University of Toronto
The Edward S. Rogers Sr. Department of
Electrical and Computer Engineering
ECE496Y Design Project Course

Group Final Report

Team#	260	Date	March 30th, 2021

Project Title	Designing a rainwater monitoring system to combat water scarcity in slums of Mexico City	
Supervisor	Jorg Liebeherr	
Administrator	Khoman Phang	
Section	2	
Students	Jeremy Lee Justin Leung Yongrui Zhang	jeremyjy.lee@mail.utoronto.ca justinpc.leung@mail.utoronto.ca yr.zhang@mail.utoronto.ca

Group Final Report Attribution Table

This table should be filled out to accurately reflect who contributed to each section of the report and what they contributed. Provide a **column** for each student, a **row** for each major section of the report, and the appropriate codes (e.g. 'RD, MR') in each of the necessary **cells** in the table. You may expand the table, inserting rows as needed, but you should not require more than two pages. The original completed and signed form must be included in the hardcopies of the final report. Please make a copy of it for your own reference.

Section	**Student Names**			
	Justin Leung	Jeremy Lee	Yongrui Zhang	
Group Highlights	ET	RD,ET	ET	
Individual Contributions	RD,ET	RD,ET	RD	
Acknowledgements	ET	RD	ET	
Executive Summary	ET	ET	RD	
Introduction	ET	RD,ET	RD	
System Level Overview	RD,ET	ET	RD	
Module Level Descriptions	RD,ET	ET	RD	
Assessment of Final Design	ET	RD	ET	
Testing and Verification	RD,ET	RD	RD	
Summary and Conclusions	ET	RD,ET	RD	
References		RD,ET		
Appendix A	ET		RD	
Appendix B		RD		
All	FP	FP	FP	

Abbreviation Codes:

Fill in abbreviations for roles for each of the required content elements. You do not have to fill in every cell. The "**All**" row refers to the complete report and should indicate who was responsible for the final compilation and final read through of the completed document.

RS – responsible for research of information
RD – wrote the first draft
MR – responsible for major revision
ET – edited for grammar, spelling, and expression
OR – other
"All" row abbreviations:
- FP – final read through of complete document for flow and consistency
- CM – responsible for compiling the elements into the complete document
- OR - other

If you put OR (other) in a cell please put it in as OR1, OR2, etc. Explain briefly below the role referred to:
OR1: enter brief description here
OR2: enter brief description here

Signatures

By signing below, you verify that you have read the attribution table and agree that it accurately reflects your contribution to this document.

Name Justin Leung Signature Justin Leung Date: March 30th, 2021

Name Jeremy Lee Signature Jeremy Lee Date: March 30th, 2021

Name Yongrui Zhang Signature Yongrui Zhang Date: March 30th, 2021

Voluntary Document Release Consent Form[1]

To all ECE496 students:

To better help future students, we would like to provide examples that are drawn from excerpts of past student reports. The examples will be used to illustrate general communication principles as well as how the document guidelines can be applied to a variety of categories of design projects (e.g. electronics, computer, software, networking, research).

Any material chosen for the examples will be altered so that all names are removed. In addition, where possible, much of the technical details will also be removed so that the structure or presentation style are highlighted rather than the original technical content. These examples will be made available to students on the course website, and in general may be accessible by the public. The original reports will not be released but will be accessible only to the course instructors and administrative staff.

Participation is completely voluntary and students may refuse to participate or may withdraw their permission at any time. Reports will only be used with the signed consent of all team members. Participating will have no influence on the grading of your work and there is no penalty for not taking part.

If your group agrees to take part, please have all members sign the bottom of this form. The original completed and signed form should be included in the hardcopies of the final report.

Sincerely,
Khoman Phang
Phil Anderson
ECE496Y Course Coordinators

Consent Statement

We verify that we have read the above letter and are giving permission for the ECE496 course coordinator to use our reports as outlined above.

Team #: 260
Project Title: Designing a rainwater monitoring system to combat water scarcity in slums of Mexico City

Supervisor: Jorg Liebeherr Administrator: Khoman Phang

Name	Justin Leung	Signature	Justin Leung	Date:	March 30th, 2021
Name	Jeremy Lee	Signature	Jeremy Lee	Date:	March 30th, 2021
Name	Yongrui Zhang	Signature	Yongrui Zhang	Date:	March 30th, 2021

[1] This form will be detached from the hardcopy of the final report. Please make sure you have nothing printed on the back page.

Group Highlights (Author: Jeremy Lee)

All hardware, software, and testing components of the design have been completed.

In accordance with our requirements and design, a microprocessor with a LoRa module, rain gauge, real-time clock (RTC), SD card module, battery, and antenna was assembled and integrated. A total of 3 nodes and 1 gateway prototypes were created and operate cohesively.

Using the CottonCandy protocol, we generated a functional network of microcontrollers. Within our network, nodes collect rainfall data from each rain gauge, and periodically transmits the data through the network to a designated gateway node, which stores said data onto an SD card.

When an operator uploads data from the SD card to a website which our team created, the raw CSV file is parsed into an SQLAlchemy database. The data is then visualized in a tabular or graphical version. We also utilize a map to help identify the location of each node.

Justin implemented the rain gauge component of the project, Yongrui implemented the RTC component, and Jeremy implemented the SD card component. Once these individual modules were completed and tested, Justin integrated them into the full system that we have now, and subsequently performed a series of tests to measure the effectiveness of our design. Justin also determined and tested a suitable battery for the system. Meanwhile, Yongrui and Jeremy designed and launched the web application.

There were several challenges which the team faced. Team members were located in different regions across the country, which was challenging considering hardware was a large component of the project. Due to the pandemic and delays in global manufacturing, we experienced shipping delays that hindered the progress of some aspects of the project. Additionally, the team originally intended to use an Amazon EC2 server, however after experiencing network issues, transitioned to a Heroku Platform server.

Individual Contributions

Individual Contributions - Justin Leung (Author: Justin Leung)

As team lead, Justin was responsible for laying out the project plan and approach. At the start of the project, Justin conducted preliminary research on the hardware components necessary in the project and determined if it was possible to integrate them into the system. Justin proposed to use a tipping bucket rain gauge and a SD card module to facilitate rainwater measurements and to transport rain data to a device with internet connection respectively. An external RTC was also determined to be necessary to lower power consumption of the microcontrollers running the CottonCandy library. In order for the team to work most efficiently, Justin decided that the team had to split up the 3 components (rain gauge, RTC and SD card module) among the members so that the team could work in parallel. This works with minimal dependencies as each member can individually integrate the hardware component with the microcontroller and CottonCandy library, to demonstrate functionality.

In the rainwater monitoring system, Justin was responsible for many of the hardware related tasks. Justin was responsible for selecting a suitable rain gauge and integrating that with a microcontroller and the CottonCandy library, which involves software support. Once each component (rain gauge, RTC and SD card module) had proven to function as expected, Justin was tasked with integrating all 3 hardware components together to create a single node/gateway (4 total). In addition to integrating the supporting software of the separate devices, Justin was also responsible for measuring the power consumption of our devices, determining a battery suitable for our application, and testing the hardware system.

Individual Contributions - Jeremy Lee (Author: Jeremy Lee)

Jeremy was responsible for designing and integrating the storage method for uploading the remote data onto the cloud. For this area of the project, he researched and concluded that we should use a microSD card rather than a 3G module, as there were feasibility concerns due to the potential lack of 3G infrastructure in the poorer areas of Mexico City. He also concluded that since it was more feasible to use an Adafruit microcontroller considering other components of the project, the Adalogger Featherwing SD component would be the optimal module after researching it's documentation. After these decisions were made, he participated in sourcing the most financially viable manufacturers, constructing the hardware configuration, and writing a portion of the embedded layer of software for the read-write method. After the completion of his hardware responsibilities, he assisted Yongrui in writing the frontend and backend software to store, upload, and visualize the rainwater dataset. After Yongrui had finished writing the website software, Jeremy was responsible for deploying the website to Heroku web services server. The website instance has been deployed and can be visited at https://rainwater.herokuapp.com/.

Individual Contributions - Yongrui Zhang (Author: Yongrui Zhang)

Yongrui was responsible for integrating the Real Time Clock module into the cottonCandy framework. This initially required research on which RTC to use and comparing the different alternatives across different attributes such as precision, power consumption, and size, to name a few. At the time of performing this work, cottonCandy had not yet provided support for the RTC that was chosen, as well as the main Adafruit LoRa 32u4. Therefore, Yongrui researched the libraries required to make the hardware components compatible and operational. Yongrui also adapted the vanilla cottonCandy code and integrated it with elements from the power saving cottonCandy code, to allow for the use of the RTC and the Adafruit board that was chosen for our project. The adapted code could toggle the main Adafruit board between sleep and transmission states, and also allows for the gateway to set the transmission time. This required some adaptations to the cottonCandy library as the RTC and main chip functionalities differed. This required different deep sleep libraries and RTC libraries. Once the hardware integration was completed with the RTC, Yongrui then began to research the different approaches to the data storage and visualization component of the project in collaboration with Jeremy. They decided upon a cloud service and a backend database and server to make this possible. Yongrui then wrote the code for the backend server and database.

Acknowledgements (Author: Jeremy Lee)

First and foremost, we would like to thank the University of Toronto Center of Global Engineering (CGEN) and Un Techo para mi País (TECHO) for proposing this project. The completion of this undertaking could not have been possible without the assistance of Professor Jorg Liebeherr, Professor Khoman Phang, and Dixin Wu. Thank you for providing us guidance and motivation along the way. We would also like to thank the University of Toronto community for the experiences we have had over the past 4-5 years. The end of one adventure marks the beginning of another. Finally, and most importantly, we would like to thank our parents. Our gratitude goes beyond words.

Executive Summary (Author: Yongrui Zhang)

Mexico City has a population of nearly 22 million people, making it the 5th largest city in the world by population. Unfortunately, due to Mexico City's geographical location, it is not in close proximity to any large body of freshwater, and it also lacks underground reservoirs, making it difficult for all residents of the metropolitan area to receive access to clean water resources. The non-governmental organization Un Techo para mi País (TECHO) decided to address this issue by installing rainwater collection centres to trap water to make it accessible to the millions of people who cannot afford to live in areas that have proper drainage facilities. In order to maximize the amount of rain these collection centres can collect, rainwater data is first required to determine the ideal locations to install said devices. Our team was tasked with the goal of finding a cost effective, and accurate method to measure the rainfall data in Mexico City, utilizing state-of-art computer networking technology, as well as cloud services to bring accurate data to TECHO to aid in this humanitarian endeavor.

There are two major components our team will be designing. First is utilizing a LoRa mesh network with rainwater sensors attached at each node that will upload rainfall data to a gateway. This is the main focus of the project, and the framework of which was provided by a previous capstone team that designed the protocols for LoRa communication, including the setup, recovery, and communication between the nodes and the gateway. The second component will then take this data and store it on the cloud where it can then be accessed via a web application. This part of the project was more open-ended and we chose this design to improve the accessibility and simplicity of access to data and its visualization.

Currently, we have created a working system that includes three nodes and a single gateway, where each node contains a tipping bucket rain gauge. Each node also has an RTC which regulates the power consumption by switching the node on and off. Each part has been tested and proven to transmit the correct data, and can be deployed into the field without requiring any major adjustments to the setup. The backend database and web server are also up and running on the internet, where it can be accessed by anyone in the world.

