LoRa Mesh Network for IoT Applications

Senior Capstone Project Progress Report

Andrew Courtemanche
Computer Engineering

Josh Lariviere
Computer Engineering

MD Shaad Mahmud
Faculty Advisor

University of New Hampshire
College of Engineering and Physical Sciences

College of Engineering and Physical Sciences
University of New Hampshire
December 2021

Abstract

This document is a progress report for our team's senior capstone project. It aims to explain our project's end goal as well as provide an update as to its current status. We will also discuss the plan for continuing work on the project in the future.

Table of Contents

1 Introduction

The ability to collect real-time environmental data across a large area can allow farmers to more efficiently allocate their limited time and resources, but current solutions are prohibitively expensive. The most popular options still rely on a single gateway to the internet that all sensors must directly connect to. These expensive gateways can only service a limited number of sensors that are within range, and local geography may limit suitable installation locations. Multiple gateways must be purchased in order cover large areas, further reducing the cost effectiveness of such a solution. We seek to design a mesh networking protocol for this and similar low bandwidth applications that need to cover a large geographical area. A mesh network enables low cost, long range, and low maintenance communications without relying on any preexisting infrastructure.

2 Background

2.1 LoRa PHY

The LoRa Physical Layer (PHY) is a chirp-spread-spectrum modulation[1] technique designed for long range, lower power, low bit rate applications. It operates on the unlicensed Industrial Scientific Medical (ISM) 902-928MHz band in the USA. LoRa has three primary parameters: frequency, bandwidth, and spreading factor. The bandwidth parameter defines the range around the center frequency each chirp occupies. The spreading factor parameter defines how long each chirp takes to complete. Higher bandwidths result in a higher data rate and lower sensitivity, while higher spreading factors result in a lower data rate and higher sensitivity [2]. Both sending and receiving nodes need to be configured with the same parameters in order to communicate.

2.2 LoRaWAN

The most widely used protocol built on top of LoRa PHY is LoRaWAN. LoRaWAN implements the data link and network layers of the Open Systems Interconnection (OSI) model and uses LoRa PHY as the physical layer. LoRaWAN uses a star topology, where each node in the network connects directly to one or more network gateways [3].

3 Objectives

The original motivation of this project was to better facilitate the collection of sensor data from nodes spread across a large geographical region. Current

[1]In digital communications, chirp spread spectrum is a spread spectrum technique that uses wideband linear frequency modulated chirp pulses to encode information. A chirp is a sinusoidal signal whose frequency increases or decreases over time.[1]

solutions require each node to be within range of a gateway, which limits deployment flexibility, increases hardware costs, and reduces overall network robustness.

We intend to design a set of protocols that utilizes the LoRa physical layer to implement a low cost mesh packet-switched network. The primary objective is a highly robust "set it and forget it" network that is capable of self configuration. Secondary objectives include low cost and energy efficiency.

4 Overview

4.1 Library Code Overview

The ultimate goal of this capstone project is to create a full LoRa Mesh library to include into any application for most devices. We want to create the library so that it is fully usable with any Integrated Development Environment (IDE) that the user wants. Additionally, we aim to create driver templates for use with the library to allow for easy addition of any sensors.

The library will consist of three "layers" of code[2]:

- Driver
- Hardware Abstraction Layer (HAL)
- Hardware

Each layer is designed so that it only depends on the layer immediately beneath it.

The hardware layer is used by the library to abstract away any hardware-specific configuration such as register addresses, package pinouts, and how peripherals are connected. Each hardware target has its own folder in this layer and contains two files. One of these files contains register definitions, while the other file contains any required software dependencies and pre-processor definitions/macros. These files also ensure that only one hardware target can be used at a time.

The Hardware Abstraction Layer (HAL) provides the driver layer with a consistent interface regardless of the hardware it is running on. It is able to do this through use of the information provided by the hardware layer. The HAL is written in such a way that each of the functions are able to change based on the pre-processor definitions from the hardware layer. To add support for new hardware, one can add hardware information into the hardware layer, then for each function of the HAL you can tailor the function to work with your hardware using the compiler directives laid out.

The driver layer is what houses the drivers for the LoRa Mesh itself and the various sensors to be used with it. Each driver is given its own folder to hold all its necessary components for organization. For the sensor drivers, we aim

[2]LoRa Library GitHub Code Link A

to create abstract driver classes that a user can inherit to easily create a driver for any sensor they would like without hassle. Our team is intending to create drivers as well for all the sensors that are on the IBUG node[3] that this library is being created for.

