

IMPROVING SUSTAINABILITY IN AGRICULTURE USING WIRELESS SENSOR NETWORKS

By

Jonathan Gresl

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF ARTS

in

Irving K. Barber School of Arts and Sciences
(Computer Science)

Supervisor: Dr. Ramon Lawrence

THE UNIVERSITY OF BRITISH COLUMBIA
(Okanagan)

September 2020

© Jonathan Gresl, 2020

The following individuals certify that they have read, and recommend to the Irving K. Barber School of Arts and Science for acceptance, a thesis entitled:

Improving Sustainability in Agriculture Using Wireless Sensor Networks

submitted by Jonathan Gresl in partial fulfillment of the requirements for

the degree of Bachelor of Arts

in Computer Science

Examining Committee:

Ramon Lawrence, Computer Science

Supervisor

Scott Fazackerley, Computer Science

Additional Examiner

Abstract

Precision agriculture is an approach to farming that uses real-time data to manage environmental variations, measure performance to improve upon passed seasons, and perform predictive analysis to make better farming decisions, which result in higher production yields with lower production costs. Although beneficial, collecting and analyzing the environmental data is an expensive and complicated endeavor. Wireless sensor networks can be setup in fields to sample and collect data, but many parts are needed. This leads to high overall equipment costs, complex implementations, and wireless network congestion. To further disadvantage the use of wireless sensor networks in agriculture, modern network protocols such as WiFi or Bluetooth provide limited transmission ranges with high power consumption. As wireless sensor networks require batteries to power the sensor nodes, high power consumption results in the frequent need to replace the batteries. These limitations discourage farmers from adopting sensor network technologies for their farms because the resources required to implement and maintain them can be too high to justify the investment. This research project aims to develop a wireless sensor network that uses inexpensive and effective hardware, that is easily setup and maintained. Costs to implement the network are reduced through the use of inexpensive open-source hardware. Transmission ranges are increased by using long range (LoRa) radio transceivers that operate on license-free radio frequency bands. Power consumption is also reduced by using the LoRa radio transceivers and is further reduced through strategic data and power management. By addressing these limitations, farmers will be more likely to adopt the technology on their farms, which ultimately improves sustainability in agriculture.

Lay Summary

The main goal of this research project is to develop a wireless sensor network that is optimized for sensor data collection in large open areas – particularly agricultural fields. To be practical for farmers, the network must be easy and inexpensive to implement. The network should require little user intervention from the farmers. My contributions to this project include programming the network devices, optimizing them for continual usage, and developing a website that enables users to configure the network components and analyze the collected data.

Preface

This thesis is based on research conducted at UBC Okanagan's database laboratory supervised by Dr. Ramon Lawrence and Dr. Scott Fazackerley.

I was responsible for writing code for the sensor nodes, central gateway, and website. I also was involved in testing the device functionality and collection of sample sensor data.

The queue used in this project to store sample data on the sensor nodes was created by Ethan Godden, who has also contributed a thesis related to this research project.

Table of Contents

Abstract	iii
Lay Summary	iv
Preface	v
Table of Contents	vi
List of Figures	viii
Acknowledgements	ix
Chapter 1: Introduction	1
1.1 Precision Agriculture	1
1.2 Sensor Networks in Agriculture.....	1
1.3 Wireless Sensor Networks in Agriculture	2
1.4 Limitations	3
1.5 Motivation	3
Chapter 2: Background	5
2.1 Microcontrollers	5
2.2 Single-board Computers.....	5
2.3 Spread Spectrum Modulation	6
2.4 LoRa Radio	6
2.4 License-free Radio Spectrum.....	8
Chapter 3: Implementation	10
3.1 Network Architecture	10
3.1.1 Sensor Node Parts	10
3.1.2 Central Gateway Parts.....	13
3.2 Data Flow	15
3.3 Database	16
Chapter 4: Deployment	19
4.1 Code Location	19
4.2 Deploy the Central Gateway	19
4.2.1 Server Dependencies.....	19
4.2.2 Server Code Execution.....	20
4.3 Deploy the User Interface	21
4.3.1 Website Dependencies.....	22
4.3.2 Website Code Execution.....	22
4.4 Deploy the Sensor Nodes	25
4.4.1 Client Dependencies.....	25
4.4.2 Client Code Execution.....	26

Chapter 5: Walkthrough	27
5.1 Access the User Interface Internally.....	27
5.2 Access the User Interface Externally	27
5.3 Manage the Network Setup	28
5.4 View Sample List	31
5.5 View Line Charts	32
5.6 Network Reset	34
Chapter 6: Future Work	35
6.1 Decreasing Power Consumption	35
6.2 Data Persistence	35
6.3 Payload Optimization	37
6.4 Agricultural Implementation	37
6.5 Equipment Protection	38
Chapter 7: Conclusion	39
Bibliography	40

List of Figures

Figure 1: Wireless sensor network deployed on an agricultural field [2]	2
Figure 2: Example layout of a central gateway and sensor nodes in a vineyard	4
Figure 3: Frequency domains of a narrowband signal and a spread-spectrum signal	6
Figure 4: Chart comparing bandwidth and range of LoRa, Wi-Fi, and cellular [5].....	7
Figure 5: Illustration of radio waves in the electromagnetic spectrum [7]	8
Figure 6: Table containing regional frequency plans and specifications [9].....	9
Figure 7: Sensor node microcontroller – Adafruit Feather M0 [10]	10
Figure 8: Sensor node radio transceiver – RF95 LoRa Radio [11]	11
Figure 9: Sensor node using a micro USB, wire antenna, and 3.7V LiPo battery [11]	12
Figure 10: Algorithm to determine battery voltage [14]	12
Figure 11: Discharge profile of a 3.7V LiPo battery [13]	13
Figure 12: Raspberry Pi 3 Model B+ [15].....	14
Figure 13: Dragino LoRa hat for the Raspberry Pi [16].....	14
Figure 14: Data flow diagram of the sensor nodes, central gateway, and server	15
Figure 15: Entity-Relationship diagram	17
Figure 16: Database data definition language (DDL)	18
Figure 17: Example of port forwarding to enable external access to the gateway	24
Figure 18: Installing the Visual Studio Code extension for C/C++.....	25
Figure 19: Installing the Visual Studio Code extension for PlatformIO.....	25
Figure 20: Network map using the Google maps API.....	28
Figure 21: Gateway and node configuration tables.....	29
Figure 22: List of samples with filter options	31
Figure 23: Time series line chart with outside node temperature data	32
Figure 24: Time series line chart with outside node battery data	33
Figure 25: Database restore page to remove all samples and synced nodes.....	34
Figure 26: Adafruit Feather M0 add-on with RTC clock and SD card slot [20]	36

Acknowledgements

I wish to express my sincere appreciation to my supervisor, Dr. Ramon Lawrence, who guided and supported me every step of the way through both the research project and my courses at UBC. By accepting me as an honours student and a contributor to his research, my educational experience has been immensely enriched. I also wish to express gratitude to Dr. Scott Fazackerley for reviewing my contributions to the research and making suggestions for improvement. Scott has been an excellent source of knowledge on hardware selection, programming concepts, and network protocols. Finally, I want to thank my fiancé Kim and my newborn son Jaxon for their continual support and patience during my studies at university and dedication to this project.

Chapter 1: Introduction

1.1 Precision Agriculture

Precision agriculture is an approach to farming that uses information to ensure crops can grow in optimal conditions. Environmental variations can result from climatic conditions, soil composition, cropping practices, weeds, or diseases. Monitoring key point indicators allow farmers to track their crop's status which can help them determine if crops are suffering from water stress, nitrogen stress, or if diseases are developing [1]. Environmental variations result in loss of crops and inconsistent growth. When conditions vary, adjustments can be made to regular farming activities to control or stimulate growth in plants. The collected data is also useful for measuring performance and improving upon passed seasons. Using precision agriculture reduces production costs by mitigating waste. Ultimately, precise farmers can grow more food with fewer resources.

1.2 Sensor Networks in Agriculture

Sensor networks are used in agriculture today. They are setup in fields to collect real-time environmental data. Originally, they were connected using wires, which would power the components and facilitate data transfer. While wired connections provide stability, they are also extremely inconvenient because large amounts of wire are needed to connect the sensor nodes. The wire must also be concealed in a manner that doesn't limit accessibility and is protected from machinery. Damaged wire will disable some or all of the sensor network and will require user intervention to troubleshoot and restore functionality.

1.3 Wireless Sensor Networks in Agriculture

Wireless sensor networks were introduced to improve practicality. The wireless components are powered by batteries and connected through a wireless medium – typically radio waves. Figure 1 depicts a typical wireless network deployed on an agricultural field. The sensor nodes are deployed strategically across a field within range of the central gateway. They communicate over ISM radio frequency bands with each other or directly with the central gateway. The gateway is connected to the Internet to allow remote users to monitor the field's state and control the sensor nodes. A database stores the sensor data and is accessible through the network for configuration and sensor data analysis [2].

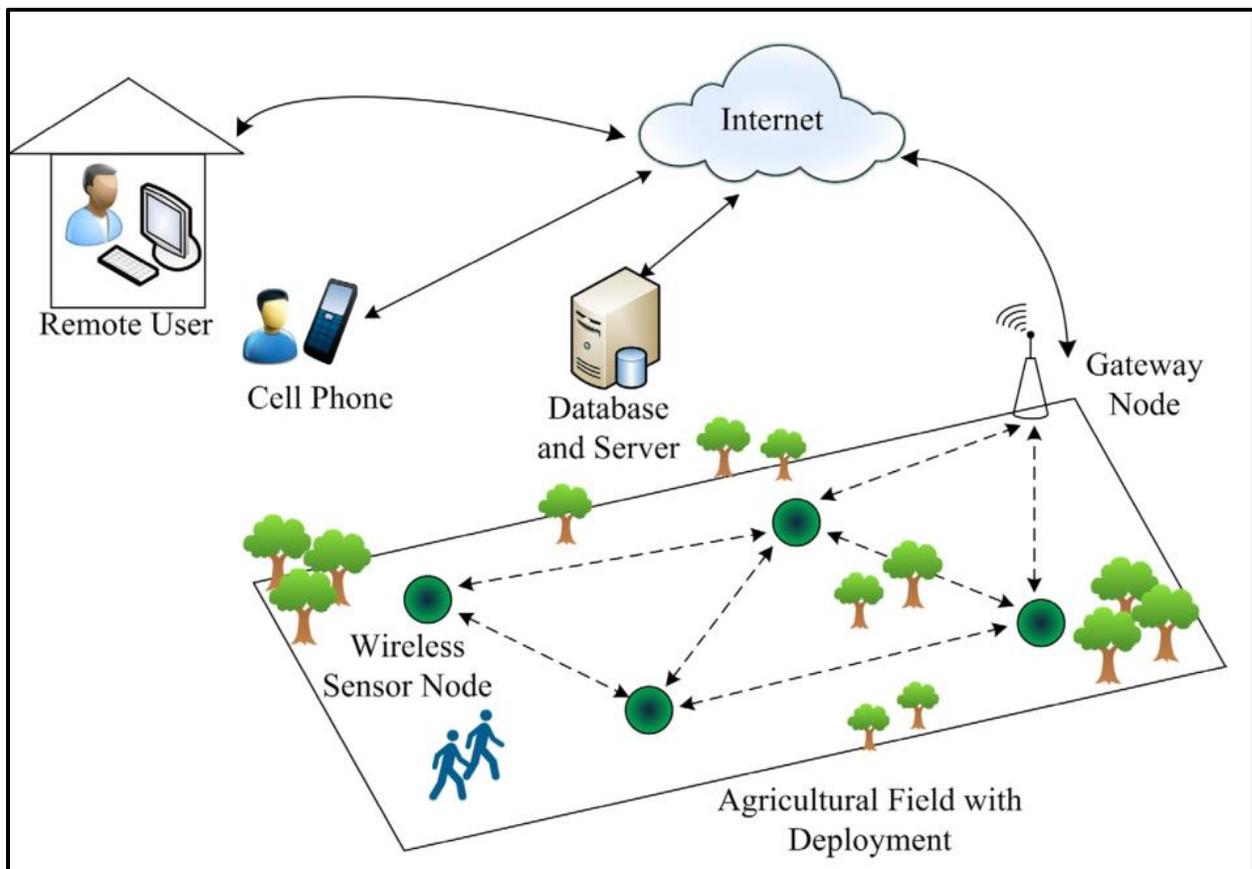


Figure 1: Wireless sensor network deployed on an agricultural field [2]

1.4 Limitations

Although wireless sensor networks are more practical than wired networks, they also come with their own set of challenges and limitations. The hardware required to setup a network large enough to cover large rural areas is expensive. There is also a large amount of effort required to setup or modify the network. WiFi or Bluetooth protocols can be used for the data transfer, but these are complicated technologies that provide limited transmission range with high power consumption. Although data can be chained through sensors to increase the transmission range, this solution requires more transmissions, which translates to more power consumption and higher amounts of radio traffic. Further, there is a lot of overhead involved in WiFi and Bluetooth protocols that are unnecessary for applications in agriculture. In summary, the major limitations are high costs, complexity, and power consumption. High costs will discourage farmers from wanting to invest in the product, especially those with larger farms. High complexity will make it difficult to implement initially or to expand the network later. Finally, and most importantly, radios with high power requirements will require farmers to replace sensor node batteries too often for the farmer to want the system.

1.5 Motivation

Agricultural industries will benefit from using wireless sensor networks, but farmers will be hesitant to invest in the technology if it is not practical for their application. An ideal agricultural product is one similar to smart home technologies with light bulbs. Farmers should be able to purchase a central gateway that sensor nodes can communicate with. New sensor nodes should easily synchronize with the central gateway and be configurable through it. The gateway should also be connected to the Internet to provide remote configuration and data analysis. Above all, farmers should rarely need to replace sensor nodes – or their batteries.