Overall, this project has been a success since we were able to complete all of the major components that were set out by TECHO, CGEN and Prof. Jorg Liebeherr. We hope that this project can be put to use and help the residents of Mexico City gain access to clean water.

Table of Contents

1.0 Introduction

Encompassed in this report is the goals, design, and testing of the Mexico City Rainwater Monitoring System as part of our final year design project course ECE496. The following section of the report will present the project with a description of the problem statement, the scope of our work, the requirements and constraints that we have worked with to ensure that our project is usable and can be deployed.

1.1 Background and Motivation (Author: Yongrui Zhang)

Mexico City has a population of 22 million with some 20 million of those residents having some form of trouble accessing clean drinking water [1]. This is due to both historical and geographical errors, both of which are outside the scope of the paper and so will not be discussed here. Currently, Mexico City obtains most of its drinking water from underground aquifers, which are depleting quickly, and cannot be replenished at the same rate as their consumption [2]. This has caused the city to literally sink deeper into the ground. The government has tried to resolve the issue by bringing in water from nearby water reserves by truck seen in Figure 1-1. This method has limited effectiveness since it is not scalable to satisfy the needs of a rapidly urbanizing country. Therefore, the NGO TECHO identified a potential solution to this problem by installing rainwater collection stations across the city so that each neighbourhood has access to these water collection stations and can alleviate part of the problem. This proves difficult since these systems are difficult and costly to install, and so the placement of these stations is essential to its success. It is important to determine the locations to place these collection stations in order to maximize the amount of water that can be collected. The current data that is available proves to be old, and quite unreliable. Therefore, TECHO has recruited the help of the Centre of Global Engineering at the University of Toronto to develop a cost-effective solution at gathering the precipitation data across different sites in Mexico City.

Figure 1-1: Mexico City's initiative to bring in water from nearby lakes

1.2 Problem Statement (Author: Yongrui Zhang)

The lack of available freshwater in Mexico City, especially in areas of lower income, requires rainwater harvesting systems to alleviate water scarcity. Since these systems are expensive to install, we must build a LoRa based network that will measure the rainfall amount at each monitoring site that the node is placed.

1.3 Scope of Solution (Author: Yongrui Zhang)

The scope of our project begins with obtaining rain data across a region and ends at providing the user with rain data through a user-friendly interface. The system's enclosure and its ability to withstand environmental conditions are not within the scope of our project. Within the scope of our project, there are critical components needed that we did not develop ourselves. We will use commercial hardware components such as rain gauges, sensors, microcontrollers, power supplies, batteries and wireless modules. The open-source software CottonCandy will be needed to create an arduino based LoRa mesh network and a commercial cloud database system will be used to store and fetch data.

The following are our responsibilities in implementing this rainwater measuring system:

- Determine a suitable rain gauge/sensor
- Integrate the rain gauge/sensor with the microcontroller
- Use CottonCandy library to transfer rain data from multiple nodes to a gateway node
- Implement strategies to lower device power consumption
- Determine suitable battery based on device power consumption and system requirements
- Implement a system in which data from the gateway node can be uploaded online
- Use a cloud based service to store rain data
- Create a simple graphical user interface to view rain data

Given that our system will not have access to an electrical outlet/grid (as seen in requirements), we have decided to limit our scope by choosing batteries as our power source. CGEN has kindly provided us with kits containing sufficient hardware to create a LoRa Mesh Network with microcontrollers. The kit includes an Adafruit feather 32u4, antennas, cables, etc. There are a total of 8 nodes that can be created with the provided kits and the distribution among the team is as follows: 4 for Justin, 2 for Jeremy and 2 for Yongrui. Therefore, the largest network size we can test consists of 3 nodes and 1 gateway as each member is located in a different city in Canada.

As international travel is risky and restricted due to COVID-19, testing the system in its intended environment in Mexico will be out of the scope of this project. Testing will be conducted in Vancouver, Calgary and Toronto as team members are located in those regions.

1.4 Project Goals, Requirements, Constraints (Author: Yongrui Zhang)

In order to assess the degree of success of our implementation, we must set requirements that our system must meet. There are three main categories: functional requirements describe the essential duties of the system, constraints that must be followed during development and operation, and objectives that quantify the effectiveness of our system. We determined these requirements through careful research of industry standards, and also through discussion with stakeholders.

Table 1-1: Requirements Table

ID	Project Requirement	Description
1	Collect Rain Data	Functional Requirement: This system collects rain data across a region
2	Display Rain Data	Functional Requirement: This system displays rain data to the user
3	Number of Rain Monitoring Sites	Constraint: This system shall be able to support at least 4 data collection points
4	Minimum Period Between Maintenance: 2 months	Constraint: This system shall be serviceable for at least 2 months before requiring human maintenance
5	Offline Data Capacity	Constraint: This system shall have data storage capacity of at least 2 months to avoid cases of data overflow. This is due to the 2 month minimum period between maintenance.
6	Legal Carrier Frequency	Constraint: This system shall use a frequency band according to Mexican laws and agreements
7	Offline Environment	Constraint: This system shall be able to collect data in areas where internet or cellular connection is not available.
8	No Access to Electricity	Constraint: This system shall be able to collect data in areas where a power outlet/grid is not available
9	Online Centralized Storage	Constraint: This system shall permanently store data online in order to be used by dependent stakeholders

10	Global Accessibility	Constraint: This system shall be accessible via interface by the dependent stakeholders
11	Maximize Period Between Maintenance	Objective: Maximize uptime to reduce manual maintenance which would disrupt data collection
12	Low Cost	Objective: Minimize cost to justify the installation of rainwater measuring system
13	Simple Data Visualization	Objective: Display visualizations such that the significance of data is clearly understood
14	Language Accessibility	Objective: Support alternative languages such as Spanish for accessibility to non-English speaking users

2.0 Final Design

2.1 System Level Overview

This section starts off by describing the core technologies the project is built on, then a general overview of the project is given.

2.1.1 Core Technology Used (Author: Justin Leung)

As our system is the continuation of a previous capstones team's work, we heavily rely on the CottonCandy library that Dixin Wu, Hongyi Wang and Yizhi Xu. CottonCandy is an Arduino software package for LoRa mesh networking, and we use it to facilitate communication between all our nodes and the gateway [3]. LoRa is a long range, lower power modulation technique the CottonCandy library uses and the LoRa spectrum can be seen in Figure 2-1. Lastly, since CottonCandy is an Arduino software package, we need a microcontroller and a LoRa module to utilize the library. CGEN was able to provide our team with multiple Adafruit Feather 32u4 LoRa Radio (RFM9x) devices and SMA antennas to build a network of devices.

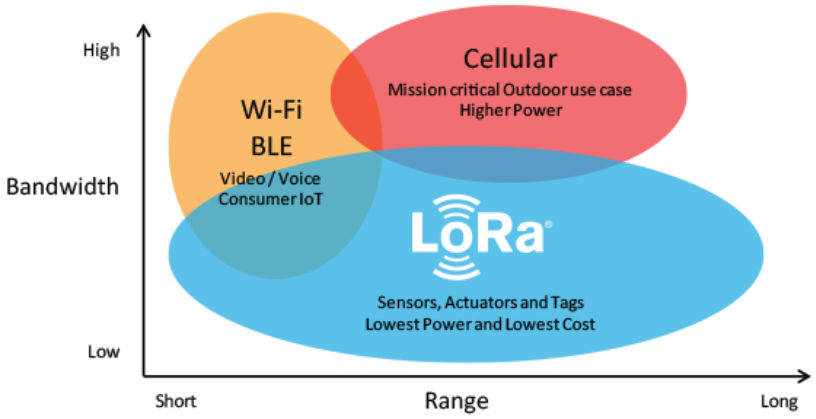


Figure 2-1: LoRa Spectrum [4]

2.1.2 Gateway and Node (Author: Justin Leung)

The nodes and gateway are the devices that form our network. The nodes are responsible for measuring precipitation and transmitting the rain data. The node operates off a battery and to satisfy our system's maintenance requirement, the node must consume minimal power. To meet that requirement, the rain gauge we selected does not consistently draw current and we've adapted techniques that the CottonCandy library uses to lower power consumption to function with our components. The gateway is responsible for sending requests to each node to gather rain data, then saving the received data onto an SD card. The gateway sends a request once a day and requires an external power source.

2.1.3 Data storage and visualization

The data storage and visualization will provide permanent storage of the rainwater data collected on the cloud so that it can be accessed anywhere in the world. Currently, we are using Heroku as the cloud service provider.

2.2 System Block Diagram (Author: Justin Leung)

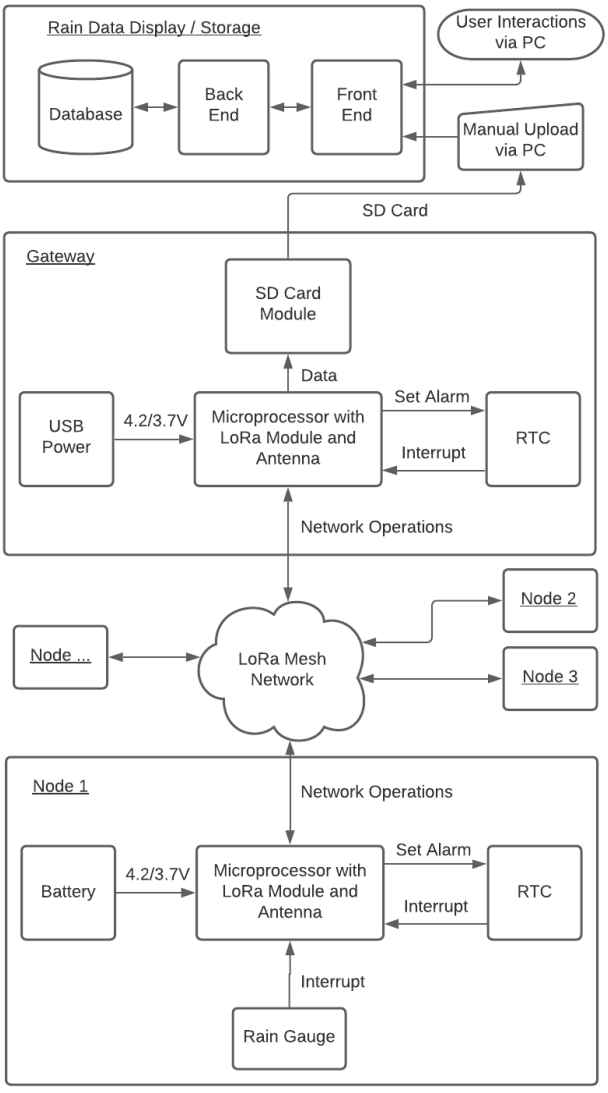


Figure 2-2: System Block Diagram

2.3 Module-level descriptions and designs

2.3.1 Adafruit Feather 32u4 RFM95 LoRa Radio (Author: Justin Leung)

As previously mentioned, our team was supplied with multiple Adafruit Feather 32u4 RFM95 LoRa Radios to form a network of devices. The Adafruit Feather 32u4 houses a ATMega32u4 microprocessor and a LoRa radio that can operate at either 868MHz or 915MHz transmission/reception. Figure 2-3 is the pin out diagram and these pins will be referred to in

later modules. The Adafruit Feather 32u4 serves as a base platform for our system, where we can add modules and components to work towards creating our rain water monitoring system.