There is also is a collection of functions and structs in the lib folder that are useful to many other aspects of the drivers code. For example, one struct houses the date and time and can return it with proper formatting. These files mainly simplify the driver layer and increase code re-usability.

Our team aims to create an application using this library as well that will run on the IBUG node[4]. The application will use the library to create a mesh network using the LoRa and mesh drivers created. Then use the sensor drivers made to gather data using the IBUG node sensors to begin the transmission of the sensor data over the mesh network. This data will ultimately end up at a gateway that leads to a central server that collects all the data for analysis.

4.2 Mesh Protocol Overview

The mesh protocol is designed to be as autonomous as possible, requiring minimal configuration to create a large network. Since the transceivers on each node can only listen to a single frequency, spreading factor, and bandwidth, an additional mechanism is required to make sure different nodes are using identical configurations in order to communicate. While the obvious solution is to use a preset configuration across the network, this would dramatically reduce network performance and susceptibility to interference. To work around these issues, each node uses a Global Positioning System (GPS)[5] receiver to synchronize their clocks to a common reference. Transceiver settings are pseudo-randomly generated, using the current date and time, as a seed for the random number generator.

[3]IBUG Node Hardware B.2

[4]IBUG Node Overview B

[5]Global Positioning System, a United States government-owned global navigation satellite system

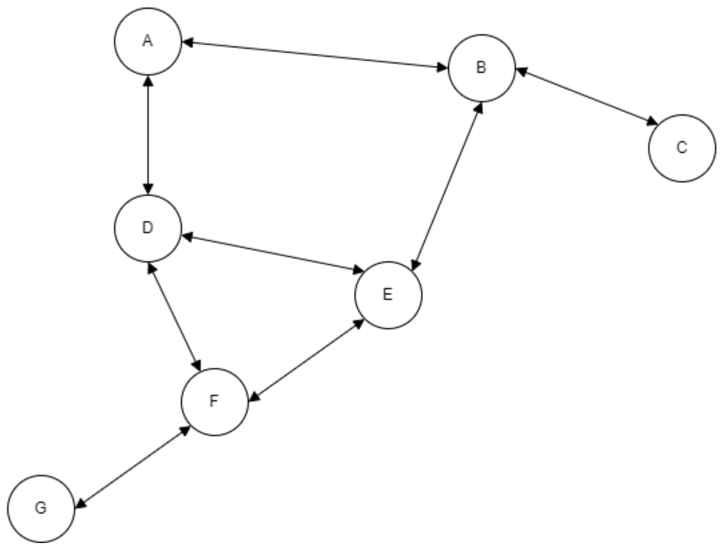


Figure 1: Example mesh topography

Every minute is divided into four periods of 15 seconds each. During each period, the available spectrum is divided into 32x500kHz, 64x250kHz, 128x125kHz, or 256x62.5kHz channels, respectively, with appropriate padding in between each channel. A new channel is pseudo-randomly selected for each of these periods as the global discovery channel. All nodes will be expected to listen to this global channel for requests to join the network, as well as transmit on that channel in regular intervals, to facilitate peer discovery and mesh construction. This random channel selection and bandwidth allocation ensures even spectrum usage and resilience to interference. Transmissions on the discovery channel will also use a pseudo-randomly selected spreading factor.

A network join packet is sent during the discovery phase and contains a random unique identifier for the node, used to address the node. This identifier is stored in a look-up table by all nodes that heard the broadcast join packet. A node uses this number to seed a pseudo-random number generator to select a random channel - not including the current discovery channel - that it will listen for potential transmissions on. If a node would like to communicate with a particular node, it can use the same number to figure out what channel the target node will be listening on.

5 Current Status

Currently, the status of our project is we have completed the hardware and hardware abstraction layers of the code. We have also laid out the full code structure and all necessary components we will need for our library[6]. We are

[6]LoRa Library GitHub Code Link A

currently working on completing the drivers layer of our library. Our team has additionally begun the documentation stage, utilizing the software Doxygen [4], to simultaneously complete while we work on the drivers.

