the damage done is relatively low. If the data is part of a user's query to the internet, it being corrupted would worsen their experience and potentially destroy more valuable data.
    - Mitigation Strategy - To mitigate this risk, our library will offer the option to include a single parity bit when it encodes a message. Space in an encoded message is already very tight, and so we must use error-checking which has as little overhead as possible. If a

packet is found to have become corrupted by the proxy server, it will simply drop the message.

- Developers misunderstanding what can be sent over LoRaWAN
    - Due to the limits of LoRaWAN, there are limits to the size of packet which our interface can support. If a developer misunderstands these limitations, they may attempt to use our library to encode and transmit a message which is too large for our interface to handle.
    - Likelihood - Moderate - There are numerous web services and applications which require a far higher throughput than can be managed over LoRaWAN. While our documentation will be explicitly clear about the capabilities of our interface, developers may still attempt to use it for applications it cannot handle.
    - Severity - Low to Moderate - If the developer only becomes aware of our interface's limitations after implementing it, they may find the interface is unsuitable for their application. They will have to spend more time and money modifying our interface or finding another one. The reputation of our interface will also suffer because of this.
    - Mitigation Strategy - To mitigate this risk, our library will be designed so that, if it is given a message to large for it to support, it will fail gracefully. The library will alert the developer that the message is too large to be supported but will not impede the sending of other, smaller messages.

6 Project Plan

In this section, we will discuss our plan to carry out the project. We have divided this tentative plan for our project into two parts: Fall and Spring semesters. These two semesters separate the planning and development phases of the project. For this requirements document we will only discuss our project plan for Spring semester.

We have two major tasks for the Spring semester: library development and proxy server development. Library development is when our team will be writing a Kotlin library for the Android OS. These libraries will allow an Android phone to send messages to the

Lora Node. In addition we will be creating extensive documentation for these libraries so future developers can more easily implement and extend them. We plan for these libraries to be done by mid February.

The second major task we will be working on is proxy server development. The proxy server development is primarily getting the message (sent from the Android Phone) to forwarded to its intended destination. We will start the task of working on the proxy server in February and finish at the end of March. It is possible we will need to develop the proxy server in parallel to the development of the library.

6.1 Milestones:

1. Design a Tech Demo
2. Write and Develop our Android Libraries
    2.1. Well documented using a WIKI
3. Write and Develop our Back End to LoraWAN
4. Testing with the CANIS lab
    4.1. Developing a "Bare-bones Test Application" that utilizes our libraries.
5. Deploying the Libraries and Code to our client.

6.2 Gantt Chart

This Gantt chart (see Fig. 5) displays the schedule of our capstone project. We have three phases for next semester, Library Development, Back End Development, Test App Development / Finalizing. The red line near the top represents where the team is currently in the timeline. Each task blow is represented by a colored box that shows the duration of how long we should be working on each task. Though this chart is obviously subject to change, especially on exact due dates, the overall progression should remain constant.

Figure 5. Gantt Chart for Team Lora.

7 Conclusion

As our world becomes increasingly networked, lacking access to the internet becomes an increasingly debilitating position. Many rural communities are in this position, lacking expensive cell towers to connect them to the world. Our client seeks to solve this problem with the new technology LoRaWAN by providing a cheap and power-efficient option which could connect rural areas and enable mobile crowdsensing. A barrier to this is the lack of an easy-to-use interface that allows a mobile application to communicate over LoRaWAN.

We will supply this interface by creating an Android library and proxy server which, together, abstract the process of transmitting a message over LoRaWAN. The library will provide functions to encode and transmit a message to the LoRa Node. Meanwhile, the proxy server will be able to receive messages from the LoRa Gateway, decode them, and forward them to their intended destination.

In this document, we clearly and carefully defined the baseline requirements of this project. As the project progresses, this document will serve as the template to which we build our solution. We also anticipated a number of risks to the project's future, determined their likelihood of occurring and severity of damage, and created strategies to mitigate their impact.

Looking to the future, our project is proceeding steadily. We have begun prototyping both the Android library and proxy server and have obtained access to the LoRa Node

and LoRa Gateway used by the client's research lab. Additionally, we have set up channels of communication so that we may work with the CANIS lab more closely as we begin implementation. This should ensure that our project remains compatible with their work while also allowing the project to benefit from their experience with the LoRaWAN technology. We believe that our interface will solve an important problem for our client's research and help further the spread of mobile crowdsensing and LoRaWAN.