

Software Testing Plan

3 April. 2020

Team Lora

Version 1.0

Community Aware Networks and Information Systems Lab

Dr. Morgan Vigil-Hayes (Sponsor)

Scooter Nowak (Mentor)

Ryan Wallace; Benjamin Couey; Mohammed Alfouzan; Brandon Salter

[**1 Introduction**](#_n0bwdk47l47e) **3**

[**2 Unit Testing**](#_h35xpc45v5e9) **3**

[2.1 Introduction](#_jriz60k32nt8) 3

[2.2 Encoding and Decoding Subprocess](#_gjte0nck91ik) 4

[2.3 createEncodingTable and createDecodingTable](#_sb1clq70fa4x) 5

[3 Integration Testing](#_wihssw6c8d95) **6**

[3.1 Introduction](#_wihssw6c8d95) 6

[3.2 LoRaMessenger to Proxy Server](#_wihssw6c8d95) 7

[3.3 LoRaMessenger Library](#_qozzg9g5rvr9) 7

[**4 Usability Testing**](#_w3a1bt7774nu) **8**

[4.1 Introduction](#_1mjmwijqqym6) 8

[4.2 Testing Plan](#_7nz9mk60ok04) 8

[**5 Conclusion**](#_a8d7gy1lgfi3) **9**

# 1 Introduction

This project is a framework composed of a configuration service, proxy server, and the LoRaMessenger Android library. Together, these modules provide an interface that allows a developer to transmit messages over a Long Range Wide Area Network (LoRaWAN) connection without worrying about the complexities of fitting their message into the extremely small packets of LoRaWAN. The framework aims to be easy to use and extend so that it can serve as a stepping stone for future development using LoRaWAN and encourage more applications to make use of the technology’s low cost and long range.

As with any software product, our framework needs to be tested before being released. This testing involves the execution of the software, or components of the software, and evaluating how they performed in various conditions. The broad goal of such software testing is to verify that the software meets the requirements it was built to fulfill and assess the quality of the software under test.

In the case of testing our framework, we have devised a comprehensive testing plan. First, we have highlighted the most vital subprocesses of our framework and will subject these subprocesses to unit tests to verify that they work in a vacuum. Next, we will perform tests on the framework as a whole to verify that all the components are correctly communicating. Finally, we will verify the usability of the framework by [Insert usability testing here] and [Insert usability testing here].

We based our testing plan primarily on the fact that our product is a framework of multiple components which all must work together to accomplish its goals, and that the primary purpose of this project is to facilitate further development with the LoRaWAN technology. For these reasons, integration testing and usability testing are our primary areas of focus. The details of this testing plan are further elaborated on in the rest of this document.

# 2 Unit Testing

## 2.1 Introduction

The first and most basic aspect of our testing plan is to run the components of our framework against a suite of unit tests. Unit tests are small, low-level tests used to verify that a specific part of the code, such as a single function or procedure, behaves correctly for all inputs given to it. To accomplish this we will be separating the main components of our framework into meaningful subprocesses, or units, which can be tested for correctness in isolation of the overall system. To aid us in the running of these tests we will be using JUnit to test the LoRaMessenger library and Pytest to test the proxy server and configuration service. With all this in mind, we will be testing the following units of our project’s codebase:

* The encoding subprocess. This subprocess of the LoRaMessenger library must, given a message and a decoding table, produce an encoded message which follows our format. This unit will be tested in tandem with the decoding subprocess.
* The decoding subprocess. This subprocess of the proxy server must, given an encoded message and a decoding table, produce the original message. This unit will be tested in tandem with the encoding subprocess.
* The createEncodingTable function. This function of the configuration service must produce a correct encoding table when given a properly formatted JSON file.
* The createDecodingTable function. This function of the configuration service must produce a correct decoding table when given a properly formatted JSON file.

Below we will go into greater detail about our plan for testing each unit, as well as describe the equivalence partitions we identified in the inputs for that unit. An equivalence partition is a group of inputs for which we expect reasonably similar behavior, and it is important that our unit tests hit all equivalence partitions. We will also propose some erroneous inputs to give the unit to verify the unit’s robustness and ability to fail gracefully.

## 2.2 Encoding and Decoding Subprocess

While these two parts of the codebase are distinct units, we believe it makes the most sense to test them together. They perform the inverse of one another and so the most straightforward way to verify their correctness is to perform a round-trip test. In such a test, we will submit an input to the encoding subprocess, then use the encoding subprocess’ output as the input for the decoding subprocess. If the output of the decoding subprocess is equivalent to the original input, then it can be verified that both subprocesses behaved correctly for the input.