5.1 Initially Proposed Plan

Our teams initial timeline proposal was outlined by our initial Gantt chart[7] we hoped to achieve. We also initially proposed the broad structure we envisioned for our library as follows:

- Server-side application
- Sensor data collection application
  - Sensor interfacing
  - Data logging
  - Compression
  - Encryption
- Mesh networking layer
  - Node discovery and coordination
  - Node to Node communication
  - Packet routing
  - Packet integrity
- Hardware Abstraction Layer (HAL)
- Node hardware design

We intended for each bullet in this list to be a layer of the code in the library with all its sub components listed.

The node hardware design would be for any circuit changes that we deemed necessary. The HAL would be used to allow the library to run specifically on the IBUG node hardware[8]. The mesh networking layer would be used to let the nodes create and access the LoRa mesh network. This layer would house everything the node needed to discover the mesh and interact with it as intended. Next the Sensor application layer was intended to be the application created that runs on the node using the library. This application would be what interacted with the sensors and collected their data, logged said data, compressed it, encrypted it, then sent the neat packaged data to the mesh layer to be sent along. The mesh layer would then eventually get the packaged data to a server running an application to decrepit the incoming data and process it accordingly.

[7]Initial Gantt Chart C.1.1
[8]IBUG Node Hardware B.2

5.2 Deviations from Original Plan

5.2.1 Library Plan Changes

Clearly with the library break down of our projects code in the Overview section[9] some alterations to the library's design have been made. The largest of these changes is the structure and rolls of the library layers:

- Server-side application
- IBUG node application
  - Compression
  - Encryption
- Drivers layer
  - LoRa mesh networking layer
    - Node discovery and coordination
    - Node to Node communication
    - Packet routing
    - Packet integrity
  - Sensor layer
    - Sensor interfacing
    - Data logging
- Hardware Abstraction Layer (HAL)
- Hardware layer

These structure changes were made as our team gained more knowledge of our project and what was needed for it.

We removed the nodes hardware design from our plan due to the issues with current chip shortages. Our team decided that it would not be the best use of time redesigning the nodes as their design was already usable and lead times for fabrication were too long.

For the HAL and hardware layer our team decided that we wanted our library to be able to be compiled and used for any IDE and any hardware a user desired. We believe this will increase the overall usability of our library.

The most noticeable of the structure changes to the library was the addition of the drivers layer. This layer now contains both the LoRa mesh drivers and the sensor drivers. This change occurred because our team realized that having the sensors controlled and talked to in an application was not efficient. We decided that giving each sensor its own driver made it more modular and easier to swap out in the application layer. With this decision having both the LoRa mesh and sensors have their own drivers, we decided that a drivers layer containing

[9]Overview 4

both sets of drivers would be the best course. This layer would allow us to have better organization of the code structure.

From the driver layer decision, this meant that the application layer we wanted would then have to include any of the sensors we wanted to use instead. This would make our intended application much more portable to other devices as well, making it a good example code for our library. We still intend for the application to have the compression and encryption of the sensor data to then send to the mesh sub-layer however.

Another minor change was the use of the lib folder found in the GitHub repository[10]. This folder is just used for helpful bits of code that can be used in multiple sections as needed.

The last deviation from our original plan has to do with the server side application. Purpose wise, this layer is unchanged, however our team has decided that this part of the library is not essential for development purposes and will be the lowest priority for the team.

Finally, we decided to de-prioritise the server side application since it is not essential to the functionality of our project.

5.2.2 Timeline Changes

Our team, since the completion of the HAL, has moved our meetings from in person to online. This was done because at this stage of our project our tasks have been divided for the driver layer and our work has become more independent. This has allowed us to be more flexible in our meeting times and require less meetings overall. We currently meet about once a week online to give updates to one another and discuss major design decisions.We use direct messaging to communicate any minor questions and updates that arise as we work. This is a change from our teams initial plan for meetings as we previously met every Tuesday and Thursday for three hours to collaborate on the code up to the HAL. Our current meeting setup will remain this way until the next semester. However, it is likely to revert back to the original in person meetings at scheduled times next semester as we bring the split layers together again. We have also periodically attended project meetings with our advisor Professor MD Shaad Mahmud and will continue to do so.