For wireless sensor networks to be sustainable in agriculture, they must be affordable, require little user intervention, stay powered for long periods of time, stay protected from the elements of nature, be easy to implement and modify, and transmit at long ranges while handling radio interference. This is a long list of requirements, but they can all be achieved inexpensively with the emergence of open-source hardware, open-source software, and 3D printing technology.

In general, improving transmission distances and reducing power requirements will make wireless sensor networks more practical and affordable for the farmer. Another consideration is that agricultural industries yielding high-value crops will benefit more from implementing the system than agricultural industries yielding low-value crops. Fruit products, for example, are more valuable than many other agricultural products and are often grown in smaller rural areas. The main motivation of this research is to develop a wireless sensor network that can be implemented in a local vineyard.



Figure 2: Example layout of a central gateway and sensor nodes in a vineyard

Chapter 2: Background

2.1 Microcontrollers

A microcontroller is a small computer that is designed to execute a single program. They use the same elements as traditional computers but host them on a single integrated circuit board. Typically, they are fitted with a micro USB port used for transferring data or powering the device and some have a separate port for batteries. Because of the condensed architecture, they are more portable, cost less, consume less power, and can be optimized for a given application [3]. Microcontrollers also typically have general input output pins (GPIO), which can be configured to read sensor data or drive external devices such as LED's or radio transceivers. These advantages make microcontrollers a suitable computer for the network's sensor node components.

2.2 Single-board Computers

Single-board computers are similar to microcontrollers in that they are small computers built on a single circuit board. While they share many of the same advantages, single-board computers are more powerful and can often run operating systems. Many of them come with HDMI ports, USB ports, and LAN ports, WiFi modules, Bluetooth modules, and microSD card support. As a result, single-board computers have higher power requirements than microcontrollers, but still much less than servers or personal computers. As low costs are important for this project, the reasonable purchase price and power requirements of single-board computers make them suitable computers for the network's central gateway component. As the gateway must always be running – and may be setup indoors without too much obstruction – the fact that the devices create little heat and noise is another advantage of using them for wireless sensor networks where the central gateway is setup inside a farmhouse or structure.

2.3 Spread Spectrum Modulation

Spread spectrum modulation deliberately spreads radio frequency (RF) signals within the frequency domain to create a signal with a wider bandwidth. This technique increases resistance to other transmitters operating at the same frequencies, allowing for more robust RF communications over long ranges using a much lower average amplitude [4]. Figure 3 shows an original narrowband signal that covers a narrow range of frequencies and a modified signal which has been spread to make the bandwidth much wider.

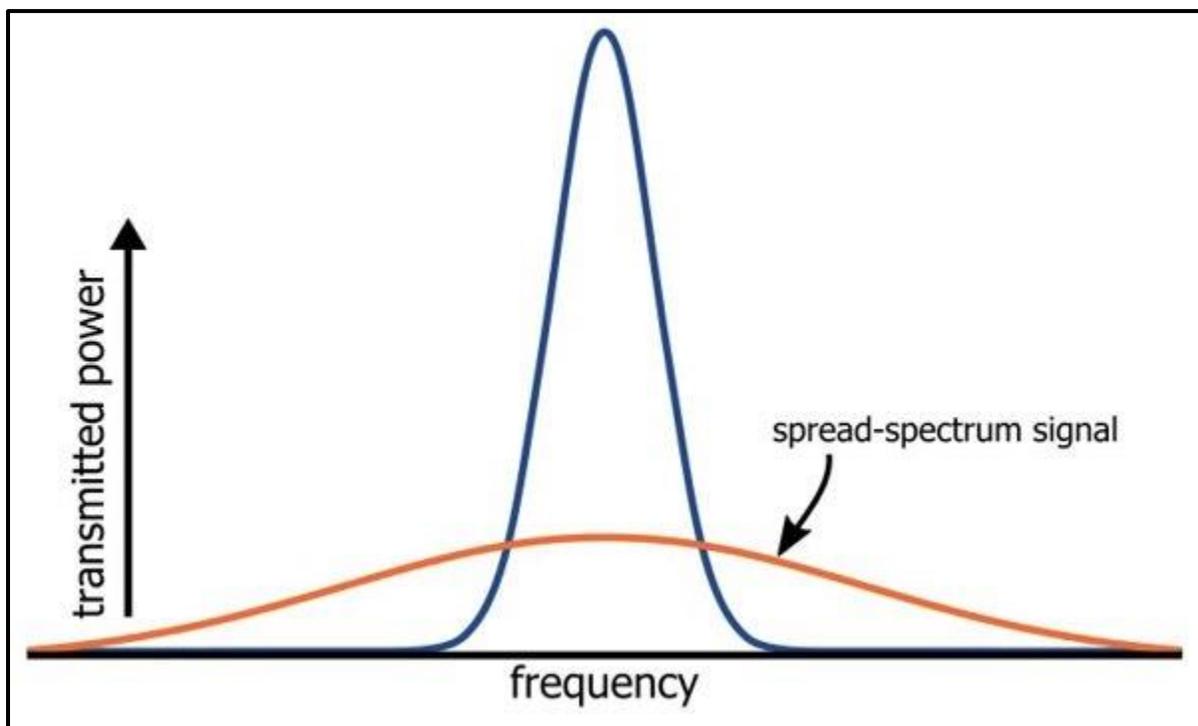


Figure 3: Frequency domains of a narrowband signal and a spread-spectrum signal [4]

2.4 LoRa Radio

LoRa, short for long range, is a radio wave modulation technology developed by Cycleo in 2009, which was later acquired by Semtech in 2012. The technology uses a proprietary spread spectrum modulation technique which enables data communication over long ranges while using little power, making it a flexible solution for rural use cases in smart

agriculture [5]. Figure 4 illustrates how WiFi and Bluetooth Low Energy (BLE) communicate on higher frequencies with a shorter range. With cellular, high frequencies are still used but longer ranges are achieved through expensive towers and increased power output. Because LoRa uses lower frequencies, the wavelengths are longer, which result in longer ranges but with lower data rates. This makes LoRa most suitable for implementations that do not require large amounts of data transferred over short periods of time. Basically, LoRa would not be a good choice for streaming video but an excellent choice for periodically transmitting packets of sensor data.

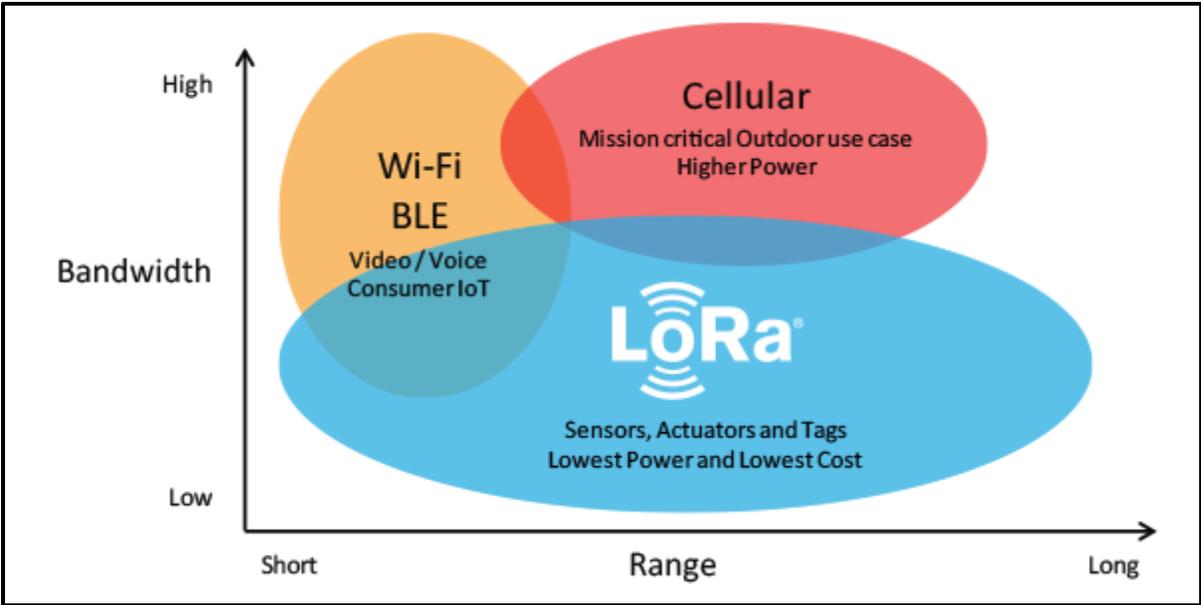


Figure 4: Chart comparing bandwidth and range of LoRa, Wi-Fi, and cellular [5]

An important thing to note is that LoRa is different from LoRaWAN. The patented LoRa wireless radio frequency technology stands for the physical layer protocol while LoRaWAN, which is officially developed by the LoRa Alliance, stands for the media access control (MAC) layer protocol, leveraging and including the physical LoRa modulation of Semtech [6]. In short, LoRa itself is the technology used for sending and receiving signals and LoRaWAN is the software put on the chip to enable networking.

2.4 License-free Radio Spectrum

Radio waves are a type of electromagnetic energy that make up a small part of the electromagnetic spectrum. They are used in communication technologies to transmit data from one place to another. Most modern data communication occurs between the 300 megahertz (MHz) and 6 gigahertz (GHz) frequencies.

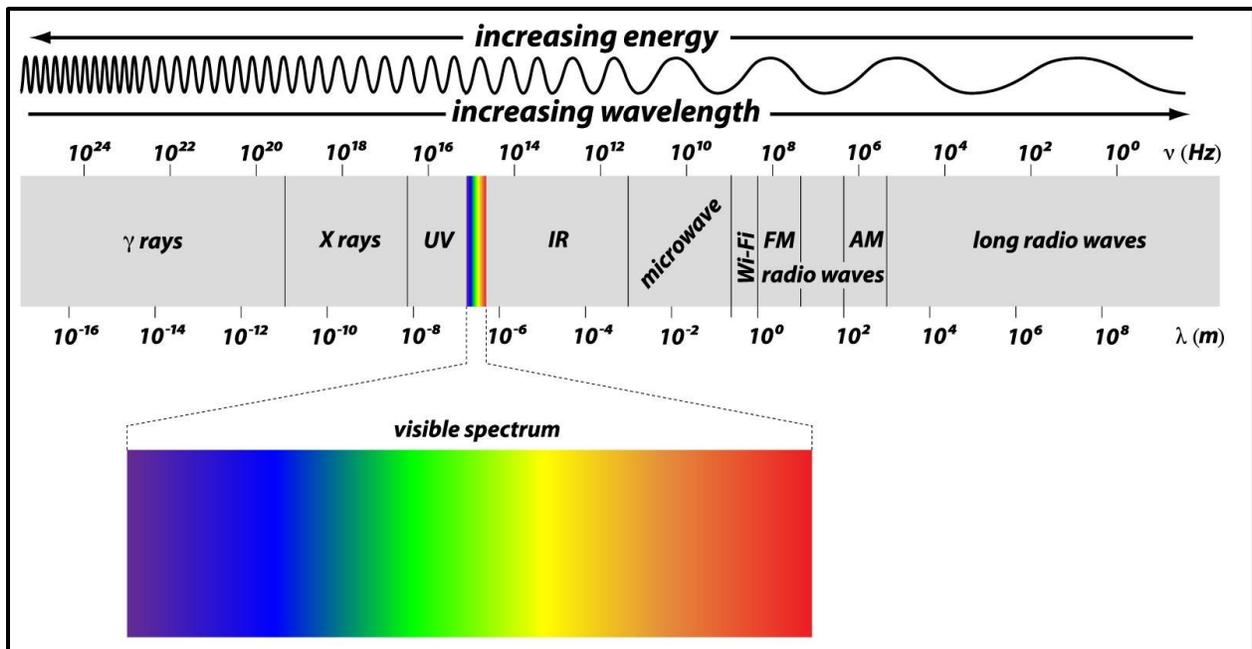


Figure 5: Illustration of radio waves in the electromagnetic spectrum [7]

As the electromagnetic spectrum is a limited resource, many countries allocate specific frequency ranges for different uses. Much of the spectrum is licensed to organizations who pay a licensing fee that gives them exclusive rights to transmit on assigned frequencies and assurance that nobody else will use them [7]. There are often small parts of the spectrum which are not licensed and are available for anyone to transmit on. The advantage of using license-free radio frequency bands is that permission is not needed – they are free to use; the disadvantage is that there is high potential for interference from other devices transmitting on the same or similar frequencies.

Although there are often multiple license-free bands, most long range protocols operate in the sub-gigahertz license-free bands, the most prominent of which are a large contiguous band from 902-928 MHz and narrower bands at 868 MHz, 470 MHz, 433 MHz, and 315 MHz [8]. This is because in open rural areas, these bands have lower path loss than higher frequency bands. LoRa uses the unlicensed radio spectrum in the Industrial, Scientific and Medical (ISM) bands [6]. LoRa radios should be configured to comply with regional frequency plans and specifications.

	Europe	North America	China	Korea	Japan	India
Frequency band	867-869MHz	902-928MHz	470-510MHz	920-925MHz	920-925MHz	865-867MHz
Channels	10	64 + 8 + 8	In definition by Technical Committee			
Channel BW Up	125/250kHz	125/500kHz				
Channel BW Dn	125kHz	500kHz				
TX Power Up	+14dBm	+20dBm typ (+30dBm allowed)				
TX Power Dn	+14dBm	+27dBm				
SF Up	7-12	7-10				
Data rate	250bps- 50kbps	980bps-21.9kbps				
Link Budget Up	155dB	154dB				
Link Budget Dn	155dB	157dB				

Figure 6: Table containing regional frequency plans and specifications [9]

Chapter 3: Implementation

3.1 Network Architecture

The wireless sensor network consists of several parts, but they can be categorized into two main components: the sensor nodes and the central gateway. The sensor nodes are portable devices that can be distributed across the agricultural field to collect and transmit sensor data. The central gateway is a stationary device that is positioned in a central location to receive and relay the sensor data. For this project, the central gateway also hosts a web-based user interface allowing the user to manage the sensor network and analyze the collected sensor data.