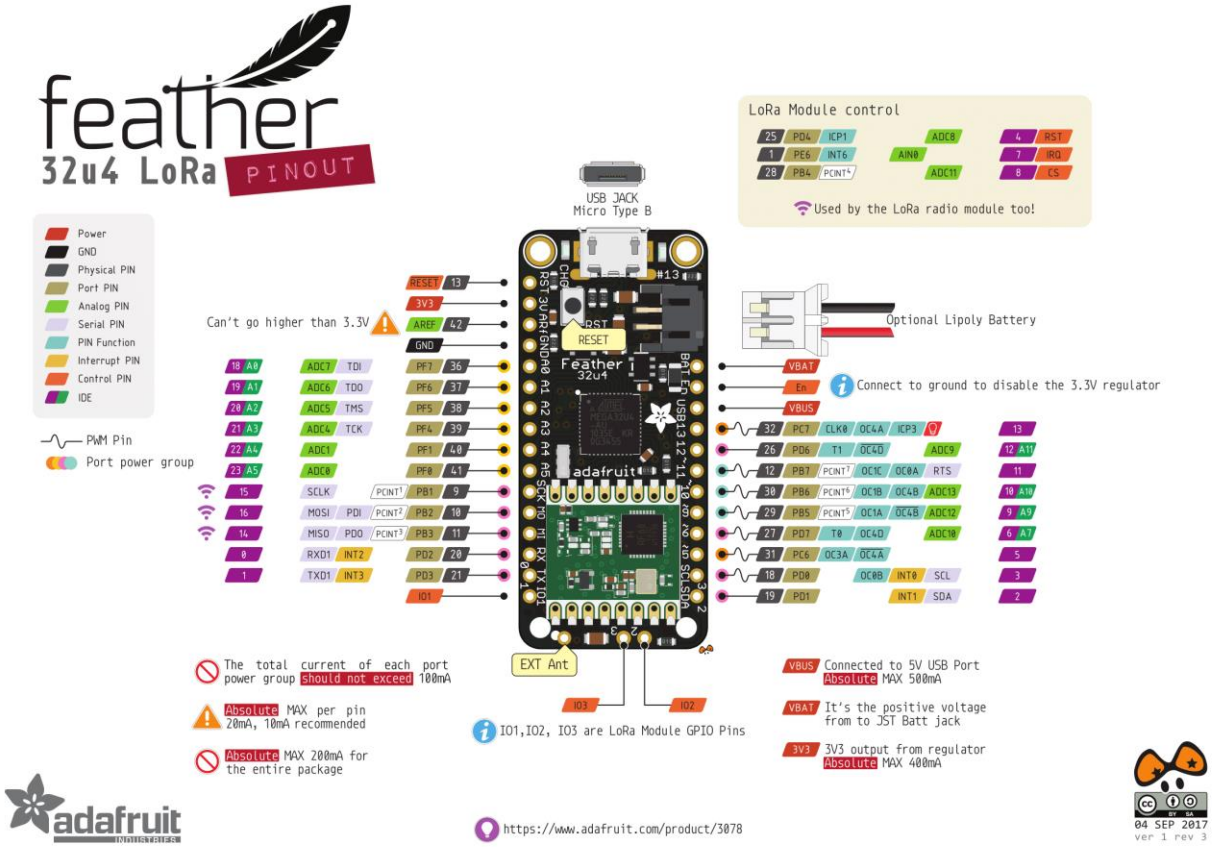


Figure 2-3: Adafruit Feather 32u4 RFM95 LoRa Radio Pinout Diagram [5]

2.3.2 Rain Gauge (Author: Justin Leung)

The rain gauge chosen for our system was a tipping bucket rain gauge as seen in Figure 2-4. The primary reason for selecting this style of rain gauge is because of its low power consumption [6]. Other reasons for selecting the tipping bucket rain gauge is affordability and accessibility, being available on e-commerce websites [7]. Alternatives such as weighing rain gauges or optical rain gauges are more expensive, less accessible, have higher power consumption and cannot be easily integrated with our Adafruit Feather 32u4 platform [8, 9].

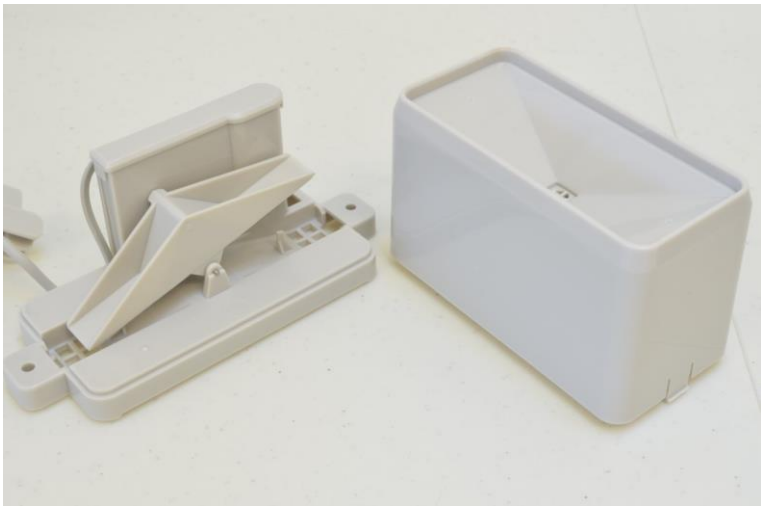


Figure 2-4: Tipping Bucket Rain Gauge [10]

Figure 2-5 is a diagram of the components in a tipping bucket rain gauge; however, the rain gauge in our system does not include the calibration screws. The tipping bucket rain gauge functions by funneling water into one of the two connected buckets and as the bucket reaches its capacity, the pivot point between the buckets enables them to tip. Beside the bucket is a magnetic switch that closes the circuit as the bucket tips. The bucket requires 0.3mm of precipitation in order to tip the bucket and trigger the switch. Once the bucket tips, the water exits the rain gauge via drain holes at the bottom of the gauge. The tipping bucket rain gauge consumes very little power as it does not consistently draw current. It is essentially a switch that remains in an open state and pulses when 0.3mm of precipitation is accumulated.

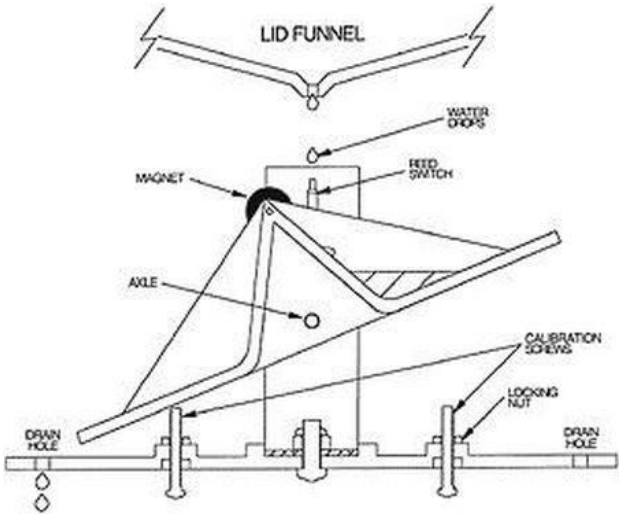


Figure 2-5: Tipping Bucket Rain Gauge Diagram [11]

As seen in Figure 2-6, the rain gauge uses the +3.3V and GND pinouts on the Adafruit Feather 32u4. The tipping bucket rain gauge is setup such that when the bucket tips, the 3.3V output triggers an interrupt at pin #1 (interrupt 3) on the Adafruit Feather 32u4 and the 10k Ω resistor is used as a pull up resistor.

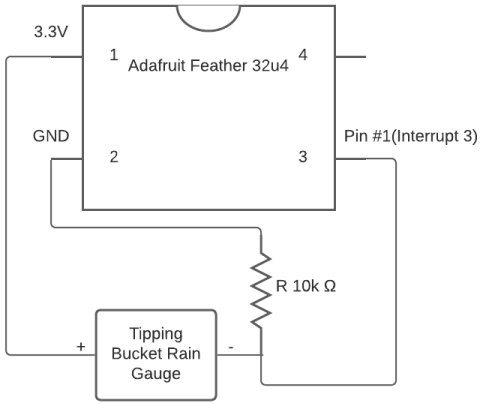


Figure 2-6: Adafruit Feather 32u4 integration with Rain Gauge

The code below is the interrupt service routine used to handle the rain gauge interrupt. The logic is fairly simple as it increments a global counter which is later transmitted over to the gateway. The conditional statement to check if the interrupt_count is less than 3 is required to handle debouncing. We observed that upon a single tip of the bucket, 3 interrupts are consistently triggered and this conditional statement is able to debounce the signal from the rain gauge.

```
81 void rain_gauge_interrupt() {
82
83
84   detachInterrupt(digitalPinToInterrupt(RAIN_GAUGE));
85   detachInterrupt(digitalPinToInterrupt(myRTCInterruptPin));
86
87   // For debouncing
88   interrupt_count = interrupt_count + 1;
89   if(interrupt_count >= 3)
90   {
91     // Do interrupt stuff here
92     Serial.println("Rain gauge pulsed");
93     daily_rain_count = daily_rain_count + 1;
94     interrupt_count = 0;
95   }
96
97   attachInterrupt(digitalPinToInterrupt(RAIN_GAUGE), rain_gauge_interrupt, FALLING);
98   attachInterrupt(digitalPinToInterrupt(myRTCInterruptPin), rtcISR, FALLING);
99
100 }
```

2.3.3 Real Time Clock (Author: Yongrui Zhang)

The Real Time Clock (RTC) is a hardware add-on that was required for this project in order to fulfill the power saving requirements set out in the requirements section and with discussions with stakeholders and our supervisor. We picked the Adalogger FeatherWing, which contains both the RTC module and the microSD module. In this section, the discussion will be focused on the RTC module. The RTC embedded on the chip is the PCF8523, a precision RTC with interrupt, polling, and query capabilities. Figure 2-7 displays the inner detailed circuit diagram of the PCF8523.

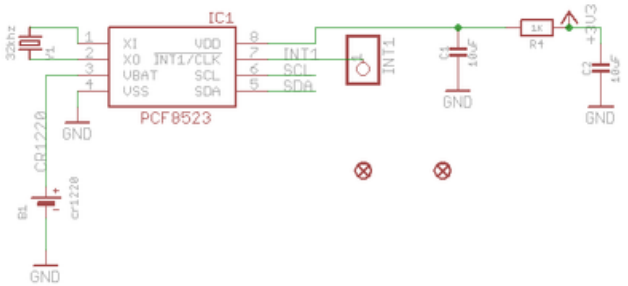


Figure 2-7: PCB of the Adalogger Featherwing

The RTC is a necessary component to facilitate the power saving mode for our system. Since transmission requires many times the power consumption compared to a deep sleeping system, we need some mechanism to transfer the system between the sleep and transmission states. This cannot be done using the internal clock, since deep sleep disables the built in timer on the microcontroller. In addition, our system is made up of many nodes each trying to communicate to the central gateway. This requires that transmission and sleep times be synchronized to preserve the correct operation of the network system. This is made possible by the Adalogger Featherwing's RTC. The RTC provides precise timing that will guarantee synchronization of the transmission and sleep periods of each node in the network.

The Adalogger Featherwing was chosen because it meets the requirements that we had set out at the beginning of the project. The Adalogger Featherwing has interrupt pins for countdown timers, and external power capabilities that are not dependent upon power provided to it by the 3.3V and GND pins. This means that the RTC will be able to track the state of each node independent of whether it is in the sleep or transmission cycle.