Our team has also updated our timeline in our new Gantt chart we created[11]. This new timeline created is updated to the status of our project thus far. Our team has not yet completed the full implementation stage of our library in the initial timeline[12]. We also began the documentation earlier than planned overlapping our implementation stage. Due to these differences in the plan, we have extended out the time frame for implementation into the next semester, and changed the documentation timeline. The reason for changing the documentation timeline is because our team has concluded that we would be better off documenting as we code rather than once it is all done, and therefore the timeline has

[10]LoRa Library GitHub Code Link A
[11]Updated Current Gantt Chart C.2.1
[12]Initial Gantt Chart C.1.1

been greatly increased. Due to the extension of our implementation stage, we have extended our testing timeline as well as it now overlaps implementation. We intend to at least get the project to a testable stage as soon as possible over January break so we are deciding to keep the testing timeline overlapping implementation.

5.3 Tasks Breakdown

Both team members collectively worked on the overall architecture of the library, as well as making decisions regarding code conventions, documentation, and version control systems. Both members also worked together on proposal and report papers using the online LaTeX editor Overleaf. Josh Lariviere handled much of the advanced formatting features available in LaTeX, the implementation of the hardware and hardware abstraction layer portions of the software, and the generalized templates for sensor drivers using abstract C++ classes. Andrew Courtemanche is responsible for the design and implementation of the mesh protocol itself, the drivers for interacting with the LoRa and GPS radios, and improved automatic build tools using GNU make. The vast majority of the work has been done collaboratively via in-person meetings multiple days a week. It has been a noticeable advantage having only two members on the team, since it eliminates a lot of the overhead of coordinating a large team.

6 Future Work

6.1 Core Mesh Implementation

While many of the core components of the mesh, LoRa, and GPS have been designed and have skeleton code written, most of the actual functions have yet to be implemented. This is arguably the highest priority task, as much of our other code and testing relies on this portion of the project's completion.

6.2 Driver Implementations

In addition to the core components of the project, many of our driver implementations have yet to be written. This includes components such as temperature sensor, humidity sensor, SD card support, etc. While not the focus of the project, they are critical in practical testing of the mesh as a whole.

6.3 Documentation

6.3.1 Doxygen

To make the code base easier to maintain, all of our source files will be enhanced with the automatic documentation generator Doxygen [4]. Doxygen generates HTML documentation based on comments added directly to the source files.

This will make future improvements to the software much easier, as well as providing library users a starting point for integrating with their own projects.

6.4 Testing

6.4.1 Application

The application layer depends on the completion of all of the other layers of the project, and is primarily going to be used to test and validate the functionality of the software stack[13]. Our goal with this project is not to write a particular application, but rather facilitate the creation of useful applications that depend on the networking capabilities of the mesh library itself.

We would additionally use this application to conduct field tests of the library running on the IBUG node[14] to further validate functionality.

7 Discussion & Conclusion

Overall our team encountered very few technical setbacks, but lost a lot of time to factors unrelated to the project itself, including significant college course load. Furthermore, due to the current global situation there has been a significant integrated circuit shortage and Printed Circuit Board (PCB) fabrication lead times have increased dramatically. This has resulted in an inability to make any significant changes to the IBUG node hardware[15]. The increased hardware costs have limited our ability to obtain additional IBUG node hardware beyond what was already provided. This will limit large scale testing of our library. Without the freedom to make hardware changes, the team was forced to test LoRa communications using an Arduino MKR WAN 1300 [6]. As a result, we have entirely shifted our focus to our software.

Our approach so far has involved a lot of organization and boilerplate[16] to give us a good foundation to build off of. This has consumed a lot of our time, leaving us with a little less progress than we would have liked. Despite that, it should result in significant time savings in the future, and we plan to spend winter break getting our implementation to a point where we can begin testing.

[13]A software stack is a collection of independent components that work together to support the execution of an application.[5]

[14]IBUG Node Overview B

[15]IBUG Node Hardware B.2

[16]In computer programming, boilerplate code—or simply, boilerplate—are sections of code that are repeated in multiple places with little to no variation [7].

Acknowledgment

- University of New Hampshire College of Engineering and Physical Sciences [8]
- National Science Foundation Grant #1935578 [9]

Figure 2: National Science Foundation

References

[1] W. Foundation, *Chirp Spred Spectrum Definition*, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Chirp_spread_spectrum.

[2] T. Joachim, *Complete Reverse Engineering of LoRa PHY*, 2021. [Online]. Available: https://www.epfl.ch/labs/tcl/wp-content/uploads/2020/02/Reverse_Eng_Report.pdf.

[3] Semtech, *What are LoRa and LoRaWAN?* 2021. [Online]. Available: https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan.