3.1.1 Sensor Node Parts

The sensor node consists of four parts: the microcontroller, the radio transceiver, the battery, and sensors. For this project, the microcontroller is an Adafruit Feather M0.

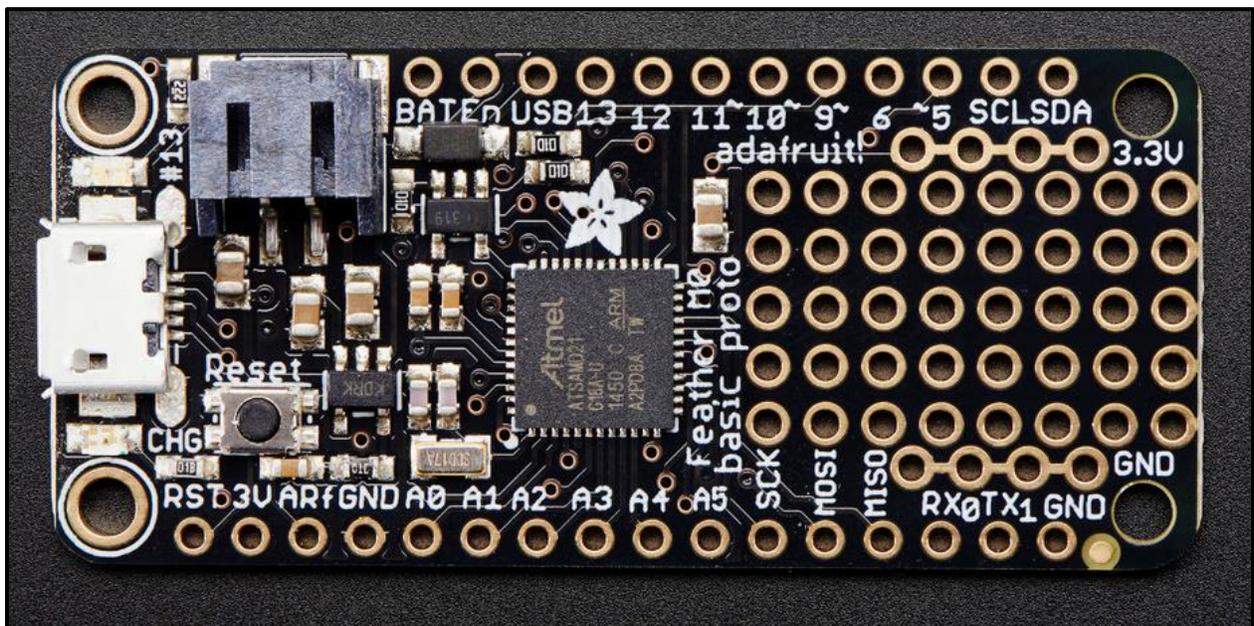


Figure 7: Sensor node microcontroller – Adafruit Feather M0 [10]

This microcontroller was selected because it is small, has a connector for 3.7V lithium polymer batteries, and has some extra room left over where the radio transceiver can be attached – in place of the prototype area.

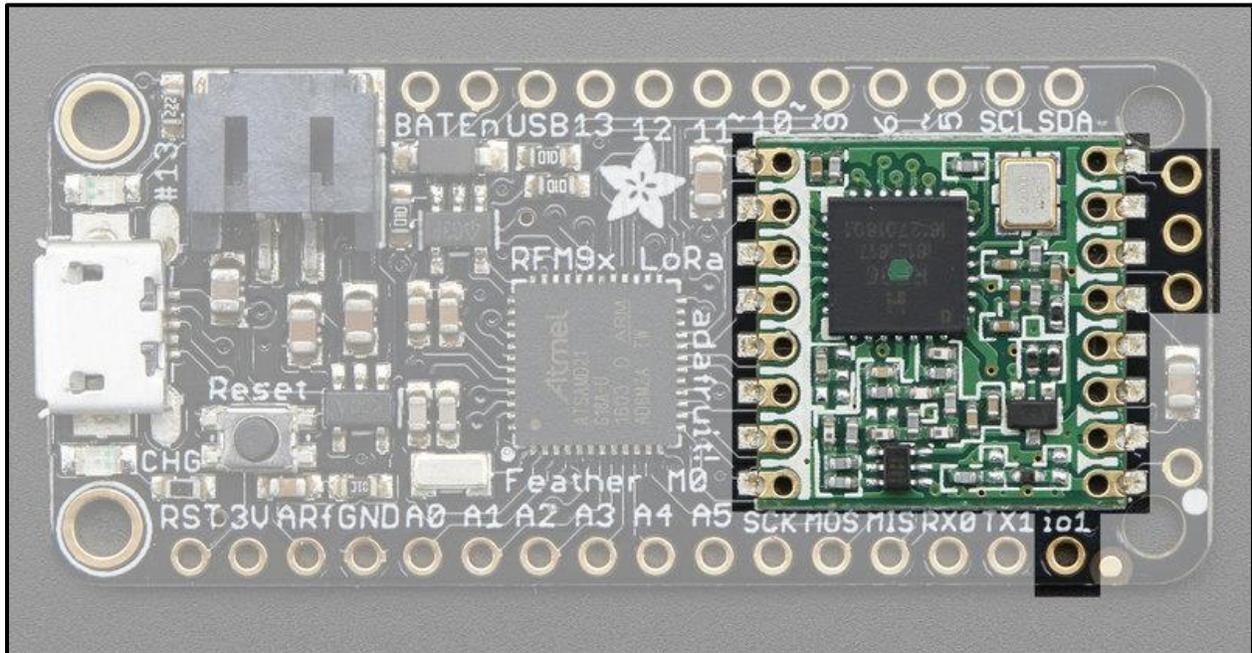


Figure 8: Sensor node radio transceiver – RF95 LoRa Radio [11]

The RF95 LoRa radio transceiver fits nicely on the microcontroller and has a spot where a simple wire antenna can be attached. The radio has an SPI interface and can be easily programmed using existing programming libraries. Packets are transmitted over license-free ISM radio frequency bands. For this project, the radios are configured to transmit over the 915MHz frequency band to comply with regional ITU radio regulations [12]. The power output for radio transmissions is configurable through software and can be set between +5dBm and +20dBm. The transceiver uses ~300uA during full sleep, ~120mA peak during +20dBm transmissions, and ~40mA during active radio listening [11]. While low power consumption is important for the sustainability of the sensor nodes, the real value of these radio transceivers come from the long distances they are able to communicate – particularly in rural areas.

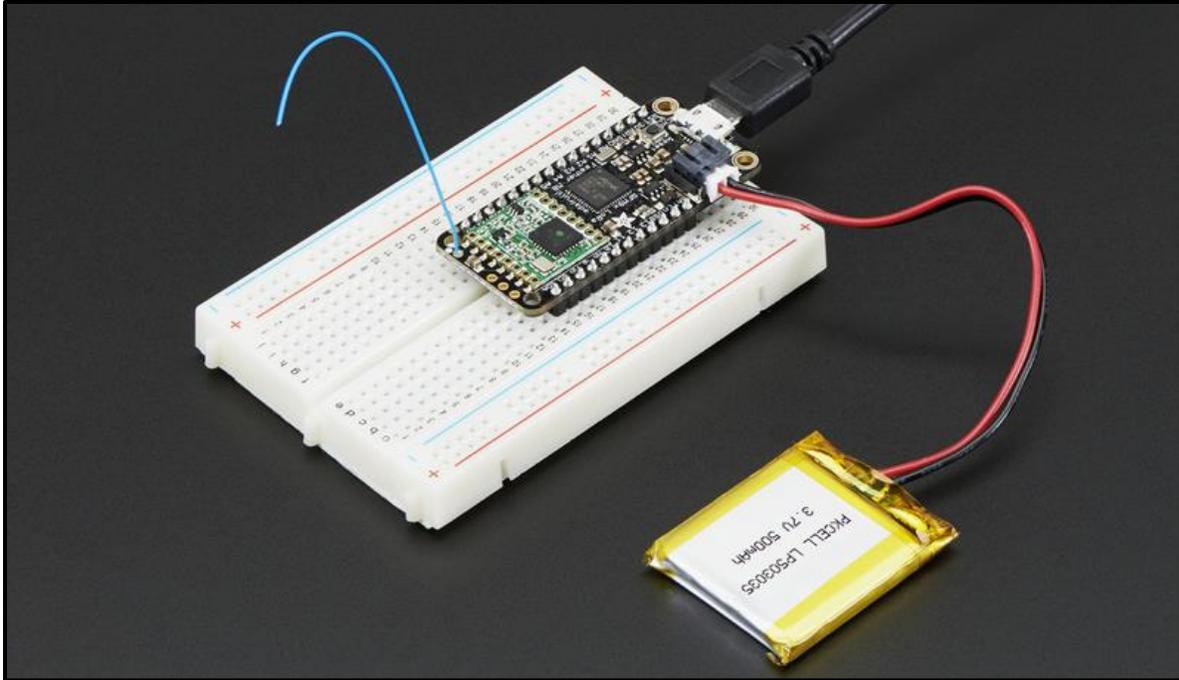


Figure 9: Sensor node using a micro USB, wire antenna, and 3.7V LiPo battery [11]

The built-in micro USB connector is used to upload the programming code, power the device, and automatically charge a connected battery. The lithium polymer batteries come in different capacities, generally ranging from 500mAh to 1200mAh. The batteries used in this project have 800mAh capacities. Although the batteries are labeled with 3.7V, they do not always operate at this voltage. Figure 11 depicts how the voltage starts at ~4.2V and drops to ~3.7V for the majority of the battery life, until it hits ~3.4V and quickly dies [13]. This is important because the sensor node's battery life can be estimated using its current operating voltage. Adafruit has made this easy by putting a double-100K resistor divider on the BAT pin and connected it to the A7 pin so that you can read the pin's voltage, and double it to get the battery voltage [14].

```
1. #define VBATPIN A7
2.
3. float measuredvbat = analogRead(VBATPIN);
4. measuredvbat *= 2; // we divided by 2, so multiply back
5. measuredvbat *= 3.3; // Multiply by 3.3V, our reference voltage
6. measuredvbat /= 1024; // convert to voltage
7. Serial.print("VBat: "); Serial.println(measuredvbat);
```

Figure 10: Algorithm to determine battery voltage [14]

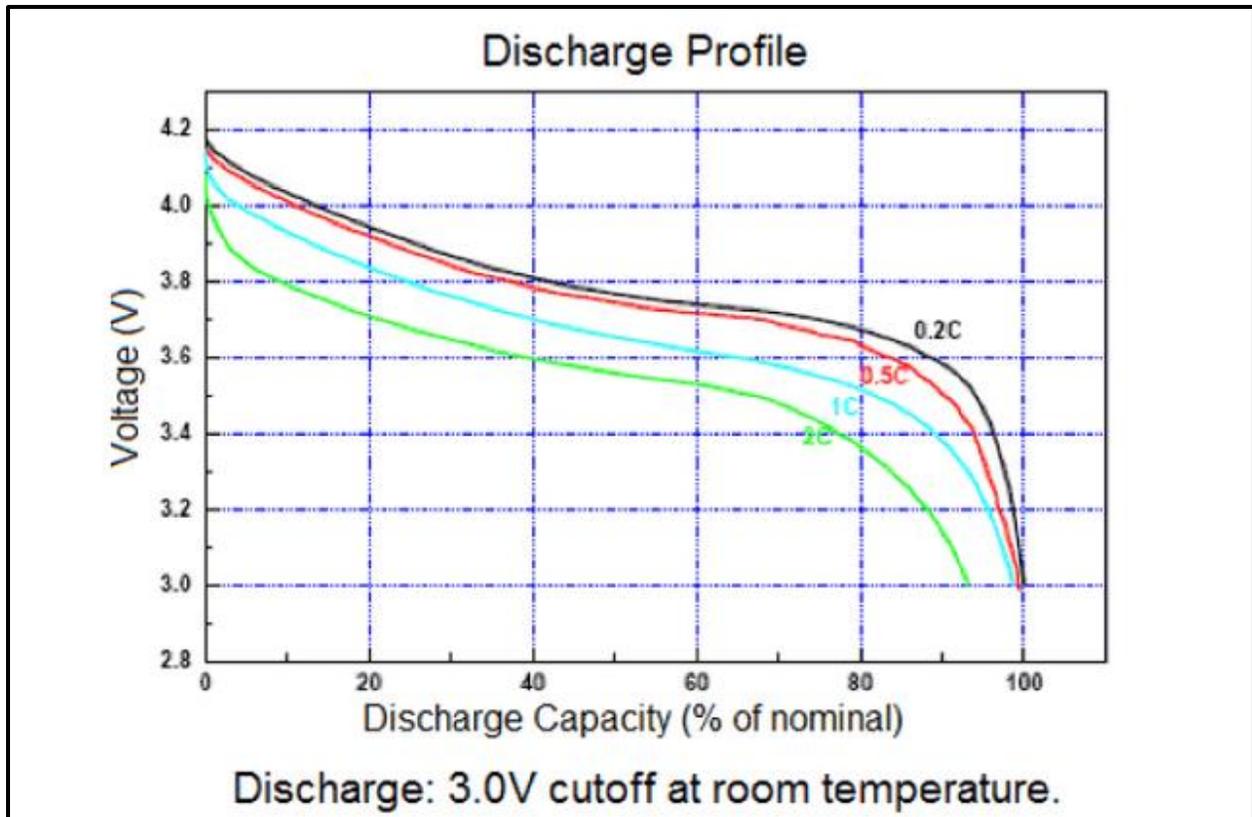


Figure 11: Discharge profile of a 3.7V LiPo battery [13]

3.1.2 Central Gateway Parts

The central gateway consists of three parts: the computer, the storage device, and the radio transceiver. For the computer, this project uses a Raspberry Pi 3 B+, which is one model in a collection of Raspberry Pi single-board computers developed with low cost in mind. With a small price tag, this version boasts a 1.4 GHz processor, 1 GB of SDRAM memory, and a WiFi module. It also has a slot for a microSD memory card which is used as the storage device holding the operating system. The central gateway uses that same LoRa radio transceiver as the sensor nodes, except it is not directly attached to the main circuit board. Instead, an expansion module – the Dragino LoRa hat – is attached to the Raspberry Pi’s general input-output pins. Although not implemented in this project, the hat also comes with a GPS module which could be helpful for future improvements.