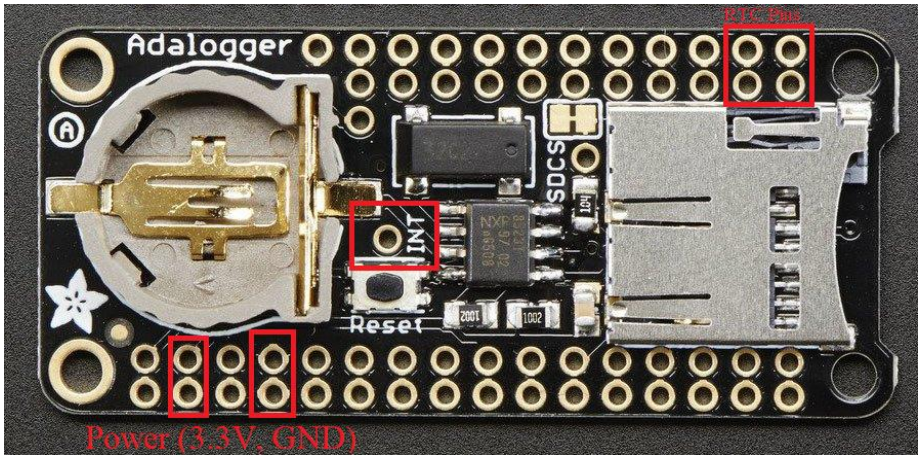


Figure 2-8: Adalogger Featherwing with labels for RTC

This RTC is powered either by a connection to the 3.3V and GND pins to the main microcontroller, or, when power is shut off, it will use the CR1220 battery. The SCL and SDA pins must also be connected as they are used to communicate with the RTC (setting time, setting the countdown timer etc.). Enabling interrupts requires a connection to the INT pin highlighted in Figure 2-8. This pin includes a built in 10k pullup resistor to 3.3V [12]. The INT pin is used for interrupts or used to generate a square wave. These pins are displayed in Figure 2-8.

Our system uses the interrupt functionality provided by the Adalogger Featherwing to toggle the Adafruit main board between sleep and transmission. Since this pin is an open drain, it requires the internal pull up resistor to be enabled to whatever pin the interrupt is connected to. The following code displays how to set a countdown timer and enable the Adalogger Featherwing to provide interrupts when the countdown has finished.

```
// rtc.enableCountdownTimer(PCF8523_FrequencyHour, 24);    // 1 day
// rtc.enableCountdownTimer(PCF8523_FrequencyMinute, 150); // 2.5 hours
rtc.enableCountdownTimer(PCF8523_FrequencySecond, 10);  // 10 seconds
// rtc.enableCountdownTimer(PCF8523_Frequency64Hz, 32);    // 1/2 second
// rtc.enableCountdownTimer(PCF8523_Frequency64Hz, 16);    // 1/4 second
// rtc.enableCountdownTimer(PCF8523_Frequency4kHz, 205);   // 50 milliseconds

attachInterrupt(digitalPinToInterrupt(timerInterruptPin), countdownOver, FALLING);
```

This also requires some setup that is performed as follows:

```
pinMode(timerInterruptPin, INPUT_PULLUP);
```

This initiates a pullup resistor on the specified pin to allow for interrupts. To have a fully functioning RTC that can put the Adafruit microcontroller to deep sleep, we use the standard RTClib [13] and the deep sleep Low-Power [14] library by rocketscream to give us a wide variety of functions that can correctly perform the desired functionality.

2.3.4 MicroSD Card Module (Author: Jeremy Lee)

We chose to use MicroSD technology as the method of data storage for our system. We considered several other methods of storage. Originally, we had considered using a 3G module to connect to a centralized data storage system. After each data collection cycle, our system would be able to directly update the rainwater data on our website in real-time. However, considering the inconsistency and unreliability of internet infrastructure in Mexico City [15], we believed this storage method was not feasible. After consulting our stakeholders, we also considered a bluetooth module. The bluetooth module would be able to communicate with the operator's cellular device, which then the operator could manually upload data to the server. Considering our preference for an offline, low-cost, ease-of-use method, we felt that a microSD card module would be the superior solution. Not only is a microSD card module a cheaper solution [16, 17],

but the microSD card can be used for storage over several months without maintenance and is generally considered more reliable [18, 19].

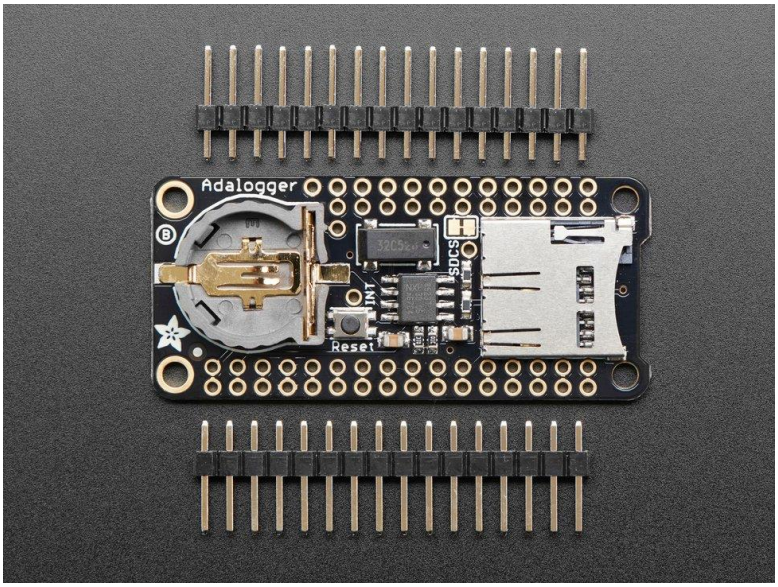


Figure 2-9: Adalogger FeatherWing [20]

After researching MicroSD card modules from several suppliers, we chose the Adalogger FeatherWing - RTC + SD Add-on, as seen in Figure 2-9. Considering our need for an RTC module, this board gave us both modules on a single board. The board is also financially competitive with competitors [21, 22], while coming from the same supplier as our Arduino module, simplifying compatibility issues.

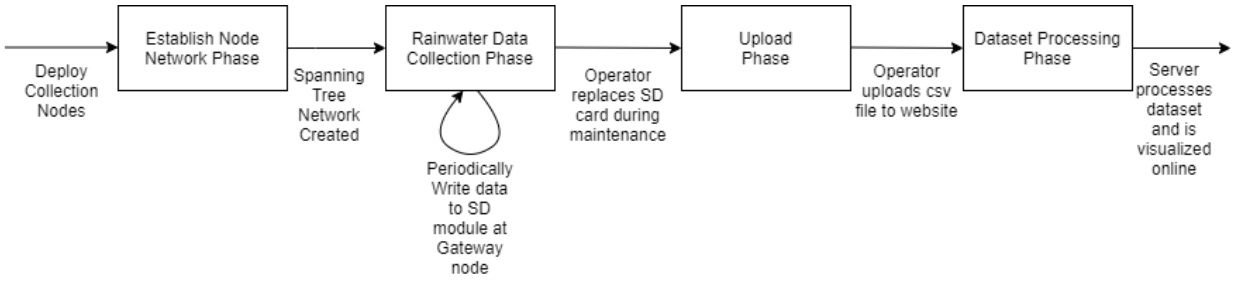


Figure 2-10: Data Storage Process

The routine for saving and uploading is simple, as shown in Figure 2-10. Once a node network is established and the data collection cycle begins, at the end of every cycle, the Arduino module performs a write onto the Adalogger FeatherWing SD module. During maintenance, the operator

can replace the microSD card from the gateway node and upload the data through the website. The server will process the dataset and update the visualization tools.

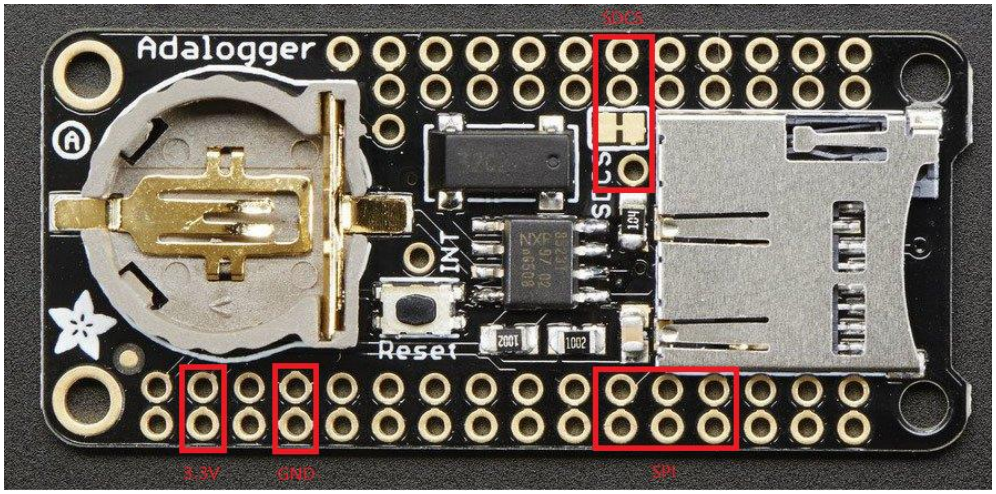


Figure 2-11: MicroSD Card Module Pinout [23]

The pins of the Adalogger related to the microSD card module are labelled in Figure 2-11. The 3.3V and GND pins are used to power the module if a coin cell battery is not connected. The SPI pins from left to right are the SPI Clock (SCK), SPI Microcontroller Out Sensor In (MOSI), and the SPI Microcontroller In Sensor Out (MISO). These pins are used to communicate with the microSD card. The SCK pin is the clock data line from the Arduino to the microSD. The MOSI pin is the data line from the Arduino to the microSD. The MISO pin is the data line from the microSD to the Ardiuno. The SDCS pin is the chip select line.

When the gateway node receives a sensor value from a node, the gateway node writes the passed data to the microSD card. An example of the code for writing to the card is provided below.

```
    if (! rtc.begin()) {
      Serial.println("Couldn't find RTC");
      Serial.flush();
      abort();
    }

    Serial.print("Initializing SD card...");
    if (!SD.begin(chipSelect)) {
      Serial.println("Card failed, or not present");
      while (1);
    }
    Serial.println("card initialized.");

    // Convert the byte array back to a long-type integer
    union LongToBytes myConverter;
    memcpy(myConverter.b, data, len);
    long value = myConverter.l;
    DateTime now = rtc.now();

    File dataFile = SD.open("datalog.csv", FILE_WRITE);

    if (dataFile) { // if the file is available, write to it:
      dataFile.print(value);
      dataFile.print(',');
      dataFile.print(now.day(), DEC);
      dataFile.println();
      dataFile.close();
    }
    else { // if the file isn't open, pop up an error:
      Serial.println("error opening datalog.txt");
    }
  }
```

2.3.5 Node (Author: Yongrui Zhang)

The node is responsible for collecting rainfall data, and then transmitting this data to the gateway. The node requires the Adafruit Feather 32u4, the Adalogger Featherwing, and the Tipping Bucket Rain Gauge for hardware. Figure 2-12 depicts the connections required for assembly of the node and Figure 2-13 shows the assembled node without the battery.