[4] Doxygen, *Doxygen Code Documentation Generator*, 2021. [Online]. Available: https://www.doxygen.nl/index.html.

[5] TechTarget, *Software Stack Definition*, 2021. [Online]. Available: https://searchapparchitecture.techtarget.com/definition/software-stack.

[6] Arduino CC, *MKRWAN 1300 and Antenna Pricing*, 2021. [Online]. Available: https://store-usa.arduino.cc/products/arduino-mkr-wan-1300-lora-connectivity.

[7] W. Foundation, *Boilerplate code*, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Boilerplate_code.

[8] University of New Hampshire, *UNH College of Engineering and Physical Sciences*, 2021. [Online]. Available: https://ceps.unh.edu/.

[9] National Science Foundation, *National Science Foundation Grant*, 2021. [Online]. Available: https://www.nsf.gov/.

[10] RAK, *RAK4600 Documentation*, 2021. [Online]. Available: https://docs.rakwireless.com/Product-Categories/WisDuo/RAK4600-Module/Datasheet/.

[11] A. I. LLC, *MTK3339 Datasheet*, 2021. [Online]. Available: https://cdn-shop.adafruit.com/product-files/746/CD+PA1616S+Datasheet.v03.pdf.

[12] B. Sensortec, *BME688 Documentation*, 2021. [Online]. Available: https://www.bosch-sensortec.com/products/environmental-sensors/gas-sensors/bme688/.

[13] A. I. LLC, *LIS3DH Datasheet*, 2021. [Online]. Available: https://cdn-learn.adafruit.com/assets/assets/000/085/846/original/lis3dh.pdf?1576396666.

[14] M. C. AG, *RV-8803-C7 Datasheet*, 2021. [Online]. Available: https://www.microcrystal.com/fileadmin/Media/Products/RTC/Datasheet/RV-8803-C7.pdf.

[15] A. Osram, *CCS811 Datasheet*, 2021. [Online]. Available: https://cdn.sparkfun.com/assets/learn_tutorials/1/4/3/CCS811_Datasheet-DS000459.pdf.

[16] JLCPCB, *JLCPCB.com Pricing*, 2021. [Online]. Available: https://jlcpcb.com/.

[17] PCBWay, *PCBWay.com Pricing*, 2021. [Online]. Available: https://www.pcbway.com/orderonline.aspx.

[18] Voltera, *Voltera PCB Printer*, 2021. [Online]. Available: https://www.voltera.io/.

[19] Vultr, *Vultr Cloud Compute pricing*, 2021. [Online]. Available: https://www.vultr.com/products/cloud-compute/#pricing.

[20] Linode, *Linode Shared CPU pricing*, 2021. [Online]. Available: https://www.linode.com/pricing/#compute-shared.

[21] Amazon, *Amazon web services elastic compute cloud pricing*, 2021. [Online]. Available: https://aws.amazon.com/ec2/pricing/on-demand/.

[22] Octopart, *Octopart Bill Of Material Creator*, 2021. [Online]. Available: https://octopart.com/.

[23] D. L. Mai and M. K. Kim, "Multi-Hop LoRa Network Protocol with Minimized Latency," 2020. [Online]. Available: https://mdpi-res.com/d_attachment/energies/energies-13-01368/article_deploy/energies-13-01368.pdf.

[24] G. Boquet, P. Tuset-Peiro, F. Adelantado, T. Watteyne, and X. Vilajosana, *LR-FHSS: Overview and Performance Analysis*, 2010. [Online]. Available: https://arxiv.org/pdf/2010.00491.pdf.

Appendices

A LoRa Library GitHub Code Link

This link will direct you to the latest main branch of the library GitHub repository: https://github.com/arc968/LoRaMeshCapstone

This will additionally give access to the documentation for the library created through the program Doxygen [4]. Use the README file as a guide for navigating the GitHub repository and how to access the Doxygen documentation.

B IBUG Node Overview

The IBUG node is a PCB designed to utilize many data collecting sensors. These nodes are able to be placed in fields and collect data on any area they are in. For our team's project the library[17] will be used to create an application described in the Overview section[18] that will be able to run on the IBUG node. This application will allow the IBUG node to connect to the mesh network of all the IBUG nodes in an area as well as read its sensor's data and transmit it over the network to the central gateway to pass to the user.