One downside of this plan is that, should a test fail, it is not immediately clear which subprocess is at fault. For each failing input, we will need to manually verify the correctness of both subprocess’ outputs to determine which is at fault. However, this is still preferable to manually determining the correctness of output for all inputs. By using the round-trip testing scheme, we can entirely automate the identification of passing inputs.

The input to encoding subprocess consists of two parts: the associated encoding table JSON file and the list of app name, api name, and parameter values which represents the message to be encoded. Similarly, the input to the decoding subprocess consists of a decoding table JSON file and an encoded byte message. For the sake of unit testing, we will be assuming that the encoding and decoding tables are not mismatched. From this set of inputs, we have identified the following equivalence partitions:

* The tables define an integer parameter and the encoding subprocess is passed an integer.
* The tables define a floating point parameter and the encoding subprocess is passed a floating point number.
* The tables define a double parameter and the encoding subprocess is passed a double.
* Enough parameters are defined and passed to the subprocess so as to necessitate fragmentation.

We have also identified the following equivalence partitions as erroneous inputs:

* Either the encoding table or the decoding table are malformed.
* The table defines a parameter and the encoding subprocess is passed a parameter of a different type.
* The table defines an integer parameter to take up X bytes and the encoding subprocess is passed an integer larger than 2^X.
* The subprocess is passed a different number of parameters than the table defines

## 2.3 createEncodingTable and createDecodingTable

The createEncodingTable and createDecodingTable functions produce an encoding and decoding table respectively based on a developer’s definition of their applications and what messages it might send. These functions are run at the same time on the same input by the configuration service and output the resulting tables as JSON files. Due to their similarity, we will test these units together. To do so, we will create a set of developer inputs and manually produce an expected encoding and decoding table for each one. Each input will then be passed to the table creation functions to produce a set of output. The test will loop through the output and the manually produced tables to make sure that both of the files look the same. Finally, the outputs of this test will be used in the unit tests of the encoding and decoding subprocesses. This will give us another chance to verify the correctness of the tables and how the subprocesses are handling them, while also allowing us to quickly test the integration of these two vital units.

As input, the table creation functions take a JSON file which contains a specially formatted definition of an application and its messages. The rules for this formatting are explained in the documentation stored on the project’s wiki. Due to their complexity, these inputs will have to be manually generated. For testing this unit, we have identified the following equivalence partitions with a set of inputs:

* The app definition defines a parameter with a finite set of accepted values.
* The app definition defines an integer parameter with a given byte length.
* The app definition defines a float or double parameter.
* The app definition defines multiple types of messages to be sent.

We have also identified the following equivalence partitions as erroneous inputs:

* The app definition is not a valid JSON file, either due to being malformed or missing.
* The app definition defines multiple message types with the same name.
* The app definition defines a parameter with an empty set of accepted values.
* The app definition defines an integer parameter with a byte length less than one.

# 3 Integration Testing

## 3.1 Introduction

Integration testing is done after unit testing is completed by putting all of the pieces together and ensuring that they work together as anticipated. Our framework exists in two main parts, an Android library and a runnable proxy server. The goal of our integration testing will be to show that LoRaMessenger and the proxy server still function together in several environments. The library and proxy server are very different in that the server is a runnable application whereas the library will be used as a framework inside other runnable applications on an Android device. Due to these two parts being so different, we’ll be testing each of them in different environments. However, we will still be testing to ensure that a message can make it from the Android device to the proxy server to further prove that the given integration on both ends are compatible. We aim to show that our framework can get a message from an Android device to a proxy server at a given IP/Hostname on any system capable of running Python3, on the same network. Our goal is to allow Android developers to take these tools and further LoRaWAN mobile development in a way that could assist researchers. With this goal in mind, we will test our proxy server on several networks and on several systems as to show that research groups could begin to deploy our tools on numerous systems

## 3.2 LoRaMessenger to Proxy Server

The proxy server required to work with LoRaMessenger runs using Python3 so any system that can connect to a standard IP based network and can support Python3, should be able to run our proxy server. To test this theory we will be running our proxy server on several different computers as well as virtual machines (VMs) to ensure that it works on multiple systems. We will additionally test on several home networks as NAU’s network. The proxy server’s job is to receive messages from an Android device and decode them using a configuration service that is run before the proxy server is started up; the configuration service will be bundled with the proxy server package. To verify that the proxy server is functioning correctly in these different environments we will send several messages from our test android device to the proxy server in each environment and if we receive the correct, decoded message, then we will have verified that environment functions as we expected.