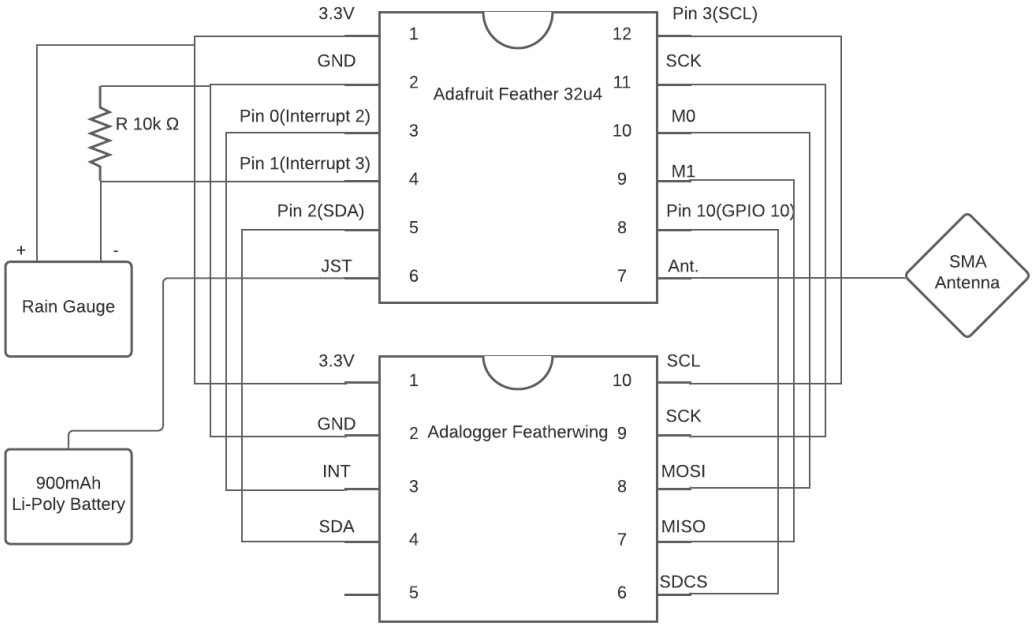


Figure 2-12: Node Diagram

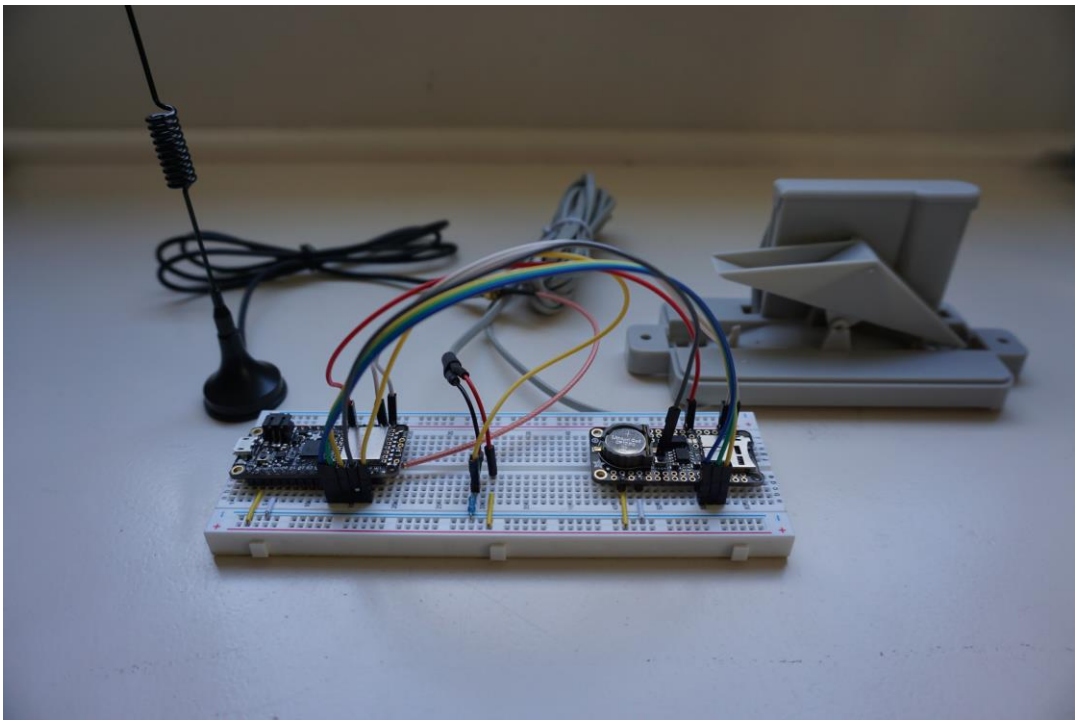


Figure 2-13: Assembled Node without Battery

To achieve low power consumption, we also have an Adalogger Featherwing external attachment that toggles the Adafruit Feather 32u4 between deep sleep and transmission states. In deep sleep, the power consumption is kept at a minimum, maximizing the service time required for each node. The RTC operates similarly to the rain gauge in that it communicates with the Adafruit Feather 32u4 through the use of interrupts. Below is the code for the RTC's interrupt service routine. The if statement prevents the RTC from immediately putting the Adafruit Feather 32u4 to sleep. Each time the rtcISR() is called, the allowReceiving flag is flipped, toggling it between sleep and transmission modes.

```
void rtcISR()
{
    if (immediateInterruptPreventer == 0){
        immediateInterruptPreventer++;
        return;
    }
    delay(100);
    Serial.println(F("Got interrupted"));
    delay(100);
    allowReceiving = !allowReceiving;
}
```

To ensure that the Adafruit Feather 32u4 continues to sleep even after a rain gauge interrupt, we added the following code to put the Adafruit Feather 32u4 back to sleep should it be woken up by the rain gauge and not the RTC.

```
while(!allowReceiving){
    LowPower.powerDown(SLEEP_FOREVER, ADC_OFF, BOD_OFF);
    delay(100);
}
```

At the time of writing the RTC code for our project, the CottonCandy library had not yet expanded support for the Adafruit Feather 32u4. Therefore, we adapted the code so that it could be compatible with the microprocessor and RTC that we are using. The following code is performed prior to the Adafruit Feather 32u4 entering deep sleep mode. Here, we perform some setup steps to ensure that the RTC countdown timer is configured correctly so that it will interrupt once the timer hits zero.

```
detachInterrupt(digitalPinToInterrupt(myRTCInterruptPin));
delay(1000);
Serial.print(F("adalogger_rtc sleeps for: "));
Serial.println(sleepTime);
adalogger_rtc.deconfigureAllTimers();
adalogger_rtc.enableCountdownTimer(PCF8523_FrequencySecond, sleepTime-3);
delay(100);
USBCON |= _BV(FRZCLK);
delay(100);
PLLCSR &= ~_BV(PLLE);
delay(100);
USBCON &= ~_BV(USBE);
delay(100);
attachInterrupt(digitalPinToInterrupt(myRTCInterruptPin), rtcISR, FALLING);
delay(100);
```

The sleep and wakeup times of the node is determined by a gateway request time value that is sent to the node upon communicating with the gateway. This value notifies the node when the gateway will request data next, so that the node can sleep until the next time the gateway requests data.

2.3.6 Gateway (Author: Justin Leung)

As seen in Figure 2-14 and 2-15, the gateway only uses the Adafruit Feather 32u4 and the Adalogger Featherwing. Every 24 hours, the gateway sends a request and gathers rain data from all the nodes. It then stores the rain data of each node, the node address and the date and time onto the microSD card in a CSV file format as seen in figure 2-16. Unlike the node, the gateway uses the original CottonCandy ForwardEngine.cpp and does not sleep. As a result, the gateway does not use the same battery as the node and will need access to a consistent power supply.

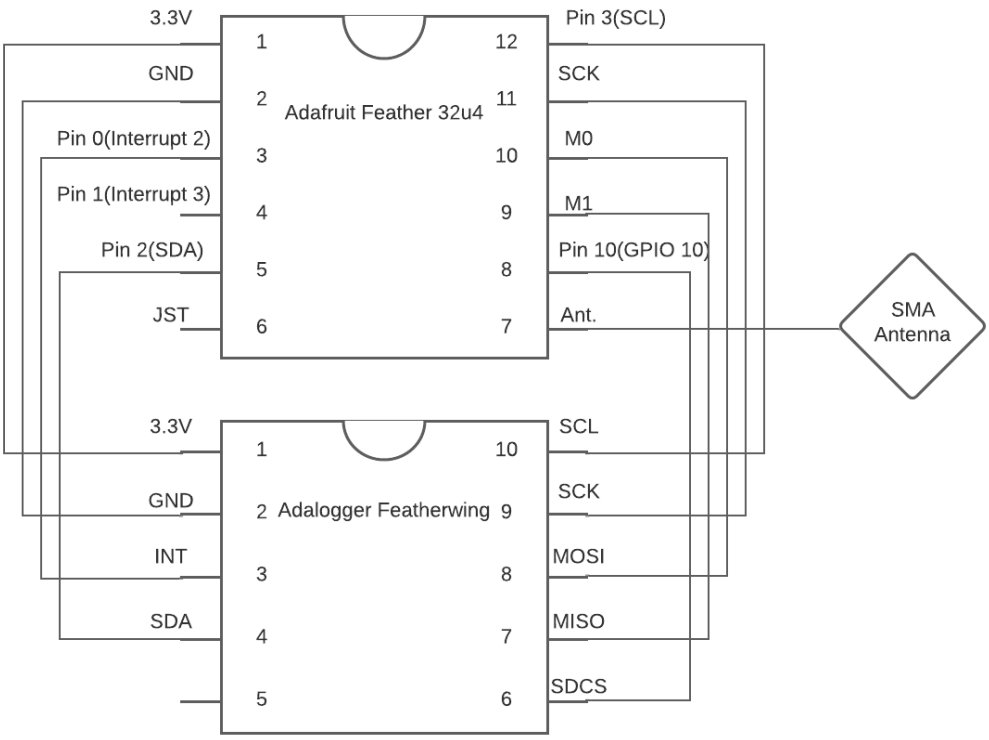


Figure 2-14: Gateway Diagram

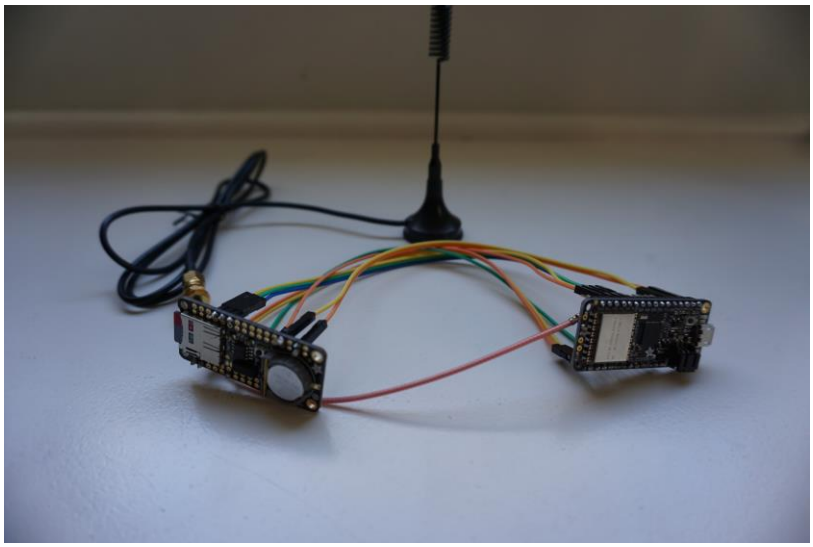


Figure 2-15: Assembled Gateway

A380	0.6	2/6/2021	15:16:31
A180	1.8	2/6/2021	15:17:17
A280	1.8	2/6/2021	15:17:20

Figure 2-16: Data logged on CSV file

2.3.7 Power Consumption and Battery Selection (Author: Justin Leung)

To determine a suitable battery for our node we must measure the current draw of the node in both sleep and communication states. Figure 2-17 is the diagram of the current measurement setup and Figure 2-18 is the actual current measurement setup. We were able to measure a current draw of 0.2mA during the sleep state and a current draw of 22.1-22.7mA during the communication state.