The IBUG node also has an optional SD card slot for the users. This SD card slot will allow the user to have a memory card in the node for a method to back up any data the node gathers that is gathered and sent over the mesh network. This back up of the data is used to allow the node to store any data in the event the node cannot successfully transmit the data over the network to ensure the user is protected against data loss.

[17]LoRa Library GitHub Code Link A

[18]Overview 4

B.1 IBUG Hardware Images

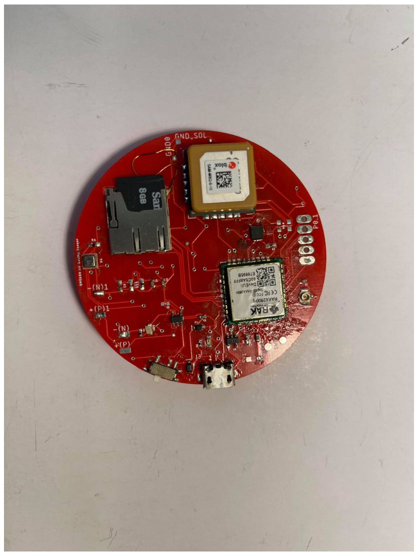


Figure 3: Top Side of IBUG Node Hardware

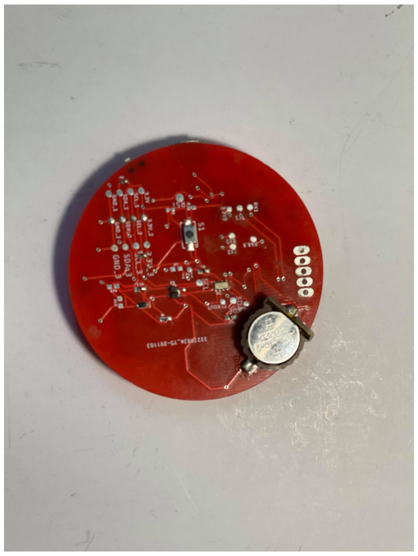


Figure 4: Bottom Side of IBUG Node Hardware

B.2 IBUG Node Hardware

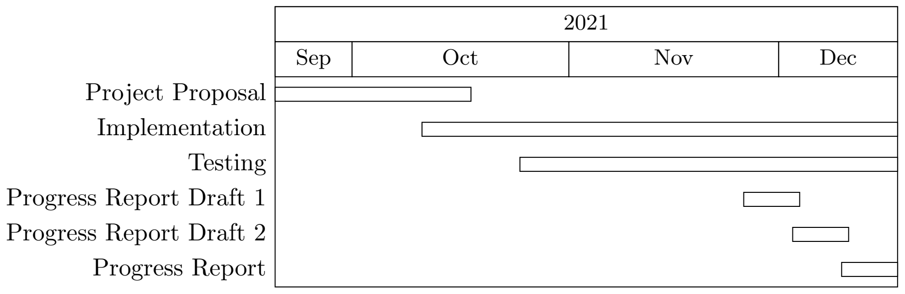
The library application is being written for the IBUG node hardware for testing purposes. The hardware being used in the IBUG node being used is its version 1.1, the hardware list is as follows:

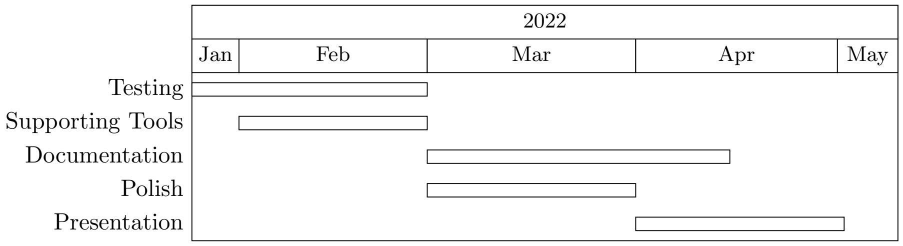
- Mesh components:
  - RAK4600 [10]: Processor
    - Datasheet: RAK4600_Datasheet.pdf
  - MTK3339 [11]: GPS
    - Datasheet: MTK3339_Datasheet.pdf
- Sensors:
  - BME688 [12]: Gas Sensor
    - Datasheet: BME688_Datasheet.pdf
  - LIS3DH [13]: Accelerometer
    - Datasheet: LIS3DH_Datasheet
  - RV-8803-C7 [14]: Real Time Clock (RTC)
    - Datasheet: RV8803C7_Datasheet.pdf
  - CCS811 [15]: Air Quality Sensor
    - Datasheet: CCS811_Datasheet.pdf