Figure 12: Raspberry Pi 3 Model B+ [15]

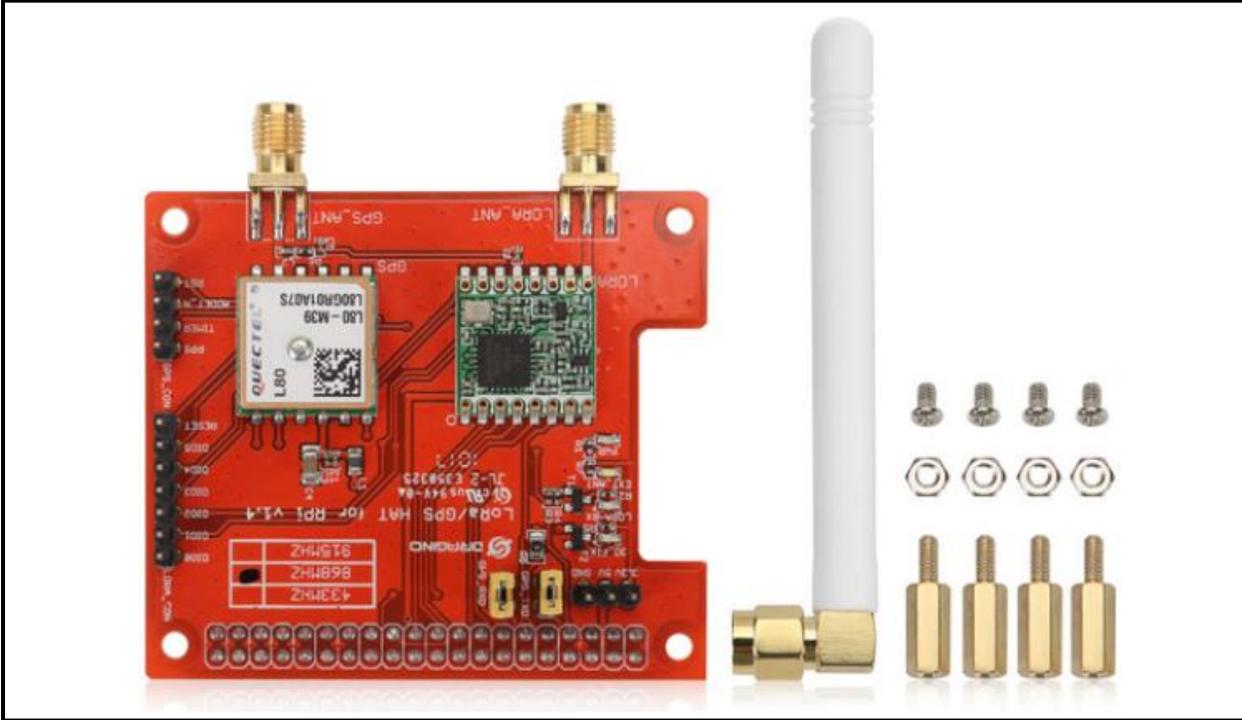


Figure 13: Dragino LoRa hat for the Raspberry Pi [16]

3.2 Data Flow

The data begins with the sensors that are connected to the sensor nodes, which are configured to periodically take sensor samples and store them in memory using a queue. Using this data structure removes the need to transmit them to the central gateway as they are taken. This is important because if the gateway is not available to receive them, the samples will be lost if they are not stored. The queue also makes way for a battery preservation technique where the devices briefly wake up, take sensor samples, store them, and fall back asleep. The sensor nodes are configured to periodically transmit all samples in the queue and only remove them upon acknowledgment from the gateway.

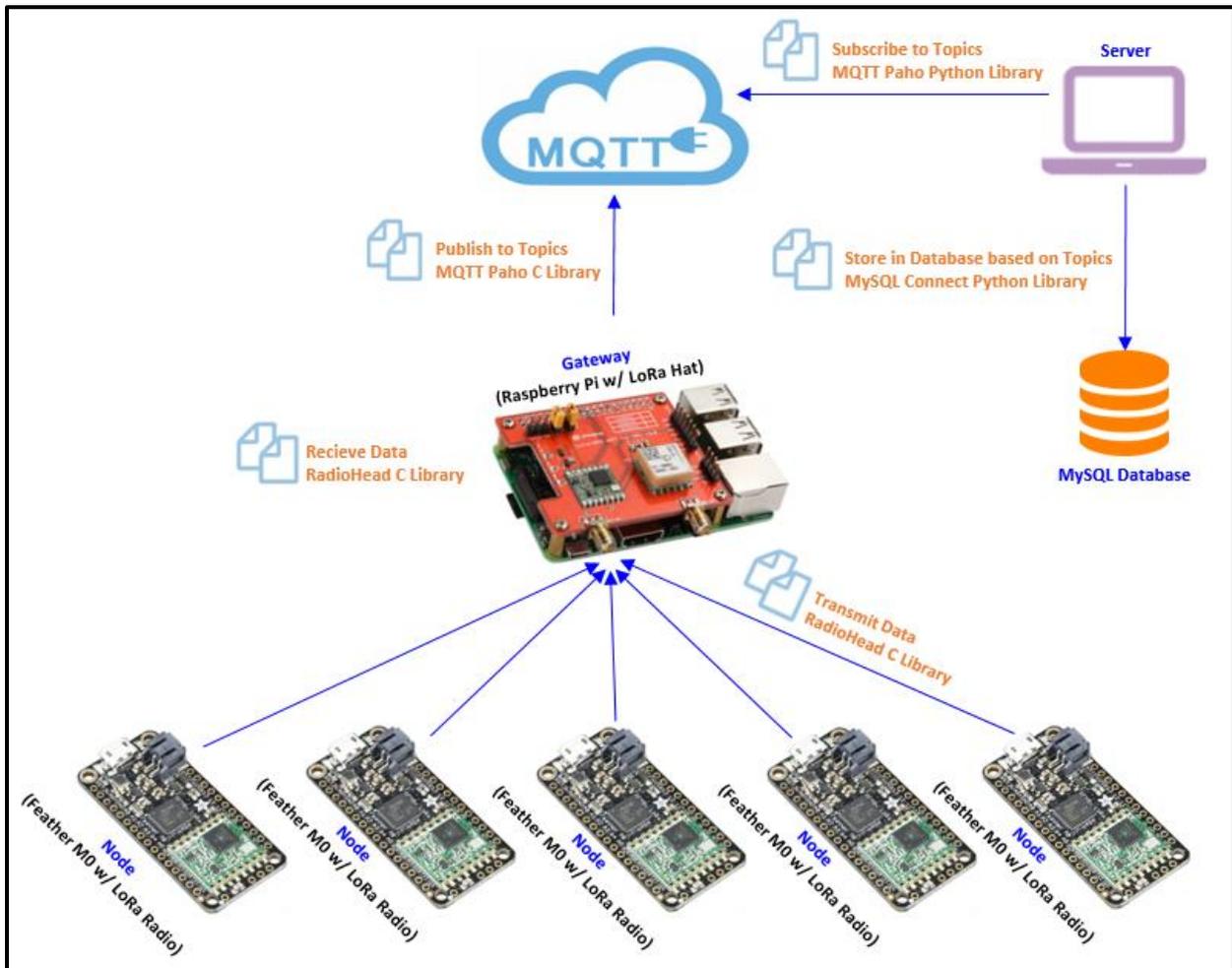


Figure 14: Data flow diagram of the sensor nodes, central gateway, and server

Once sensor samples are received by the gateway, they can either be stored locally on the gateway or forwarded to a MySQL database on an external server. Figure 14 illustrates a system where the samples are forwarded to the external server using a lightweight messaging protocol called Message Queueing Telemetry Transport (MQTT), which is a standard messaging protocol for the Internet of Things (IoT) because it is efficient, reliable, and scalable. It uses a publish/subscribe transport method that is ideal for connecting many devices using little code and network bandwidth [17]. This research project uses a simplified version of this system, where the sensor samples are immediately stored in a MySQL database on the central gateway and the MQTT publish/subscribe method is considered for future implementation.

3.3 Database

The wireless sensor network uses a MySQL database to store information about the central gateway, sensor nodes, and sensor samples. The Gateway entity has attributes for a description and GPS coordinates so that gateways can be named and pinned on a satellite network map within the user interface. Likewise, the Node entity also has attributes for a description and GPS coordinates so that sensor nodes can be named and located on the network map. The Node entity also stores information which is used by the gateway while it synchronizes with sensor nodes. A serial number – unique to each sensor node processor – is used to identify sensor nodes before synchronization. Timestamp attributes are included to track a sensor node's most recent transmission and synchronization with the gateway. Lastly, the Node entity has attributes for storing the number of minutes a sensor node should wait before sampling and transmitting samples. The Sample entity stores each sample with a single character representing the type of sensor used, a timestamp indicating when a sample was taken, and a decimal number for the measured value. The sample attribute is a counter used for determining how many

samples a node has taken since being powered on. The consecutive sample count can also help verify samples are not missing. Figure 15 is an entity-relationship (ER) diagram which shows the database entities and the relationships between them. A gateway can have zero to many nodes, but a node can only belong to a single gateway. Similarly, a node can take zero to many samples, but a sample can only be taken by a single node.