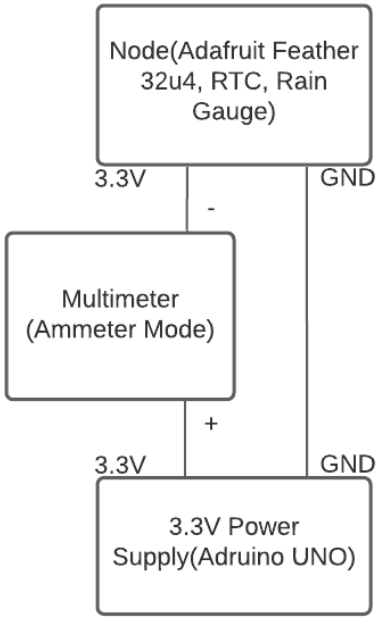


Figure 2-17: Power Measurement Diagram

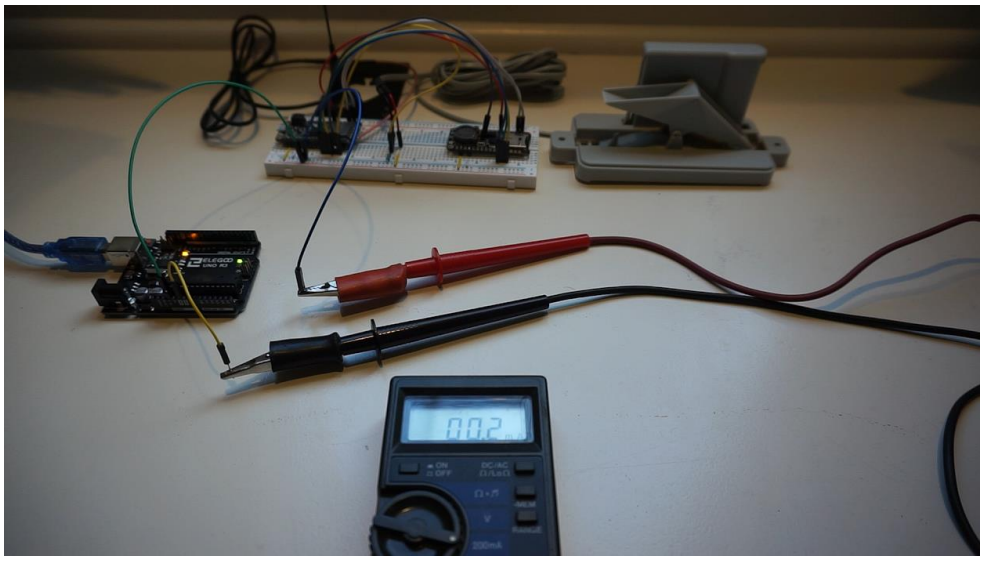


Figure 2-18: Power Measurement Setup

Now we can calculate the total mAh used by the node per day. From the ATMega32u4 datasheet, we know the microcontroller start up time to be 69ms and we can assume the time to increment the rain counter in the ISR to be negligible given the few operations and 16MHz clock speed [24]. With an average daily rainfall of 2.9 mm, we can estimate that the ISR will be run ~10 times a day [25]. That leaves us with an additional 666ms the Adafruit Feather 32u4 spends awake, which can be deemed negligible compared to the 10 minutes it already spends awake in the communication period. That leaves us with a total of 501mAh consumed during the 2 months of operation as seen in the following calculation:

*23.83h * 0.2mA + 0.16h * 22.4mA = 8.35 mAh per day*
*8.35mAh per day * 60days = 501 mAh over 60 days*

The Li-Poly Battery (Figure 2-19) was chosen because of compatibility with the Adafruit Feather 32u4. As seen in Figure 2-20, the device has a JST jack that the battery can simply plug into and the microcontroller can charge the battery if it is connected to USB power [26]. The reason for the 900mAh capacity is that it satisfies the power demands as calculated previously and there was limited selection at the local electronics store.

Figure 2-19: 900mAh Li-Poly Battery

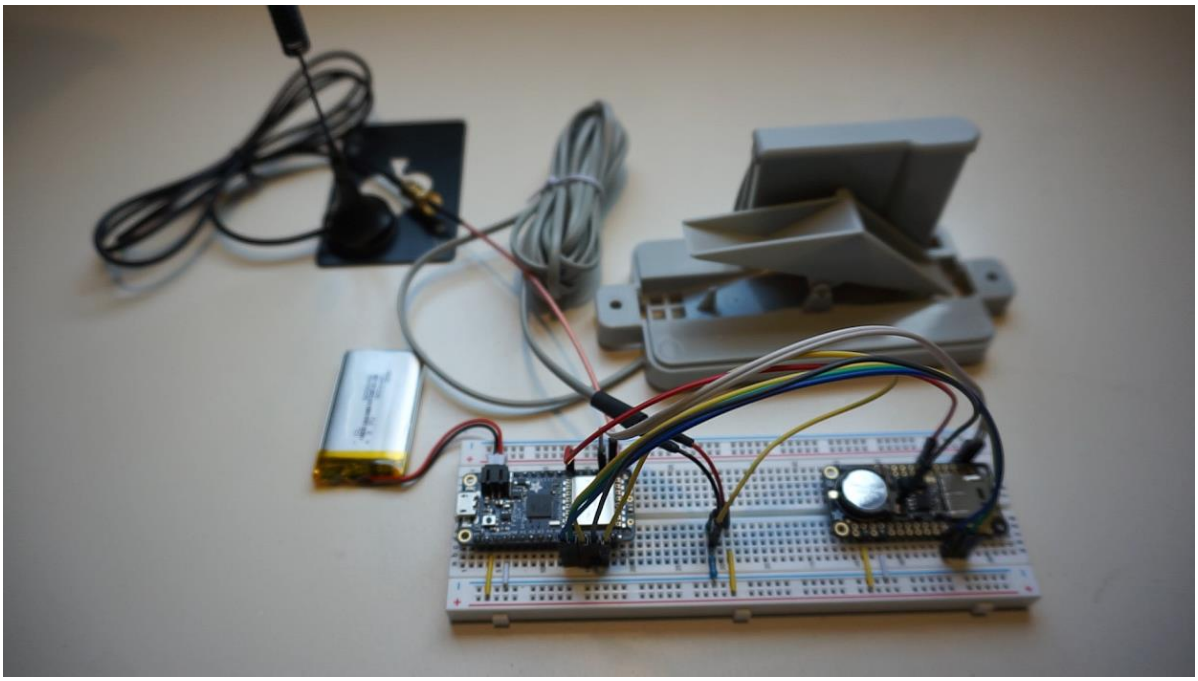


Figure 2-20: Node with Li-Poly Battery

2.3.8 Web Application (Author: Yongrui Zhang)

Data visualization and storage is another major component to this project. For this area, we decided to utilize a web application to achieve data visualization, and a cloud storage service to provide the persistent data storage. The server can be divided into two main technical components, the backend database, which we chose as a SQLite3 database, and the server logic to handle requests from the user, which was chosen to be Python Flask.

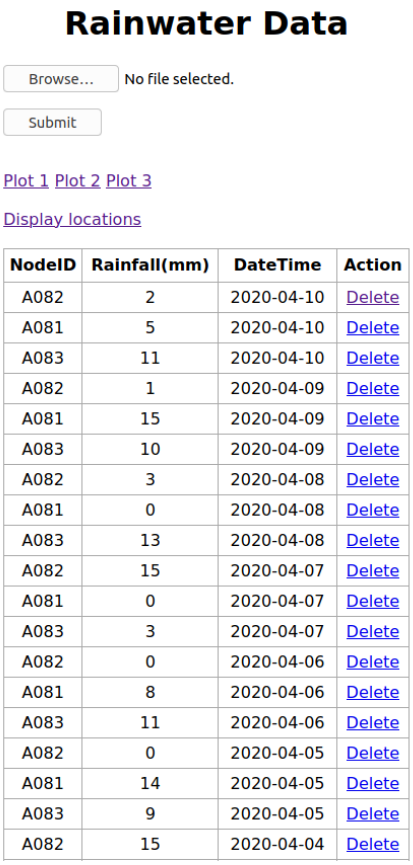
```
class Todo(db.Model):
    EntryID = db.Column(db.Integer, primary_key=True)
    NodeID = db.Column(db.String(200), nullable=False)
    RainData = db.Column(db.Integer)
    Date = db.Column(db.DateTime, default=datetime.utcnow)

    def __repr__(self):
        return '<Task %r>' % self.EntryID

class locations(db.Model):
    NodeID = db.Column(db.Integer, primary_key=True)
    Long = db.Column(db.Float)
    Lat = db.Column(db.Float)
```

This code provides the two tables in our database that contain the required information - the rain data of each node, and the location of the nodes and gateway. These are created in a class in Python Flask, that are initialized upon starting the server which creates a .db file in the main folder of the web application.

Further interactions with the data are performed through our engaging interactive webpage. The homepage adopts a minimalist design style that is easy to read. The user will upload the desired data that is stored on the microSD card, taken from the gateway. The CSV file is then inserted into our database, which will then be displayed onto the homepage in the form of raw data (Figure 2-21).



NodeID	Rainfall(mm)	DateTime	Action
A082	2	2020-04-10	Delete
A081	5	2020-04-10	Delete
A083	11	2020-04-10	Delete
A082	1	2020-04-09	Delete
A081	15	2020-04-09	Delete
A083	10	2020-04-09	Delete
A082	3	2020-04-08	Delete
A081	0	2020-04-08	Delete
A083	13	2020-04-08	Delete
A082	15	2020-04-07	Delete
A081	0	2020-04-07	Delete
A083	3	2020-04-07	Delete
A082	0	2020-04-06	Delete
A081	8	2020-04-06	Delete
A083	11	2020-04-06	Delete
A082	0	2020-04-05	Delete
A081	14	2020-04-05	Delete
A083	9	2020-04-05	Delete
A082	15	2020-04-04	Delete

Figure 2-21: Homepage

Users can then navigate to a multitude of informative pages. Figure 2-22 is generated after clicking on the “Display Locations” link. This page allows the user to see the locations of every part of the system, and also allows the user to change the location of a node or gateway.