C Project Timeline

C.1 Initial Intended Project Timeline

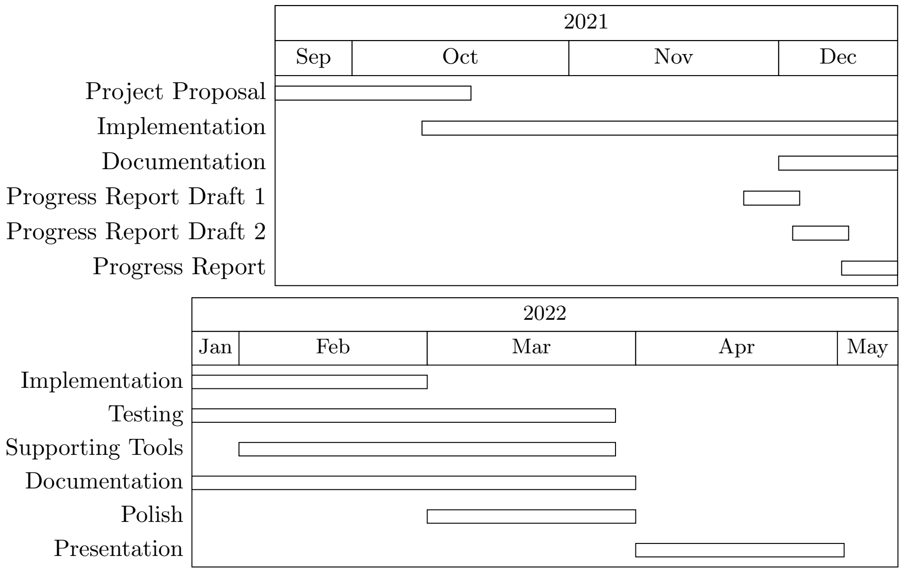
C.1.1 Initial Gantt Chart





C.2 Current Project Timeline

C.2.1 Updated Current Gantt Chart



D Project Budget Data

D.1 Anticipated Initial Budget Breakdown

Our team is comprised of two students for which the school allocates $100 per team member giving our project $200. Our faculty advisor has also generously agreed to match our $200 budget to give us a total use-able budget of $400. The costs associated with our project breaks down into three sections as follows:

- Node production expenses

- Testing setup
- Web services

D.1.1 Node Production Expenses

The cost of producing a node is comprised of two parts. The fist part is the cost to have a PCB fabricated and assembled with all its components soldered on fully assembled, on delivery. This cost can vary greatly depending on what company is producing the PCB. The two companies the team has found that we believe are the best choices for this are called JLCPCB [16] and PCBWay [17]. These companies offer the lowest prices for small batches of PCBs along with the best deals on assembly for the boards. The price breakdown for the companies are:

- JLCPCB [16]
  - PROS:
    - Offers $2 fabrication for five 2-4 layer PCBs under 100mmx100mm in size using the standard materials or $5 fabrication for five 4-6 layer PCBs of the same requirements.
    - Fast 1-3 day production.
    - Offers $7 for PCB assembly base cost plus PCB Bill of Materials (BOM) and special assembly fees.
  - CONS:
    - Non free shipping plus long shipping times overseas.
    - Assembly is limited to JLCPCB's own parts supply.
    - No part sourcing for assembly.
- PCBWay [17]
  - PROS:
    - Offers $5 fabrication of 10 1-2 layer PCBs under 100mmx100mm in size using the standard materials.
    - Fast 24 hour production.
    - Offer $30 assembly for 20 PCBs plus BOM cost and special assembly fees.
    - They have part sourcing and allow use of any component.
    - Allow for fabrication of boards then choosing how many of them are assembled.
  - CONS:
    - Non free shipping plus long shipping times overseas.
    - No deals for more than 2 layer PCBs.

We estimate the total cost for all the components for a single node to be around $80.