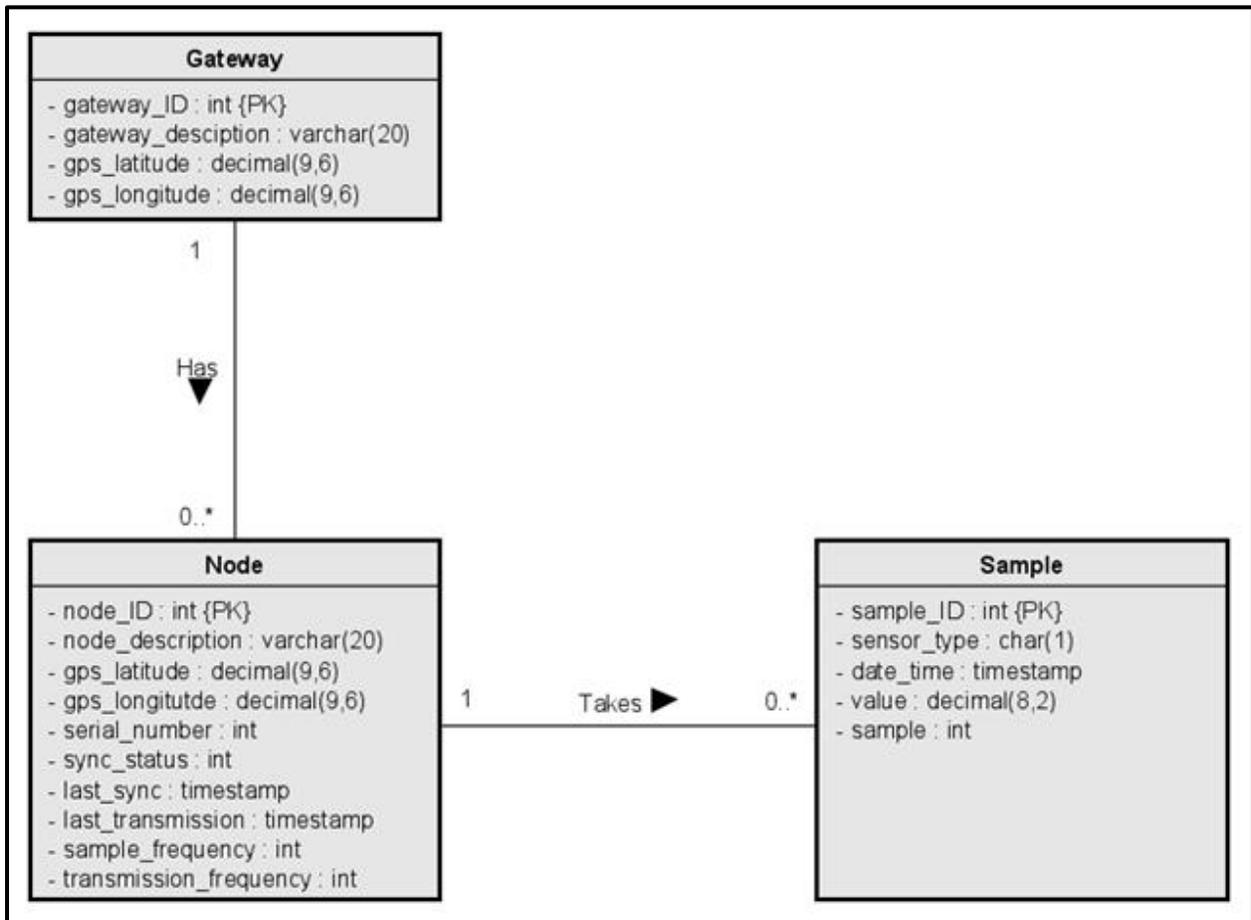


Figure 15: Entity-Relationship diagram

The ER diagram lists each attribute with their respective data types and can be used to write the SQL CREATE TABLE statements needed to setup the database. The relationships indicate foreign key constraints should be added the Sample and Node tables referencing their parent tables. A web user is created and granted the privileges needed for the website to connect to the database. Finally, a gateway is inserted during setup so that it can be named, and GPS coordinates can be assigned to it within the user interface.

```

lora_nodes.sql
1  CREATE DATABASE lorax;
2
3  GRANT ALL PRIVILEGES ON lorax.* TO 'web_user'@'%' IDENTIFIED BY 'web_pass';
4
5  CREATE TABLE Gateway (
6      gateway_ID INT AUTO_INCREMENT,
7      gateway_description VARCHAR(20),
8      gps_latitude DECIMAL (9,6),
9      gps_longitude DECIMAL (9,6),
10     PRIMARY KEY (gateway_ID)
11 );
12
13 CREATE TABLE Node (
14     node_ID INT AUTO_INCREMENT,
15     node_description VARCHAR(20),
16     gps_latitude DECIMAL(9,6),
17     gps_longitude DECIMAL(9,6),
18     serial_number VARCHAR(150),
19     sync_status INT DEFAULT 1,
20     last_sync TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
21     last_transmission TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
22     sample_frequency INT DEFAULT 10,
23     transmission_frequency INT DEFAULT 30,
24     gateway_ID INT DEFAULT 1,
25     PRIMARY KEY (node_id),
26     FOREIGN KEY (gateway_ID) REFERENCES Gateway(gateway_ID)
27     | ON DELETE NO ACTION ON UPDATE CASCADE
28 );
29
30 CREATE TABLE Sample (
31     sample_ID INT AUTO_INCREMENT,
32     sensor_type CHAR(1),
33     date_time TIMESTAMP,
34     value DECIMAL(8,2),
35     sample INT,
36     node_ID INT,
37     PRIMARY KEY (sample_id),
38     FOREIGN KEY (node_ID) REFERENCES Node(node_ID)
39     | ON DELETE NO ACTION ON UPDATE CASCADE
40 );
41
42 INSERT INTO Gateway (gateway_description) VALUES ('Gateway 1');

```

Figure 16: Database data definition language (DDL)

Chapter 4: Deployment

4.1 Code Location

The code for this project is stored in a GitHub repository. The repository contains a folder for each of the three code sets: the central gateway, the sensor nodes, and the web-based user interface. Although the code for the entire project is stored in one repository, each code set is deployed separately on their respective devices. The project code is accessible on GitHub at <https://github.com/jgres/cosc449>.

4.2 Deploy the Central Gateway

With the Raspberry Pi being used for the central gateway, the device can be accessed directly using peripheral devices or through the network using a Secure Shell (SSH) connection. After accessing the Raspberry Pi's command line interface, install the dependencies required to create the database and compile the server program.

4.2.1 Server Dependencies

The server program uses a C library specifically for the Raspberry Pi that provides access to the GPIO pins needed to communicate with the Dragino LoRa hat. To install the BCM2835 library, run the following commands in the CLI:

1. Download the library.

```
tar zxvf bcm2835-1.68.tar.gz
```

2. Navigate into library folder.

```
cd bcm2835-1.68
```

3. Configure and install the library.

```
./configure  
make  
sudo make check  
sudo make install
```

The server program also uses a MySQL-C connector to communicate with the database during node synchronization. To install the MySQL-C connector, run the following commands in the CLI:

1. Update the package manager.

```
sudo apt-get update && sudo apt-get upgrade -y
```

2. Update the package manager.

```
sudo apt-get install libmysqlclient-dev libmysqlcppconn-dev
```

4.2.2 Server Code Execution

To execute the server program run the following commands in the CLI after installing the server dependencies:

1. Clone the project repository.

```
git clone https://github.com/jgresl/cosc449.git
```

2. Navigate to the server source code.

```
cd lorax_server/RadioHead/examples/raspi/rf95_lorax_server
```

3. Build and run the server program.

```
make && sudo ./rf95_lorax_server
```

Once the server code executes, the central gateway initializes the LoRa radio driver using the defined configurations found in the top of the server code. The radio frequency, transmission power, spreading factor, bandwidth, preamble length, sync word, and coding rate are all configurable to optimize the wireless communications. After the gateway initializes, it will listen for sensor node messages addressed to it. If a message is received, the gateway evaluates the content to determine what to do with it. If the message payload is a serial number, the gateway will synchronize with the sending sensor node by providing it with their node ID, the current date and time, and how frequently it should sample and transmit sensor data. If the message payload is a JSON string containing sensor sample data, the gateway will parse the message and insert the sample into the server database. The gateway will continue to listen for messages from sensor nodes in an infinite loop.

It is worth noting that processes executed through an SSH connection could be terminated when the connection is closed. One way to avoid this is by using the Linux screen tool to start a screen session, run the program, and then detach from the screen session before closing the SSH connection [18]. Although there are other ways to ensure a process stays running, this is a good solution because you can re-attach to the screen and view the program's latest output in its console. This is beneficial if the server program crashes because the console output can help identify what crashed it. Another way to prevent processes from terminating is to execute them using a process manager program.

4.3 Deploy the User Interface

With the Raspberry Pi being used for the webserver, the device can be accessed directly using peripheral devices or through the network using an SSH connection. After accessing the Raspberry Pi's command line interface (CLI), install the dependencies required to host the web-based user interface.

4.3.1 Website Dependencies

The gateway uses the LAMP stack to host the user interface through the web browser. The stack consists of Linux, Apache, MySQL, and PHP. To install the LAMP stack, run the following commands in the CLI:

1. Update the package manager.

```
sudo apt-get update && sudo apt-get upgrade -y
```

2. Install the Apache, MySQL, PHP, and the PHP-MySQL connector.

```
sudo apt-get install apache2 mysql-server php php-mysql -y
```

3. Update Apache root folder permissions.

```
sudo chmod -R 777 /var/www/html
```

4. Restart Apache.

```
sudo service apache2 restart
```

4.3.2 Website Code Execution

Run the following commands after accessing the CLI:

1. Clone the project repository (already completed in server code execution).

```
git clone https://github.com/jgresl/cosc449.git
```

2. Move the `lorax_website` folder to the Apache webserver root directory.

```
mv cosc449/lorax_website /var/www/html/lorax_website
```

The Apache webserver hosts the files in its root directory. That is, it accepts requests from clients and sends responses back. The URL entered into the web browser will point to a PHP file, which will connect to the database using the PDO_MYSQL driver and query the database using Structured Query Language (SQL) prepared statements to dynamically return Hypertext Markup Language (HTML). Cascading Style Sheets (CSS) are used to describe the presentation of the HTML. JavaScript is used to connect with the Google Maps API, generate D3 time series charts and submit forms on table input changes.

If the gateway is accessed directly with peripheral devices and the operating system GUI, the web-based user interface can be accessed by opening the locally installed web browser and entering the IP address 127.0.0.1 or <http://localhost/> on port 80 by default.

If the gateway is connected to the local network through a direct LAN or wireless WiFi connection, then the web-based user interface can be accessed by opening a web browser on another device within the network and entering the internal IP address assigned to the gateway by the network router. The internal IP can be determined by accessing the gateway's terminal and checking the network configurations, or by accessing the network router's user interface and looking through a list of devices connected to the network. There are also third-party mobile applications that can connect to the network and display a list of other connected devices. A static IP address can be assigned to the gateway to prevent the internal IP address changing when power is lost.

If the gateway is connected to the local network through a direct LAN or wireless WiFi connection that is also connected to the Internet, the web-based user-interface can be accessed by opening a web browser on another Internet enabled device and entering the external IP address for the local area network. To enable this, port forwarding rules will need to be configured on the router to allow incoming traffic and direct it the internal IP address of the central gateway and through the webserver's hosting port.

Port Forwarding

Enter ports or port ranges required to forward Internet applications to a LAN device below.

1. Set the LAN/WAN port and IP information.

Select LAN Device:	gateway(192.168.1.65) ▼
LAN IP Address:	192.168.1.65
External (WAN) Start Port:	80
External (WAN) End Port:	80
Internal (LAN) Start Port:	80
Internal (LAN) End Port:	80
Protocol:	TCP ▼

Figure 17: Example of port forwarding to enable external access to the gateway

Figure 17 shows an example of a port forwarding rule that accepts external Internet traffic on the external port 80 and directs the traffic to the internal port 80 on the gateway at the internal IP address 192.168.1.65. Any external port could be entered but the internal port must match the port configured with the Apache webserver, which is port 80 by default. Assigning both the external and internal port to 80 will eliminate the need to specify the port number in the web browser URL when accessing the gateway.

At this point, another port forwarding rule can be setup to allow external access into the gateway using SSH through port 22. This can be beneficial if a remote connection is needed to update the code, restart the program, reboot the device, or perform maintenance. If security is a concern, choosing a different external port other than 22 is one way to reduce unwanted network traffic. With port 22 exposed, it is a good idea to ensure the Raspberry Pi's username and password have been changed from the default.

4.4 Deploy the Sensor Nodes

With the Adafruit Feather M0 as the sensor node's microcontroller, the sensor node code must be uploaded to it through the on-board microUSB port. This project uses Visual Studio code with the C/C++ extension to compile the code and the PlatformIO extension to upload the code and monitor the serial connection.

4.4.1 Client Dependencies

First, download and install the latest release of Visual Studio Code for Windows, Linux, or Mac by visiting <https://code.visualstudio.com/Download> and clicking the appropriate link for your operating system. Open Visual Studio Code and click on the extension marketplace button on the left side menu. Then, search for the C/C++ extension and install it. Finally, search for the Platform IO extension and install it. Figures 18 and 19 show the extensions found in the extension marketplace.

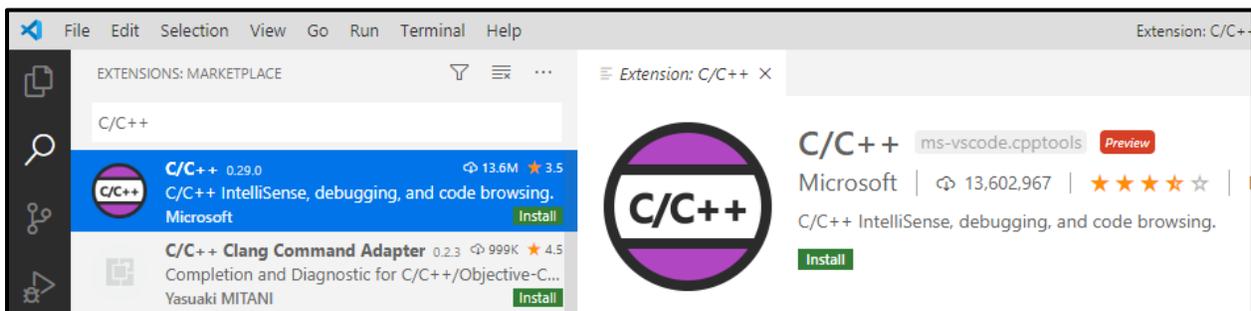


Figure 18: Installing the Visual Studio Code extension for C/C++

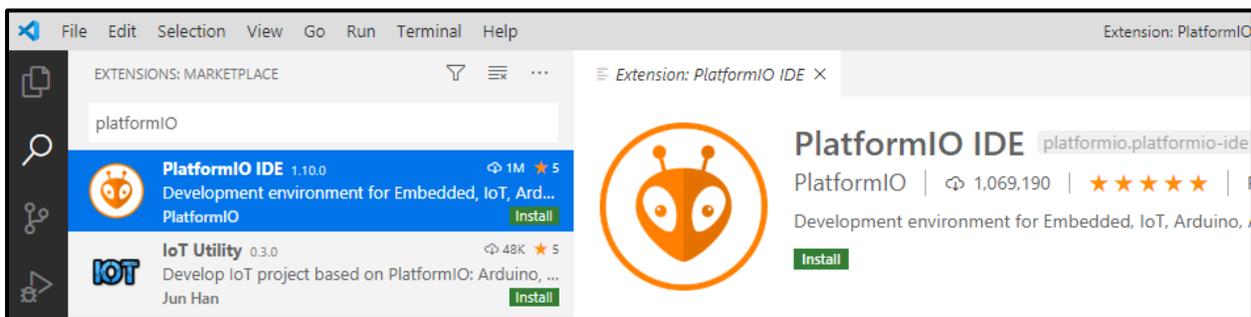


Figure 19: Installing the Visual Studio Code extension for PlatformIO

The client code uses some additional libraries to drive the digital BME280 temperature sensor, use the RF95 LoRa radio, parse JSON and run the RTC clock. To install them, click on the Libraries button within the PlatformIO extension and enter the library names in the search bar. The required libraries are listed below:

- Adafruit BME280 Library – by Adafruit
- Adafruit Unified Sensor – by Adafruit
- RadioHead – by Mike McCauley
- jsmn – by Serge Zaitsev

4.4.2 Client Code Execution

After the dependencies are installed, download the project repository from GitHub or clone it using Git. Ensure the sensor node is connected to your computer. Then, open the `lorax_client` project in Visual Studio Code and click on the PlatformIO logo in the left side menu. Using the status bar on the bottom, click on the checkmark to build the program, the cloud to upload the program to the node, or the plug to open the serial monitor. Once the client code is uploaded to the microcontroller, the sensor node initializes the LoRa radio driver using the defined configurations in the top of the server code. The sensor node LoRa radio should have the same configurations defined as the gateway LoRa radio. After the sensor node initializes, it will retrieve a serial number unique to its processor and send it to the central gateway to initialize synchronization. The gateway will return a JSON string containing the node ID, the current date and time, the sample frequency and the transmission frequency. The sensor node parses the JSON string and updates the RTC with the current date and time and updates the sample and transmission frequency global variables. The sample frequency is used to set interrupt alarms for taking samples. The transmission frequency is compared with a counter. Samples are stored in a queue until the transmission counter time matches the configured transmission frequency.