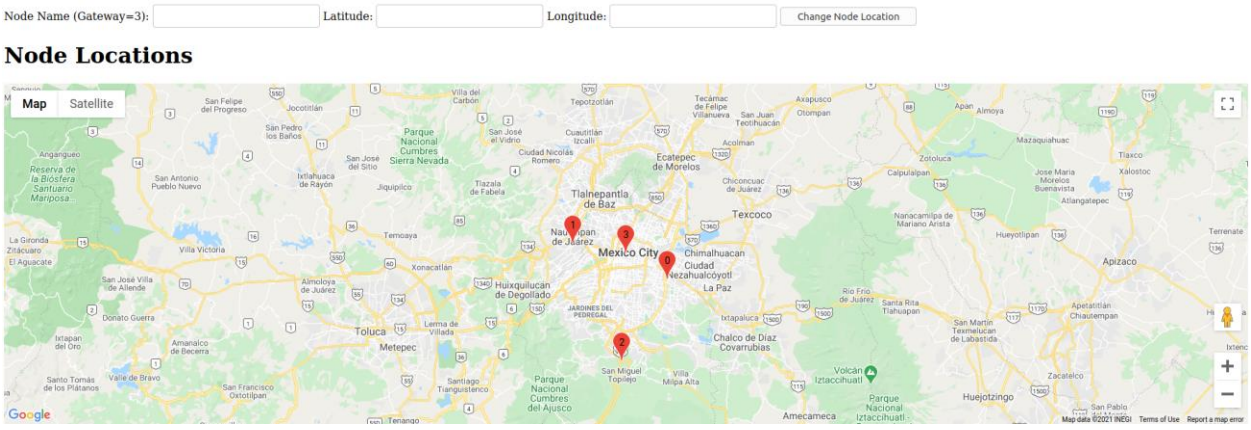


Figure 2-22: Locations of each Node

To aid in the analysis of the rain data, we also provide a graphical representation of the rain data for each node for the past 100 days. You can navigate to these pages when you click on the Plot links on the homepage. Refer to Figure 2-23, Figure 2-24, Figure 2-25.

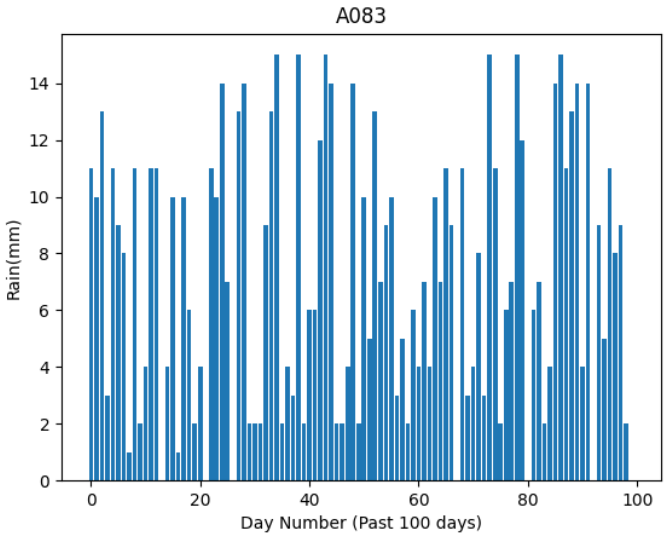


Figure 2-23: Rain data for Node A083

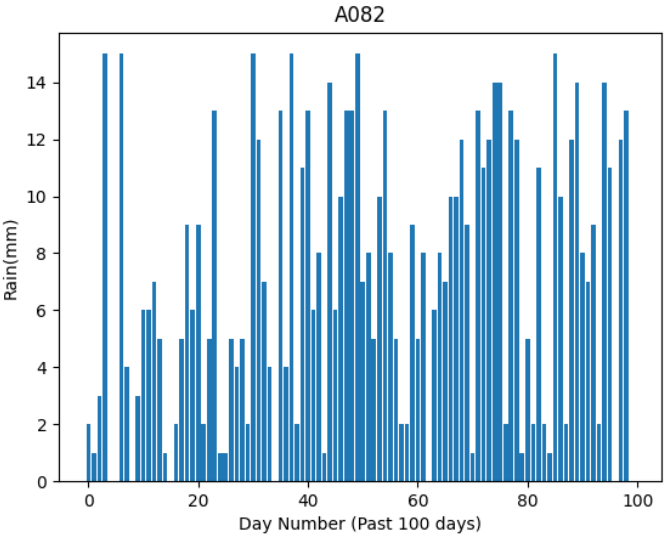


Figure 2-24: Rain data for Node A082

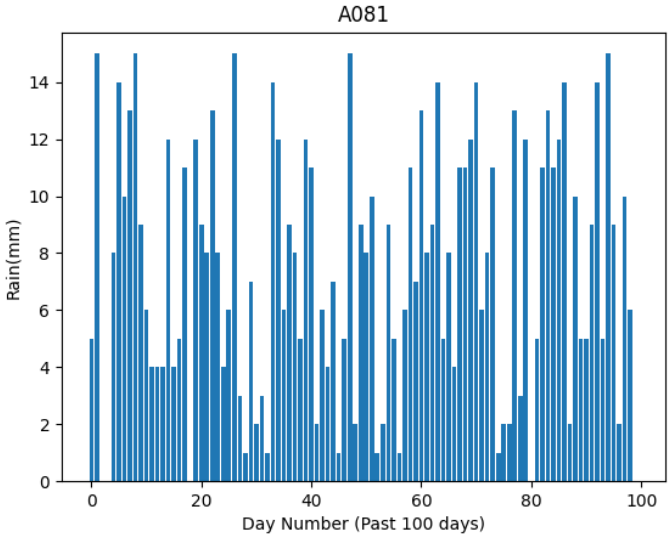


Figure 2-25: Rain data for Node A081

2.3.9 Cloud Service (Author: Jeremy Lee)

We used Heroku as a cloud-based platform server for our Flask application. The Heroku Platform service provides a network in which we can run our app in a virtual container on a reliable runtime environment. We chose to use Heroku due to familiarity within the team, pricing [27], and relative ease-of-use compared to other services [28].

We also used Gunicorn, a Python Web Server Gateway Interface (WSGI) HTTP server, to host static files, handle HTTPS connections, recover from crashes, and effectively scale our website. Gunicorn was the ideal choice since it is frequently used in Heroku Flask Python applications [29].

To deploy the web application, we first created a virtual environment using virtualenv. A virtual environment is a Python environment where the Python interpreter, libraries and scripts can be installed in isolation. We then install all the libraries we used in our web application in our virtual environment. The libraries we installed were SQLite3, Flask, flask-googlemaps, flask-sqlalchemy, gunicorn, matplotlib, numpy, pandas, and SQLAcademy. After we ensure that we can run our web application locally, we create a requirements.txt file to tell Heroku which libraries we need to install in its environment. Inside the requirements.txt file is a list of all the

previously mentioned Python libraries and it's versions. The required libraries and their versions are listed below.

```
Flask==1.1.2
gunicorn==20.0.4
flask-googlemaps==0.4.1
Flask-SQLAlchemy==2.4.4
matplotlib==3.3.4
numpy==1.19.3
pandas==1.1.5
SQLAlchemy==1.3.23
```

Once we are ready to deploy, we can install the Heroku CLI and upload the project by pushing it to Heroku via Git. If the build is successful, the app will be deployed online through the Heroku Platform service. Our website can be viewed here: https://rainwater.herokuapp.com/

2.4 Assessment of Final Design (Author: Jeremy Lee)

Considering the functional requirements, constraints, and objectives of the project, we believe that our design thoroughly fulfills the needs of the project.

We have developed a solution that consists of Adafruit Feather LoRa radios, tipping bucket rain gauges, real-time clocks, and microSD modules. Together, this system collects rain data, meeting Functional Requirement 1 (Collect Rain Data). We have also deployed a Flask website onto a Heroku Platform so that any user can view the rain data with an internet connection, meeting Functional Requirement 2 (Display Rain Data).

Using the CottonCandy protocol, our network is able to support at least 3 data collection points, meeting requirement 3 (Number of Monitoring Sites). Calculating the power usage of our system, we have chosen a 900mAh Li-Poly battery that is compatible with our Adafruit Feather 32u4 and meets the 2 month service-free period of requirement 4 (Minimum Maintenance Period). Requirement 5 (Offline Data Capacity) is met using a 32 GB microSD card, that according to our calculations, avoids data overflow. Through the use of a LoRa radio, we comply with requirement 6 (Legal Carrier Frequency) and 7 (Offline Environment) by only using LoRa

for network communication. Constraint 9 (Online Centralized Storage) and 10 (Global Accessibility) are met by using a server as a means of data storage. The only requirement we were unable to meet was requirement 8 (No access to electricity), due to the fact that our gateway needs a constant power supply.

In our assessment, we have achieved success in our objectives through our design. Battery and storage design was optimized with lifespan and cost considerations, maximizing our Objective 11 (Maximum Maintenance Period) and 12 (Low Cost). Through the use of the website we optimize for Objective 13 (Data Visualization) and 14 (Language Accessibility).

3.0 Testing and Verification

Table 3-1: Verification Table

ID	Requirements	Verification Method	Verification Result and Proof
1	Collect Rain Data	TEST: Simulate rain data by pouring water into the rain sensors and checking that it is logged in all nodes.	PASS. Verified. See table 3-2.
2	Display Rain Data	REVIEW OF DESIGN: Visual inspection to see that correct data is displayed	PASS. Web application to visualize data. See table 3-3.
3	Number of Rain Monitoring Sites	REVIEW OF DESIGN: Deploy at least 3 or more nodes	PASS. Three nodes built for development. See table 3-4.
4	Minimum Battery Lifespan: 2 months	TEST: Verify that battery lifespan of the system lasts for 2 months	PASS. Verified. See table 3-5.
5	Offline Data Capacity	TEST: Verify that the offline microSD storage system is able to store data for a 2 month time span	PASS. Verified. See table 3-6.

6	Legal Carrier Frequency	REVIEW of DESIGN: Ensure frequency band used complies with local laws	PASS. See table 3-7.
7	Offline Environment	REVIEW of DESIGN: Verify system can collect data in areas where internet or cellular connection is not available	PASS. See table 3-8.
8	No Access to Electricity	REVIEW of DESIGN : Verify system functions without access to electricity	FAIL. See table 3-9.
9	Online Centralized Storage	REVIEW of DESIGN: Check that data is accessible online	PASS. Visit https://rainwater.herokuapp.com/. See table 3-10.
10	Global Accessibility	REVIEW of DESIGN: Stakeholders confirm they can access the system and data	PASS. See table 3-11.
11	Maximize Period Between Maintenance	ANALYSIS: Check maximum period system can run without maintenance	107 Days. See table 3-12.
12	Low Cost	TEST: Perform cost analysis and budget approval from TECHO and CGEN	PASS. See table 3-13 and Appendix B.
13	Data Visualization	TEST: Perform some basic studies of key users and ensure they can understand data visualization provided	PASS. See table 3-14.
14	Language Accessibility	TEST: Key stakeholders understand the language being used	PASS. See table 3-15.

3.1 Testing and Verification Details

Table 3-2: Requirement 1 (Author: Justin Leung)

Requirement ID: 1 Collect Rain Data
Test Goal: Verify that rainfall is measured, transferred from the nodes to the gateway and logged onto the microSD card.
Verification Method: TEST: Simulate rain data by pouring water into the rain sensors and checking that it is logged in all nodes.
Test Procedure: First reduce the sleep period and communication period to 5 minutes for testing purposes. In each period, simulate rain data by pouring water into the tipping bucket rain gauge (trigger each rain gauge 3 times). Verify that data is logged onto the microSD card by viewing the serial monitor of the gateway.
Results: As seen in appendix C figure 6-2, all nodes are able to collect rain data(3 tips of rain gauge = 0.90 mm rain) in the communication period and send that data to the gateway. We also verify that in the sleep state, the nodes are able to record rain data. In appendix C figure 6-3, we can see all the nodes go to sleep, and in appendix C figure 6-4, we can see that when they wake up, they transmit the rain data measured during sleep.