## 

## 3.3 LoRaMessenger Library

The LoRaMessenger Library will serve as a framework to be used in Android app development. To prove that this can be implemented in development, we will be designing a test app that functions with OpenCellID and we will test the app on multiple Android devices. If the test app that uses OpenCellID is able to transmit a message that is then received by our proxy server, we will have shown that the library works in that instance.

# 4 Usability Testing

## 4.1 Introduction

Usability testing is centered around accessing how the end user interacts with our software. It seeks to validate that the software is easy to use and promotes a positive experience for the user. Our project's final product is a tool that eases the use of LoRaWAN for traditional mobile applications, our end users will be the CANIS lab as Dr. Vigil-Hayes continues her work with LoRaWAN. Usability testing is critical for our framework, as a primary requirement of the project is to produce a tool which is easy to use and extend. To help satisfy this requirement, we are building a wiki on Github to extensively document the project. Making sure this wiki is thorough and intuitive will therefore be a focus of our usability testing. To help guide our usability testing we have come up with the following goals:

* Test the ease with which a developer can implement the LoRaMessenger library.
* Test the ease with which a developer can configure the configuration service.
* Test the ease with which a developer can configure the proxy server

## 4.2 Testing Plan

Since the end users of our product will be developers familiar to some degree with the use of software frameworks, our usability testing will target senior computer science students and research team members at NAU. These groups should have some degree of technical experience developing software and using frameworks in their development. As our project is a tool for the development of an Android application, it might be expected that users have some experience developing for Android. Of course, since a LoRaWAN-enabled application could still be a user’s first Android project, this is not a hard requirement. Following only the documentation provided in the wiki and in the codebase itself, a user should be able to successfully implement our framework. If at any point a user is lost or unsure about how to proceed this would tell us that our documentation is insufficient.

Our team will reach out to other students at NAU asking them to test our capstone project. We will ask them to develop a proof of concept Android application that utilizes our Android Library while following our Github wiki. We will also ask the user testing to configure and set up the configuration service, along with the proxy server. This should be very straightforward forward as well if our wiki is through enough. The user testing this component will be asked to fill out a survey in regards to how easy it was to follow our wiki and how thorough it was. We will ask them for feedback on any information they feel is missing and how we can improve the structure to improve it.

Our team will reach out to the CANIS lab team members and ask them to test our project using the technology they have in their lab. While their work on LoRaWAN is still incomplete, it would be feasible to ask them to create a simple demo app using our framework. We will also be seeing if they can set up the configuration service and the proxy server. Due to the situation we are unsure if they have access to the CANIS lab equipment. If they do we would ask them to see if they can implement our product with their hardware. If not we will treat them as we did when testing with the NAU Computer Science Students. The goal here is to see if they can follow our wiki because they will be taking this project in and using it once it is done. So it is important that they do not feel lost or confused using our wiki. We will ask the CANIS lab to fill out a survey and give us feedback on how we can improve the wiki. We will schedule a Zoom conference call and they can give us live feedback about how they felt about or project and what changes can be made to better assist the future of our product as used by Dr. Vigil-Hayes’ research teams.

# 5 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. Dr. Vigil-Hayes 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 framework that allows a mobile application to communicate over LoRaWAN.

We will supply this framework 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 detailed our testing plan to verify that our framework behaves as intended and is as bug-free as possible. This plan consists of unit tests for the vital subprocesses of the framework, including erroneous inputs to test our software’s ability to fail gracefully, a set of integration tests to make sure the framework can work together in a variety of environments, and a usability testing scheme where we will enlist senior students and research team members to test our documentation. We have identified usability testing as the most important aspect of this plan since the ease of use and quality of documentation is a key requirement of our project.

Based upon the needs and scope of our project, we believe that this testing plan will succeed in ensuring our framework is as bug free and easily-usable as possible. By testing the encoding and decoding subprocesses in a round-trip fashion while using the results of the configuration service’s unit testing as input, we can automate a majority of the testing of the most important process of the codebase. For our usability testing, by focusing our efforts on the CANIS lab, the future users of the framework, we can tailor it to their needs and produce a truly specialized product. By carefully testing and refining our work, we believe this framework will serve as a valuable piece of our client’s research and serve as a stepping stone for future LoRaWAN development and innovation.