This shows the total cost for creating a single node using the two fabrication companies:

- JLCPCB [16]:
  - Minimum fabrication amount: 5
  - Total fabrication cost: $2
  - Minimum assembly amount: 2
  - Total assembly cost: $167
  - Minimum shipping cost: $2.99 for 12-20 business days
  - Maximum shipping cost: $12.98 for 2-4 business days
- PCBWay [17]:
  - Minimum fabrication amount: 10
  - Total fabrication cost: $5
  - Minimum assembly amount: 1
  - Total assembly cost: $110
  - Minimum shipping cost: $10 for 6-16 business days
  - Maximum shipping cost: $20 for 2-4 business days

Total Estimated Cost:

JLCPCB: Minimum: $171.99 Maximum: $181.98
PCBWay: Minimum: $125 Maximum: $135

D.1.2 Testing Setup

For this section of the budget, our team already has been provided hardware to test with. However, if we find that we require additional hardware, or if something breaks in testing, it will add to our costs. We need to ensure that we have enough budget to handle unexpected testing costs that may arise.

This cost could include replacing broken hardware in our testing setup that may occur. In our setup, we are utilizing MKR WAN 1300 Arduino [6] and antenna. The cost of the Arduino and included antenna is $40.30 dollars per unit.

Other unknown costs for testing could include:

- A GPS IC chip
- Node cases
- Wires
- Voltera PCB prototyping [18]

D.1.3 Web Services

For our project, our team needs to be able to collect the data from our mesh network and store it. We need to use a web server that allows us to store, manipulate and analyze our data. Our team has found three possible options of Vultr [19], Linode [20] and Amazon Web Services (AWS) [21].

For our project, in the testing stages, our team only needs a bare minimum server to work with we believe. Looking into all three server options our team picked out the server type we would have to purchase from each. Additionally, using our teams initial Gantt chart[19] we would need this server for around seven months or more bringing the costs to:
The server cost break down is as follows:

- Vultr [19]
  - $3.50 per month for 10GB Storage, 1 CPU, 512MB Memory, 0.5TB Bandwidth.
  - $24.50 for seven months.
- Linode [20]
  - $5 per month for 25GB Storage, 1 CPU, 1GB Memory, 1TB Bandwidth.
  - $35 for seven months.
- Amazon Web Services (AWS) [21]
  - t4g.nano for $0.0042 per hour for EBS Storage, 2 CPU, 0.5GB Memory, 5Gbit Bandwidth.
  - About $22 for seven months.

D.1.4 Total Budgeting Breakdown

For our total budget:

- Node production cost can range between $125 to $182
- Server Costs can range between $22 to $35

Our current budget is $400 total. The cost for a server is low however the cost per prototype node is very high. For our project ultimately our team will need many nodes produced for full scale testing. This cost of enough nodes to test with will significantly increase our expenses and leave little to no funds left should any additional costs arise during testing.

[19]Initial Gantt Chart C.1.1

D.2 Current Budget Status

Currently, our team has not needed to tap into our allotted budget as our progress has been entirely code based. We have been lucky enough that we have had no unforeseen testing hardware costs. Additionally due to the current chip shortage and supply chain issues, our team has not made any hardware purchases or changes for the IBUG node[20]. This is because the costs have been to high and the lead time for fabrication and parts makes hardware changes at this time not a efficient use of time.

D.3 Future Budget Expectations and Plan

For the future of our teams budget, we do still expect the costs of purchasing a server for testing. This cost will be directly associated to our teams new testing timeline[21]. We will be purchasing the use of a Vultur basic server [19][22]. This server cost will begin when the team has finished the library code and is ready for field testing of the intended application running on the IBUG node. This is the only foreseeable expected cost our team anticipates at this time and will not make a significant impact into our $400 budget.

[20]IBUG Node Overview B
[21]Updated Current Gantt Chart C.2.1
[22]Web Services D.1.3

Acronyms

A | **B** | **G** | **H** | **I** | **O** | **P** | **R** | **S** | **W**

A

AWS Amazon Web Services. 21, 23

B

BOM Bill of Materials. 19, 23

G

GPS Global Positioning System. 5, 10, 17, 20, 23

H

HAL Hardware Abstraction Layer. 4, 7, 8, 9, 23

HTML Hyper Text Markup Language. 10, 23

I

IC Integrated Circuit. 20, 23

IDE Integrated Development Environment. 4, 8, 23

ISM Industrial Scientific Medical. 3, 23

O

OSI Open Systems Interconnection. 3, 23

P

PCB Printed Circuit Board. 11, 19, 20, 23

PHY LoRa Physical Layer. 3, 23

R

RTC Real Time Clock. 17, 23

S

SD Secure Digital. 10, 15, 23

W

WAN Wide Area Network. 3, 23