Chapter 5: Walkthrough

5.1 Access the User Interface Internally

With the central gateway connected to the local area network – and the web-based user interface deployed – the website can be accessed internally by opening a web browser on another device within the network and entering the website's URL which is comprised of the internal IP address assigned to the gateway by the network router and the end point to the main webpage stored in the Apache root directory. In the case of this deployment, the internal URL is http://192.168.1.65/lorax_website/main.php.

5.2 Access the User Interface Externally

The web-based user interface can also be accessed from a computer outside of the network if the router is configured to allow external access to the gateway. To access the website externally, open a browser from a computer outside of the network and enter the external IP address of the network along with the external WAN port. A Port forwarding rule should be setup to forward network traffic on this port to the internal IP address assigned to the gateway by the network router using the internal LAN port where Apache web server is accessible, which is port 80 by default. One easy way the external IP address of the network can be determined is by opening the web browser of any computer connected to the network and accessing a web service that will return your external IP address back to you. A simple example which only returns your external IP address is <https://ipof.in/txt>. In the case of this deployment and port forwarding rule, the external URL is http://50.98.99.115:80/lorax_website/main.php. If the external WAN port in the port forwarding rule is set to 80, it can be omitted from the external URL and simplified to http://50.98.99.115/lorax_website/main.php.

5.3 Manage the Network Setup

After accessing the site, click on the Manage Network link in the left side menu to view the network map and configure the connected components. The network map uses the Google Maps API to display the map and it queries the database to get the GPS coordinates for the gateway and nodes. Figure 20 shows an example network where the central gateway (G) is setup indoors along with sensor nodes setup indoors (1 and 2) and more sensor nodes setup outdoors on a covered patio (3 and 4). The gateway and nodes will not be visible on the network map until the components are connected and GPS coordinates are assigned to them.

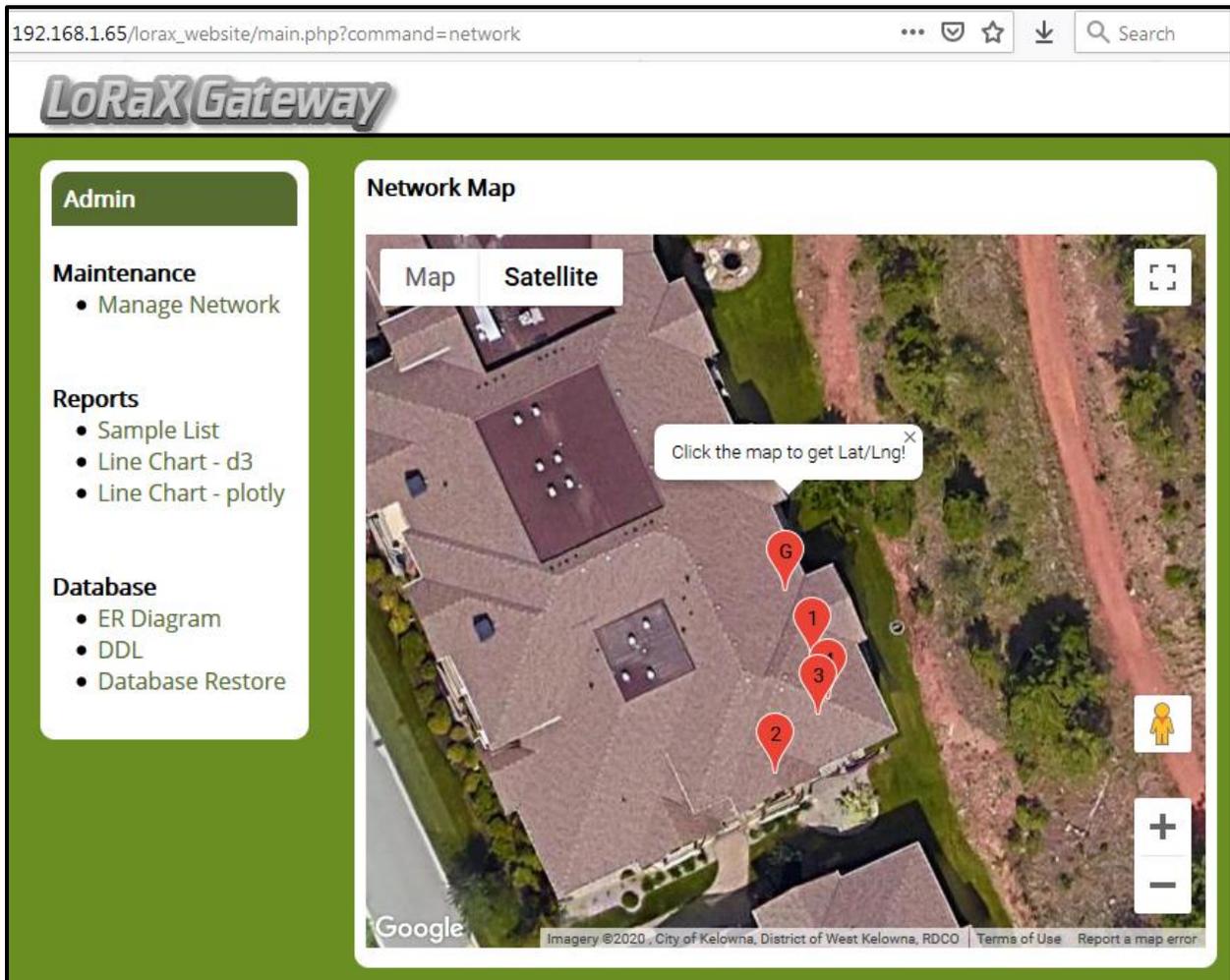


Figure 20: Network map using the Google maps API

Beneath the network map are the gateway and node configuration tables. A gateway will already exist in the gateway configuration table because it was already created using an SQL INSERT STATEMENT during the database setup. The gateway description and GPS coordinates will be blank by default. The gateway can be named in the description field and the GPS coordinate can be entered to make it visible on the network map. Click on anywhere on the network map to get the latitude and longitude of the clicked location. Copy and paste the coordinates one at a time into their respective fields within the gateway configuration table. Changes to the configuration table fields will automatically update in the database, except for the gps_latitude field; it waits for the longitude to be entered as well before updating. This is so the page does not refresh before you are ready to copy the longitude coordinate from the set returned by clicking the network map.

Gateways

gateway_ID	gateway_description	gps_latitude	gps_longitude
1 - ON	<input type="text" value="Gateway 1"/>	<input type="text" value="49.859230"/>	<input type="text" value="-119.605260"/>

Nodes

node_ID	node_description	gps_latitude	gps_longitude	sample	transmit	sync_status
1 - ON	<input type="text" value="Living Room"/>	<input type="text" value="49.859190"/>	<input type="text" value="-119.605235"/>	10 <input type="text" value="↓"/>	60 <input type="text" value="↓"/>	Synced
2 - ON	<input type="text" value="Main Bedroom"/>	<input type="text" value="49.859120"/>	<input type="text" value="-119.605270"/>	10 <input type="text" value="↓"/>	60 <input type="text" value="↓"/>	Synced
3 - OFF	<input type="text" value="Outside Thermist"/>	<input type="text" value="49.859155"/>	<input type="text" value="-119.605230"/>	10 <input type="text" value="↓"/>	60 <input type="text" value="↓"/>	Synced
4 - ON	<input type="text" value="Outside BME280"/>	<input type="text" value="49.859165"/>	<input type="text" value="-119.605220"/>	10 <input type="text" value="↓"/>	50 <input type="text" value="↓"/>	Required

III

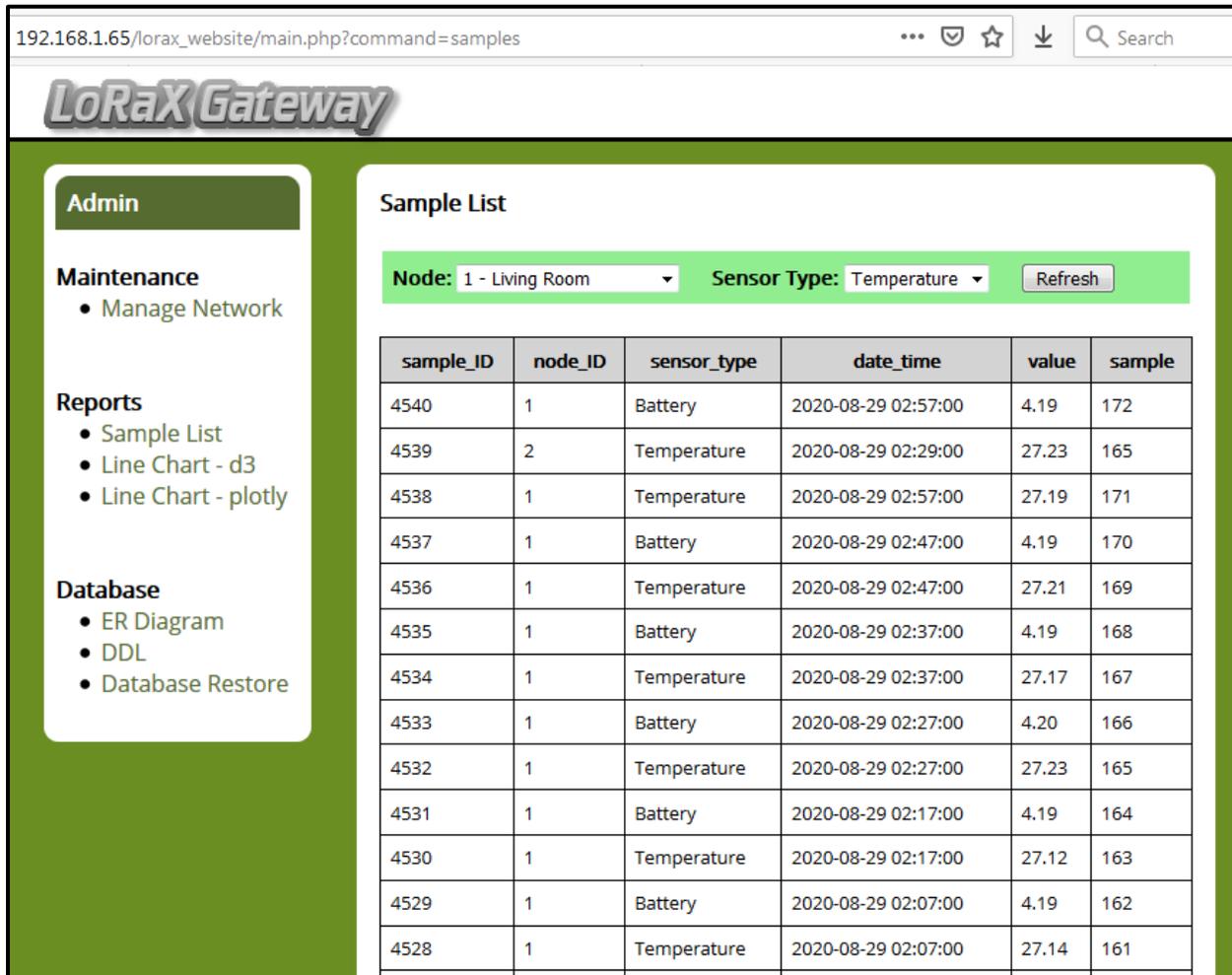
Figure 21: Gateway and node configuration tables

There will not be any nodes displayed on the node configuration table until the sensor nodes are powered on and synchronized with the gateway. To begin adding sensor nodes to the network, connect a battery to a sensor node. The sensor node will send its unique serial number to the gateway until it receives a response. When the gateway receives the serial number from the sensor node, it will query the database to determine if the node has already been synchronized. If the serial number does not exist in the Node table, it is inserted, and the sensor node is assigned a unique ID. Finally, the gateway returns a JSON string to the sensor node containing the node ID, the current date and time, the sample frequency and the transmission frequency. Once synchronized, the same steps that were used to configure the gateway can be followed to name and assign coordinates to the sensor nodes. The sample and transmission frequencies can be modified in the node configuration table. Changing either of them will update the sync status to “Required”, which will flag the gateway to initiate a re-synchronization with the sensor node the next time it receives a message from it.

By default, the sample frequency is set to 10 and the transmission frequency is set to 30. With this configuration, the sensor node will sample sensor data and store them in the queue every 10 minutes. 3 sample sets will be added to the queue before transmitting them to the gateway every 30 minutes. If the gateway acknowledges receipt of a sample, it is removed from the queue. The sensor node will transmit the sample in the front of the queue until it receives acknowledgment or until it has attempted an amount of transmissions equal to the size of the queue. If the gateway is not available, the samples will remain in the queue and are transmitted during the next scheduled transmission. If the gateway is not available for a long period of time, the queue will become full and the sensor node will be unable to store any more samples. Although the gateway should always be available, configuring the sample and transmission frequencies can help ensure the queue does not fill up quickly. Increasing the sample and transmission frequencies will also increase the longevity of the battery by reducing the total number of operations.

5.4 View Sample List

After powering the components, the sensor nodes will synchronize with the gateway and start sampling immediately. Click on the Manage Network link in the left side menu to view all the samples received by the gateway and inserted into the Sample table.



The screenshot shows the LoRaX Gateway web interface. The browser address bar displays `192.168.1.65/lorax_website/main.php?command=samples`. The page title is "LoRaX Gateway". On the left, there is a navigation menu with sections: "Admin", "Maintenance" (containing "Manage Network"), "Reports" (containing "Sample List", "Line Chart - d3", and "Line Chart - plotly"), and "Database" (containing "ER Diagram", "DDL", and "Database Restore"). The main content area is titled "Sample List" and features a filter bar with "Node: 1 - Living Room" and "Sensor Type: Temperature" dropdown menus, and a "Refresh" button. Below the filter bar is a table with the following data:

sample_ID	node_ID	sensor_type	date_time	value	sample
4540	1	Battery	2020-08-29 02:57:00	4.19	172
4539	2	Temperature	2020-08-29 02:29:00	27.23	165
4538	1	Temperature	2020-08-29 02:57:00	27.19	171
4537	1	Battery	2020-08-29 02:47:00	4.19	170
4536	1	Temperature	2020-08-29 02:47:00	27.21	169
4535	1	Battery	2020-08-29 02:37:00	4.19	168
4534	1	Temperature	2020-08-29 02:37:00	27.17	167
4533	1	Battery	2020-08-29 02:27:00	4.20	166
4532	1	Temperature	2020-08-29 02:27:00	27.23	165
4531	1	Battery	2020-08-29 02:17:00	4.19	164
4530	1	Temperature	2020-08-29 02:17:00	27.12	163
4529	1	Battery	2020-08-29 02:07:00	4.19	162
4528	1	Temperature	2020-08-29 02:07:00	27.14	161

Figure 22: List of samples with filter options

The samples list can be filtered to only display sensor data for a specific node and sensor type making the desired selections in the drop down lists and clicking on the refresh button. Figure 22 shows an unfiltered list with samples from nodes 1 and 2. There are samples for the temperature in degrees Celsius and battery voltage in volts. The samples are timestamped and segregated by 10 minutes.

5.5 View Line Charts

Click on either of the Line Chart links in the left side menu to view the samples in a time series line chart. By default, the charts will display the entire temperature data set for the first node. There are buttons above the chart and a sliding timeline bar beneath the chart which can be used to zoom in and zoom out to specific timeframes. The chart can display data sets for multiple nodes. To do this, hold the CTRL button on the keyboard while clicking on the nodes that should be displayed, then select the sensor type and click the refresh button. Figure 23 shows a chart containing the temperature data sets for nodes 3 and 4 over approximately 24 hours. The temperatures increase and decrease at the same rate, but they are consistently ~4 degrees apart. This could be because the sensor nodes are using different sensors. As described in the node names, node 3 uses an analog thermistor sensor while node 4 uses a digital BME280 sensor. This chart supports the research by measuring and displaying environment differences.

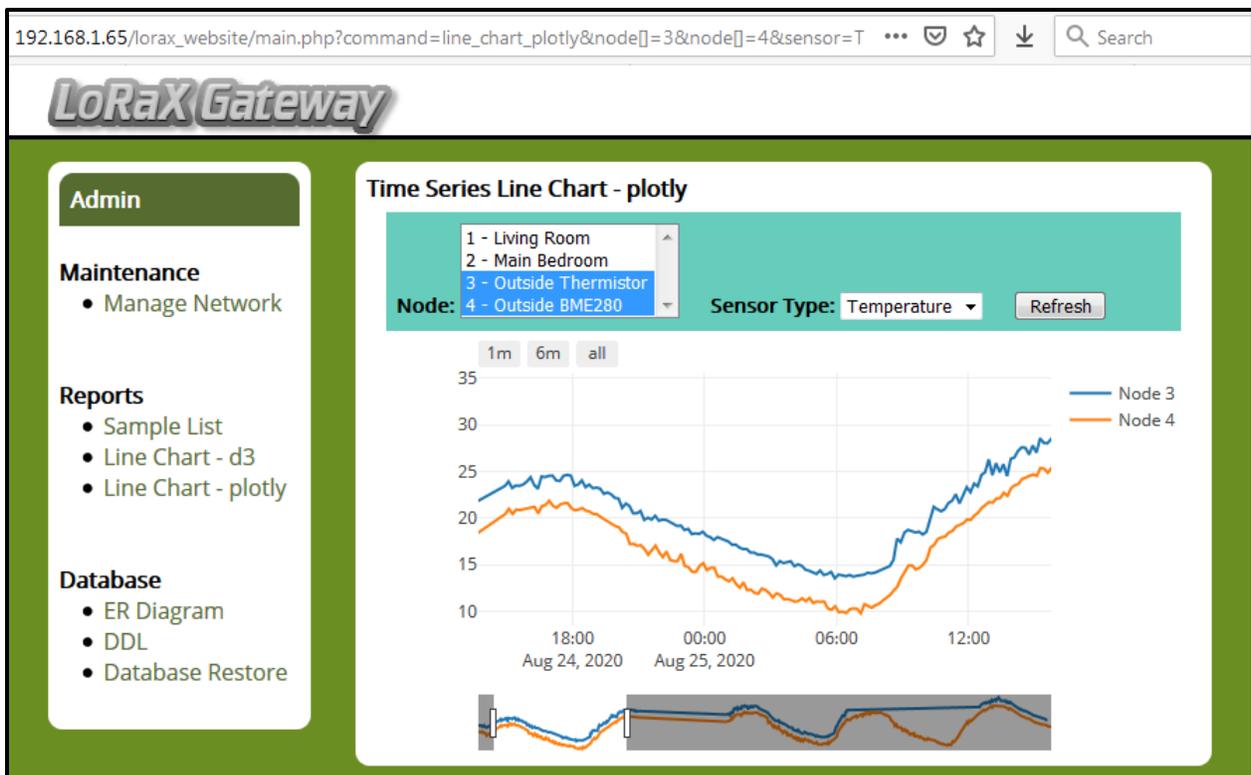


Figure 23: Time series line chart with outside node temperature data

As the sensor nodes collect both temperature and battery samples, the chart can also show the battery voltages by selecting Battery as the sensor type and then clicking the refresh button. Figure 24 shows the battery voltages for the nodes that were deployed outside, which used different sensors to take temperature samples. In this experiment, the sensor nodes were connected to batteries and also to a switch powering them with microUSB cables. When the switch is turned on, the battery voltage increases as the battery charges. When the switch is turned off, the battery voltage decreases as the battery drains. This report is interesting because it shows the battery voltage for node 4 drops quicker than the battery voltage for node 3. The difference in battery drain is a result of the nodes using different sensors with different code to sample data. With the code used for the analog sensor, power is only sent to the sensor while it is sampling. This chart shows how battery life can be preserved through optimization, and it allows users to monitor the sensor node battery states.

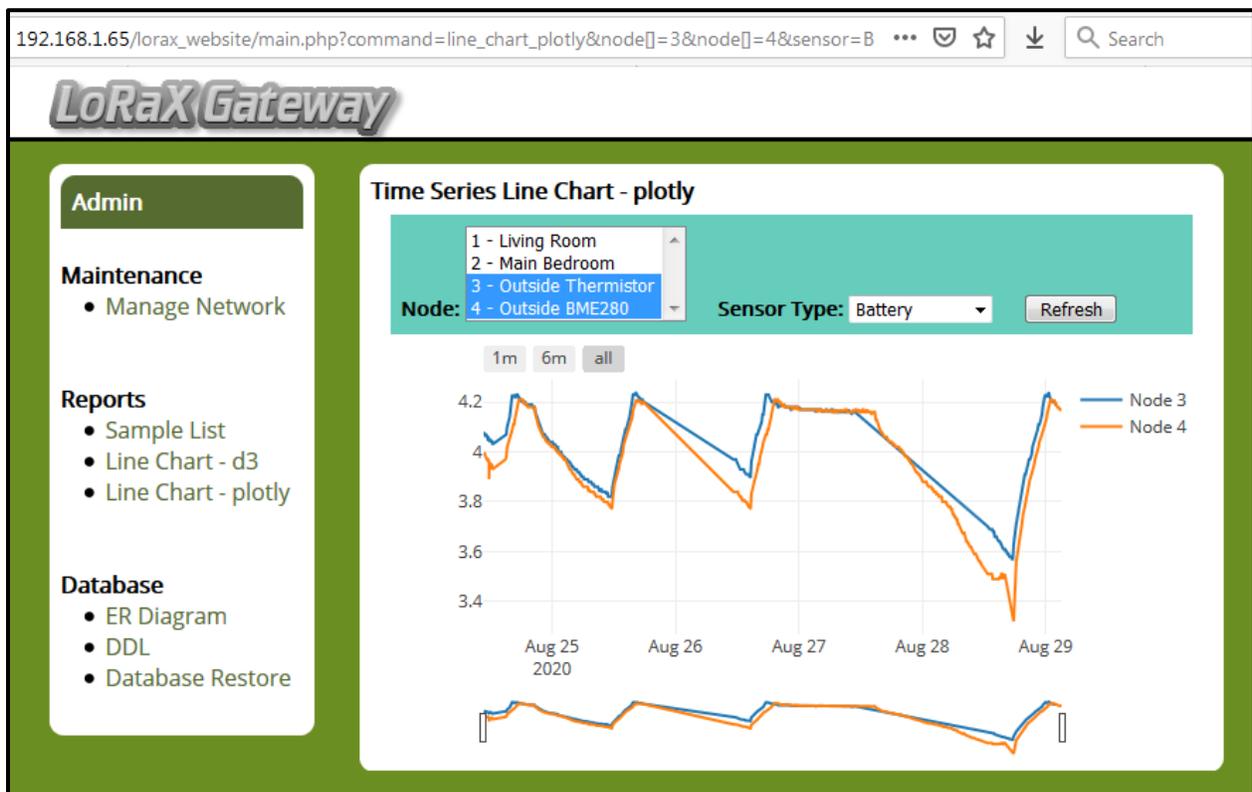


Figure 24: Time series line chart with outside node battery data

5.6 Network Reset

In some cases, a user might want to reset the network and start fresh. For example, if the equipment is moved to another farm, the samples are no longer relevant to the sensor nodes. To reset the network, click on the Database Restore link in the left side menu. Click the Restore button and then click the Yes button once prompted for confirmation. The database restore function will delete all records in the Node and Sample table. Sensor nodes will need to be reset in order for them to re-synchronize with the gateway.

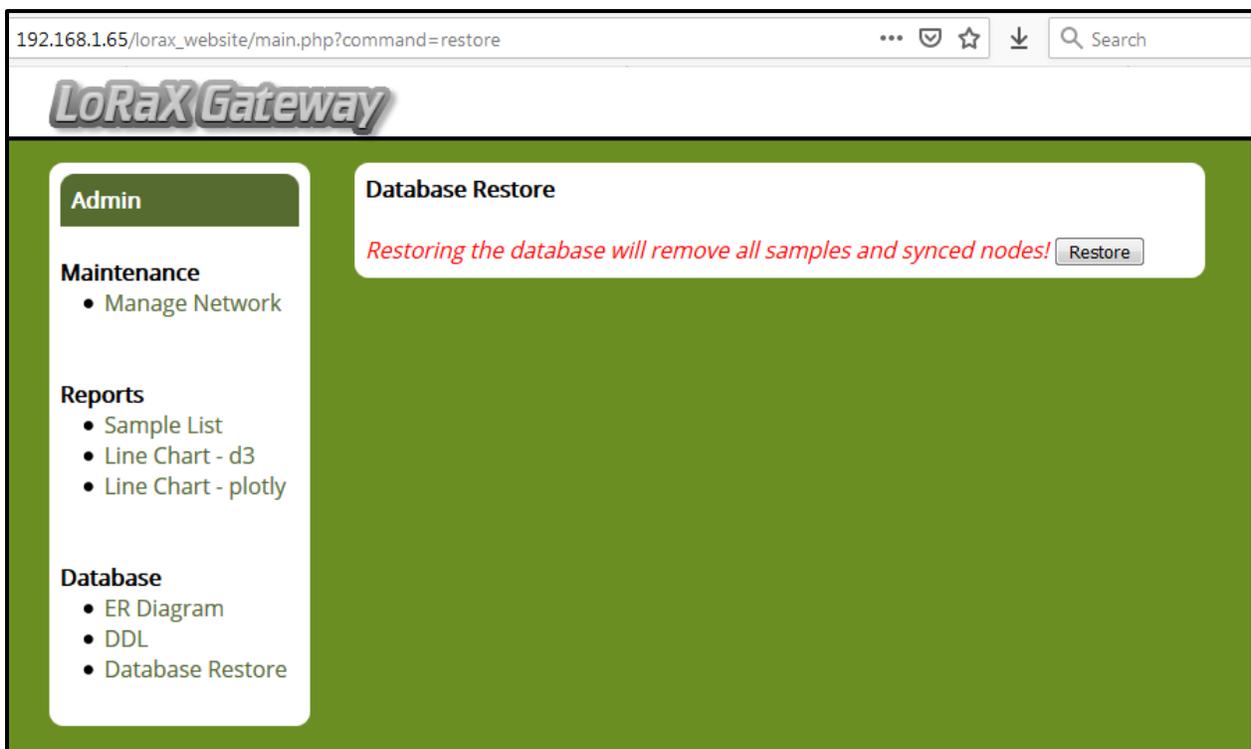


Figure 25: Database restore page to remove all samples and synced nodes

Chapter 6: Future Work

6.1 Decreasing Power Consumption

Devices that run continuously for long periods of time will drain their batteries quickly. The sensor nodes can be configured to consume less power by putting the radio or the entire device to sleep when it is not in use. The RadioHead software library used to control the RF95 LoRa radio includes a `sleep()` function that will put the radio into a low-power sleep mode until `idle`, `transmit`, or `receive` functions are called afterwards [19]. While putting the radio to sleep would definitely help preserve power, the microcontroller will still be providing power to the CPU, RAM, and peripherals. One software library that supports powering down the Adafruit Feather M0 (ATSAMD21) is `Adafruit_SleepyDog`, which can be found on GitHub at https://github.com/adafruit/Adafruit_SleepyDog. Powering down the microcontroller requires a scheduled wake-up. Some possible challenges with this method of decreasing power consumption will be to ensure the clock continues to run during sleep mode and the stored samples are not lost.

6.2 Data Persistence

The sensor nodes, in their current state, are not able to keep track of time or remember which samples are in the queue when the battery dies. This is because the time library counter and the queue data are stored using volatile memory. Ultimately, this limits the overall data collection because samples cannot be taken until the time is known and samples not yet sent to the gateway are lost. If the sensor node knows the exact date and time when it wakes up, then it can begin sampling before having synchronized with the gateway. If the sensor node can wake up and still access the sample data, it will not have to stay awake in order to keep them in memory.

One solution that could help with data persistence is to include the Adalogger FeatherWing as another part within the sensor node component. Figure 26 shows the add-on part, which includes an I2C real-time clock (32KHz crystal) with a battery backup and a microSD socket. The part can be easily attached to the Adafruit Feather M0 microcontroller using Feather Stacking Headers or Feather Female Headers [20].

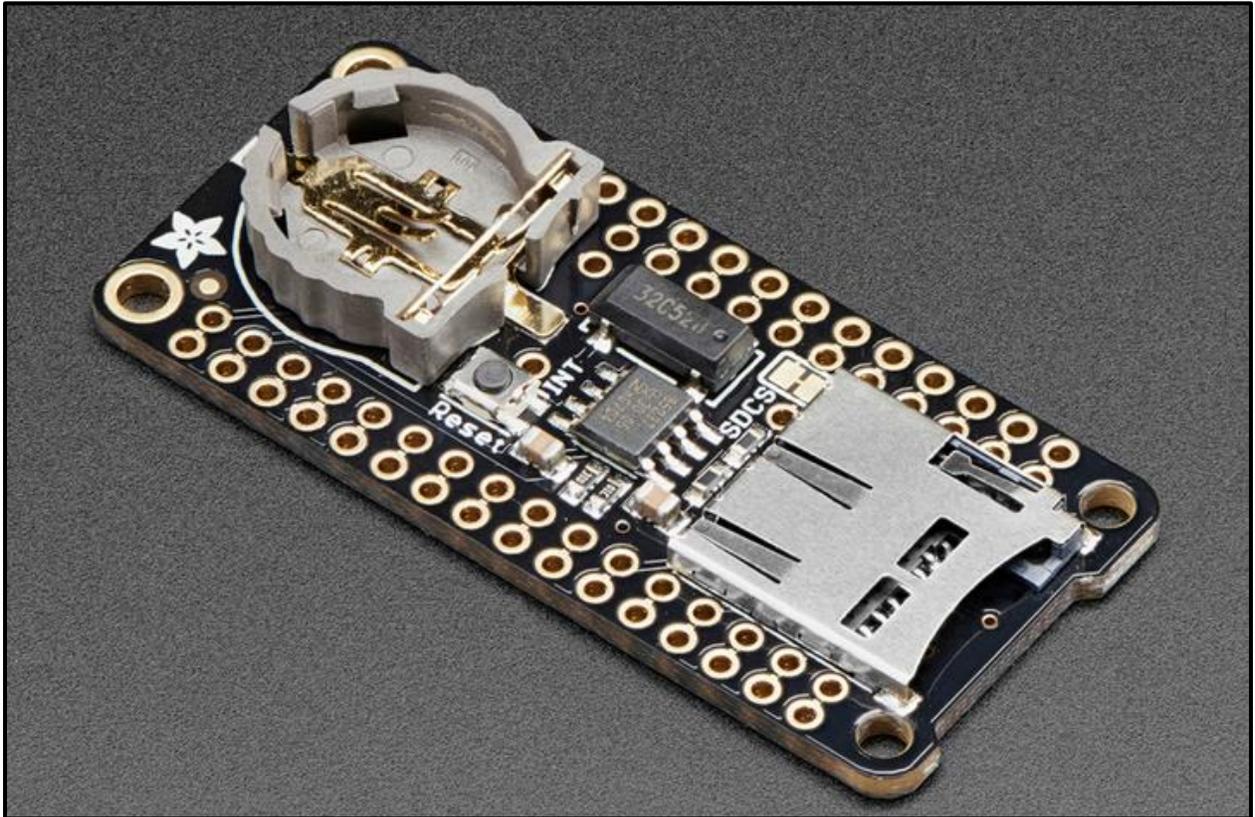


Figure 26: Adafruit Feather M0 add-on with RTC clock and SD card slot [20]

While using the Adalogger Featherwing does provide a way to ensure time and sample data are not lost, it also comes with increased cost, maintenance, and power consumption. An implementation with 80 sensor nodes would require an additional 80 Adalogger FeatherWings, CR1220 coin cells, and microSD cards. Maintenance is increased as the coin cells will require some replacing. The samples currently stored in memory using a queue, could be stored on the microSD instead, but reading from and writing to the microSD card will also consume additional power and drain the main battery faster.

6.3 Payload Optimization

The message payloads for the sample data currently uses a JSON string, which contains the node ID, sensor type, sample timestamp, and sample value. The JSON string is then parsed by the gateway upon receipt. This is useful because the important values do not always use the same number of characters and are not always located in the same element of the character array. While this message payload works and is flexible to extract data from, it also uses up to 150 bytes per message. The length of the message payload is important to consider because longer messages use more memory and transmission time than shorter messages. The queue in the sensor node code currently stores 100 messages at 150 bytes each. Although the Adafruit Feather M0 microcontroller has 32KB of RAM, only 15KB is currently reserved for the queue. Shortening the messages would allow the queue to be setup to hold more of them. More importantly, shortening the messages would also shorten the payloads sent during transmission. Shorter payloads can be communicated over the sensor network using less time and power. In summary, the wireless sensor network would benefit from optimized payloads because the sensor nodes could store more samples in memory, produce less radio interference, and consume less power.

6.4 Agricultural Implementation

With the wireless sensor network being functional, the next step will be to deploy the components into an agricultural field and start collecting data. There are many local orchards and vineyards that could benefit from having the system, but they will be skeptical to invest until they are able to realize the benefits. As there are costs associated with the implementation, the system must provide immediate or promised value in return.

Grapes yield a high return on investment; it would be most ideal to find a viticulturist who is willing to collaborate and provide guidance on how the sensor data can provide value to them. Upon recommendation, further advancements can be made to the user interface or the data can be sent to an external platform already capable of providing the desired data representations. Looking even further ahead, the sensor data could also be used to automate certain agricultural tasks, such as controlled watering or automatic dispatching of drones to apply fertilization and pesticides.

For the best radio performance, the gateway and sensor nodes should be installed above vegetation with unobstructed space between them. Vineyards typically have posts already installed to help growing operations, which are good places to fix the sensor nodes on top of. If soil moisture data is needed, sensors would be wired from the ground up to the sensor node. The sensor nodes can be spaced apart depending on the resolution of data needed. As high maintenance is expected in the early stages of implementation, it would be a good strategy to deploy a small number of sensor nodes in easily accessible locations and expand the network once stability is achieved and more coverage is needed.

6.5 Equipment Protection

Once the wireless sensor network components are deployed outside, they will be vulnerable to climatic conditions. The microcontroller and radio should be protected from direct exposure to heat and moisture. Temperature sensors should also be protected from solar radiation and other sources of heat that would interfere with accurate temperature readings. Cases and radiation shields will further increase the cost of implementation but can be made economically using 3D printing technology. Resistive soil moisture sensors are susceptible to corrosion because they cause electrolysis by nature. Corrosion can be mitigated by only powering the sensors when needed; corrosion can be avoided altogether by using capacitive soil moisture sensors instead.

Chapter 7: Conclusion

Sustainability in agriculture is becoming more important every day. As Earth's population increases, more resources are needed. And while food is in high demand, a significant amount of food is being wasted because of poor – but manageable – crop conditions. Wireless sensor networks can help increase sustainability in agriculture by providing farmers with usable data, but there are many challenges to overcome before the investment is justifiable. Agricultural fields are typically large, and they require many components to install a wireless sensor network with dense sampling coverage. As hardware is generally expensive, the cost to implement such a network is high. To make matters worse, wireless components require batteries which need periodic replacements. This research project focused on two main areas for improvement: decreasing implementation costs and decreasing power consumption. Recent advances in hardware technologies have introduced opportunities to achieve both these goals. The Raspberry Pi single board computers are inexpensive yet still powerful enough to serve as the central gateway and webserver. The emergence of open-source hardware and software have made microcontrollers and sensors easier to purchase and program. 3D printing technologies remove the need to outsource for manufactured cases and shields. With power consumption in mind, LoRa radios – which operate on unlicensed low frequency radio bands – provide an inexpensive and power-efficient data transfer solution which does not require chaining to achieve long transmission distances. The wireless sensor network developed in this project exploits these opportunities and ultimately demonstrates that farmers can participate in precision agriculture without spending too much time or money. This project works towards improved sustainability in agriculture by making wireless sensor networks easier to acquire, setup, and maintain. I implore researchers to further improve this research, and I encourage farmers to participate by demonstrating the system provides value.

Bibliography

- [1] Wikipedia, "Precision Agriculture," 17 August 2020. [Online]. Available: https://en.wikipedia.org/wiki/Precision_agriculture.
- [2] T. Ojha, S. Misra and N. S. Raghuwanshi, "Wireless sensor networks for agriculture: The state-of-the-art in practice," *Computers and Electronics in Agriculture*, vol. 118, pp. 66-84, 2015.
- [3] Electronics Notes, "What is a Microcontroller MCU: embedded systems," [Online]. Available: <https://www.electronics-notes.com/articles/digital-embedded-processing/embedded-systems/what-is-embedded-microcontroller-mcu.php>. [Accessed 1 September 2020].
- [4] R. Keim, "Understanding Spread Spectrum Modulation in RF Systems," 3 February 2017. [Online]. Available: <https://www.allaboutcircuits.com/technical-articles/understanding-spread-spectrum-modulation-rf-systems/>.
- [5] Semtech, "Why LoRa?," [Online]. Available: <https://www.semtech.com/lora/why-lora>. [Accessed 1 September 2020].
- [6] i-SCOOP, "LoRa and LoRaWAN: the technologies, ecosystems, use cases and market," [Online]. Available: <https://www.i-scoop.eu/internet-of-things-guide/lpwan/iot-network-lora-lorawan/>. [Accessed 1 September 2020].
- [7] B. Ray, "Licensed Vs. Unlicensed Spectrum: What's The Difference?," 13 January 2020. [Online]. Available: <https://www.iotacommunications.com/blog/licensed-vs-unlicensed-spectrum/>.
- [8] L. Burgess, "Inexpensive Low Data Rate Links for the Internet of Things," 2020. [Online]. Available: <https://www.volersystems.com/wearable-devices/inexpensive-low-data-rate-links-for-the-internet-of-things/>. [Accessed 1 September 2020].
- [9] 3GLTEinfo, "LoRa Tutorial – What is LoRa Wireless for IoT? LoRaWAN Frequency Bands," [Online]. Available: <http://www.3glteinfo.com/lora/lorawan-frequency-bands/>. [Accessed 1 September 2020].
- [10] Adafruit, "Adafruit Feather M0 Basic Proto - ATSAMD21 Cortex M0," 2020. [Online]. Available: <https://www.adafruit.com/product/2772>.

- [11] lady ada, "Adafruit Feather M0 Radio with LoRa Radio Module," 8 June 2016. [Online]. Available: <https://learn.adafruit.com/adafruit-feather-m0-radio-with-lora-radio-module>.
- [12] Government of Canada, "Canadian Table of Frequency Allocations," 2018. [Online]. Available: <https://www.ic.gc.ca/eic/site/smt-gst.nsf/eng/sf10759.html>.
- [13] lady ada, "Li-Ion & LiPoly Batteries," 29 July 2012. [Online]. Available: <https://learn.adafruit.com/li-ion-and-lipoly-batteries/voltages>.
- [14] lady ada, "Adafruit Feather M0 Radio with LoRa Radio Module - Power Management," 8 June 2016. [Online]. Available: <https://learn.adafruit.com/adafruit-feather-m0-radio-with-lora-radio-module/power-management>.
- [15] Raspberry Pi, "Raspberry Pi 3 Model B+," [Online]. Available: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/>.
- [16] RobotShop, "LoRa/GPS Long Range Transceiver HAT 915 MHz (North America)," 2020. [Online]. Available: <https://www.robotshop.com/en/lora-gps-long-range-transceiver-hat-915-mhz-north-america.html>.
- [17] MQTT, "MQTT: The Standard for IoT Messaging," [Online]. Available: <https://mqtt.org/>. [Accessed 1 September 2020].
- [18] S. Henry-Stocker, "How the Linux screen tool can save your tasks – and your sanity – if SSH is interrupted," 30 September 2019. [Online]. Available: <https://www.networkworld.com/article/3441777/how-the-linux-screen-tool-can-save-your-tasks-and-your-sanity-if-ssh-is-interrupted.html>.
- [19] M. McCauley, "RadioHead - RH_RF95.h," 2014. [Online]. Available: https://www.airspayce.com/mikem/arduino/RadioHead/RH__RF95_8h_source.html.
- [20] Adafruit, "Adalogger FeatherWing - RTC + SD Add-on For All Feather Boards," 2020. [Online]. Available: <https://www.adafruit.com/product/2922>.