Table 3-3: Requirement 2 (Author: Yongrui Zhang)

Requirement ID: 2 Display Rain Data
Verification Method: REVIEW OF DESIGN: Visual inspection to see that correct data is displayed
Test Results: Visually inspected the website; data is displayed correctly. Please refer to Appendix C Figure 6-5.

Table 3-4: Requirement 3 (Author: Justin Leung)

Requirement ID: 3 Number of Rain Monitoring Sites
Verification Method: REVIEW OF DESIGN: Deploy at least 3 or more nodes
Results: The number of nodes in our system can scale as long as CottonCandy can support it. For development, 3 nodes were built for use in our system and are fully functional. See Appendix C Figure 6-6 to see all 3 nodes.

Table 3-5: Requirement 4 (Author: Justin Leung)

Requirement ID: 4 Minimum Battery Lifespan: 2 months
Test Goal: The goal of this test is to ensure the device(node) will be powered for the expected 2-month duration given the 900mAh battery size.
Verification Method: TEST: Verify that battery lifespan of the system lasts for 2 months
Test Procedure: We can estimate the battery life percentage consumed by utilizing the Adafruit Feather 32u4 ability to measure the battery voltage. We will test this by running the node for a period that we estimate will consume 20 percent of the battery's life. Every 15 minutes the node will send the voltage reading to the gateway and we are able to compare the percent drop measured versus the percent drop expected. The percent drop expected is obtained by looking at appendix C, figure 6-8. For testing purposes, we can run the node without sleeping, and since we know the current draw during the communication period to be 22.1-22.7mA, we can estimate that running it for ~4 hours(90mAh/22.4mA) should consume 20 percent of the battery's life.
Results: From the battery voltage measurements seen in appendix C figure 6-7, and the battery discharge graph in appendix C figure 6-8, we can see that our data lines up with the expected discharge curve. Over the 4 hour period the battery dropped by 1.1V, which matches the discharge pattern as seen in the discharge graph.

Table 3-6: Requirement 5 (Author: Justin Leung)

Requirement ID: 5 Offline Data Capacity
Test Goal: Verify that the microSD card(32GB) is capable of storing 2 months' worth of collected rain data.
Verification Method: TEST: Verify that the offline microSD storage system is able to store data for a 2 month time span
Procedure: First, check the initial size of the CSV file, then run the communication period once with only 1 node and 1 gateway in the network. Calculate the additional storage space occupied by a single communication period of one node. Scale the additional storage space required to 3 nodes and verify that 60 communication periods worth of data does not exceed the microSD card's storage capacity.
Results: As seen in appendix C figure 6-9, we can see that one cycle of communication from one node occupies 705 bytes. That means all 3 nodes communicating with the gateway over the 2 month period would take up 126.9 Kbytes, which is significantly less than the capacity of our 32GB microSD card.

Table 3-7: Requirement 6 (Author: Justin Leung)

Requirement ID: 6 Legal Carrier Frequency
Verification Method: REVIEW of DESIGN: Ensure frequency band used complies with local laws
Results: Our LoRa modules can operate at either 868MHz or 915MHz. The 915MHz is used in North America and does not interfere with any frequency bands with prior agreements[30].

Table 3-8: Requirement 7 (Author: Justin Leung)

Requirement ID: 7 Offline Environment
Verification Method: REVIEW of DESIGN: Verify system can collect data in areas where internet or cellular connection is not available.
Results: Our system does not rely on internet or cellular connection for communication between devices. Internet access is only required once rain data is gathered.

Table 3-9: Requirement 8 (Author: Justin Leung)

Requirement ID: 8 No Access to Electricity
Verification Method: REVIEW of DESIGN: Verify system functions without access to electricity
Results: The gateway in our system does not use techniques to lower power consumption such as sleeping. The gateway needs a constant power supply and fails this requirement.

Table 3-10: Requirement 9 (Author: Jeremy Lee)

Requirement ID: 9 Online Centralized Storage
Test Goal: Verify that the operator is able to upload data from the microSD card and the database is globally viewable.
Verification Method: TEST: Upload data collected from the network during a test and check if the data is viewable on a non-local computer.
Results: The dataset from the collection is viewable after uploading it online.

Table 3-11: Requirement 10 (Author: Jeremy Lee)

Requirement ID: 10 Global Accessibility
Verification Method: REVIEW of DESIGN: Stakeholders confirm they can access the system and data
Results: Our web application is hosted on an online cloud service so that the recorded rain data can be viewed around the globe.

Table 3-12: Requirement 11 (Author: Justin Leung)

Requirement ID: 11 Maximize Period Between Maintenance
Verification Method: ANALYSIS: Check maximum period system can run without maintenance
Results: Our 900mAh battery can power our node for around 107 days(900mAh/8.35mAh per day).

Table 3-13: Requirement 12 (Author: Jeremy Lee)

Requirement ID: 12 Low Cost
Test Goal: Verify that the budget for a 3 node 1 gateway system meets the expectations of CGEN and TECHO.
Verification Method: TEST: Run a cost analysis of the total system.
Results: The total cost of a 3 node 1 gateway system was calculated and totaled to $466.67.

Table 3-14: Requirement 13 (Author: Jeremy Lee)

Requirement ID: Data Visualization
Test Goal: Verify that the dataset can be manipulated by the user, graphically visualized, and geographically visualized.
Verification Method: TEST: Visit https://rainwater.herokuapp.com/ and check for data manipulation, graphical, and geographical features.
Results: The requirements for manipulation, graphical, and geographical features have been met.

Table 3-15: Requirement 14 (Author: Yongrui Zhang)

Requirement ID: 14 Language Accessibility
Test Goal: Verify that the website is in a language that can be understood by the locals, and the workers at TECHO
Verification Method: TEST: Visit https://rainwater.herokuapp.com/ and check for the translation capability provided by Google Chrome.
Results: The webpage can be translated to any language via the use of Google online translation as seen in appendix C figure 6-10.

4.0 Summary and Conclusion **(Author: Yongrui Zhang)**

Our capstone project has concluded successfully with our team delivering a product that meets the expectations of our stakeholders as designated by both CGEN and TECHO.

The key conclusions of this project is to highlight the power of using a LoRa module and the CottonCandy protocol to offer a low-cost, offline solution to data transfer networks. This project is not only limited to its application as a rainwater monitoring system. It can be generalized and be applied to a variety of different fields. The basis of the project is a communication system, with a permanent cloud storage system. By attaching different sensors to the node, we can measure different kinds of data.

One application of this technology in a different field can be in measuring traffic data at different intersections. At each traffic light, the city can install a node that monitors the number of cars that pass through the intersection. This data can then be transmitted to the gateway, where the data is then uploaded onto the web, which can then be used by Google Maps to determine the traffic in an area.

Another field that this technology can be used in is the field of seismology. Upon detection of an earthquake, this data is instantly transmitted to the gateway node, where the data can then be analyzed and assessed whether a disaster is imminent.

In terms of future iterations of our project for the application of rainwater measurement, there are still some necessary steps required before it can be successfully deployed on the field. First is education on what the node is and the technology being used. As seen in the news in recent years, the growing number of people subscribing to anti scientific conspiracy theories regarding technological improvements (5G) is concerning. The installation of these nodes in the community could cause distrust and misunderstanding of what the technology does and how it is being used.

The system also requires a packaging that encloses the microprocessors so that it is not damaged every time it rains or is windy. It must also be tamper proof to prevent curious animals from flipping the rain gauge causing the incorrect tallying of rain data information. This case should be able to withstand heavy storms, and have a lock on it, but also allow for rain to pass through so that measurements of precipitation remain accurate.

Our team, with the help of CGEN, Prof. Liebeherr, Dixin Wu, have been able to build a functional framework to measure rainwater in the slums of Mexico City to aid TECHO in the installation for rainwater gathering stations. These nodes can provide the NGO with up to date, and accurate information which can then be used to determine the ideal locations to place these rainwater gathering stations. With minimal modifications to the current state of the system, it can be successfully deployed onto the field and be used to help alleviate the water crisis.

5.0 References

[1] C. Kahn, “Mexico City Keeps Sinking As Its Water Supply Wastes Away,” *NPR*, 14-Sep-2018. [Online]. Available: https://www.npr.org/2018/09/14/647601623/mexico-city-keeps-sinking-as-its-water-supply-wastes-away. [Accessed: 30-Mar-2021].

[2] J. Gutierrez, “Water Scarcity and Supply Challenges in Mexico City's Informal Settlements,” 01-Nov-2019. [Online]. Available: https://penniur.upenn.edu/uploads/media/02_Gutierrez.pdf. [Accessed: 29-Mar-2021].

[3] infernoDison, “infernoDison/cottonCandy,” GitHub. [Online]. Available: https://github.com/infernoDison/cottonCandy/tree/power. [Accessed: 29-Mar-2021].

[4] “What Is LoRa®?,” *Semtech*. [Online]. Available: https://www.semtech.com/lora. [Accessed: 29-Mar-2021].

[5] “STEMMA,” *Adafruit Learning System*. [Online]. Available: https://learn.adafruit.com/assets/46253. [Accessed: 29-Mar-2021].

[6] J. Barani, “Guide to rain gauges: types, sizes, benefits and drawbacks,” *BARANI DESIGN Technologies*, 27-Oct-2020. [Online]. Available: https://www.baranidesign.com/faq-articles/2020/1/20/guide-to-rain-gauges-types-sizes-benefits-and-drawbacks. [Accessed: 30-Mar-2021].

[7] “16.43US $ 20% OFF: WH SP RG Rain Gauge Meteorological Test Rain Gauge Meteorological Equipment Accessories for Misol: Gauges: - AliExpress,” *aliexpress.com*. [Online]. Available: https://www.aliexpress.com/item/4000407106038.html?spm=a2g0s.9042311.0.0.5cc44c4dIomEqK. [Accessed: 30-Mar-2021].

[8] “Hydreon RG-11 Optical Rain Gauge,” *Fondriest Environmental*. [Online]. Available: https://www.fondriest.com/hydreon-rg-11.htm. [Accessed: 30-Mar-2021].

[9] “Weighing Rain Gauge,” *Darrera*, 22-May-2020. [Online]. Available: https://www.darrera.com/wp/en/product/15184-weighing-rain-gauge/. [Accessed: 30-Mar-2021].

[10] “Raspberry Pi Weather Station - Part 1,” *BC Robotics*. [Online]. Available: https://bc-robotics.com/tutorials/raspberry-pi-weather-station-part-1/. [Accessed: 30-Mar-2021].

[11] P. B. arduino engineer, “Arduino Weather Station Part3, Rain,” *Use Arduino for Projects*, 23-Apr-2013. [Online]. Available: https://duino4projects.com/arduino-weather-station-part3-rain/. [Accessed: 30-Mar-2021].
[12] P. B. arduino engineer, “Arduino Weather Station Part3, Rain,” *Use Arduino for Projects*, 23-Apr-2013. [Online]. Available: https://duino4projects.com/arduino-weather-station-part3-rain/. [Accessed: 30-Mar-2021].
[13] Adafruit, “adafruit/RTClib,” *GitHub*. [Online]. Available: https://github.com/adafruit/RTClib. [Accessed: 30-Mar-2021].
[14] Rocketscream, “rocketscream/Low-Power,” *GitHub*. [Online]. Available: https://github.com/rocketscream/Low-Power. [Accessed: 30-Mar-2021].
[15] J. M. M. Montiel, “THE DIGITAL DIVIDE IN MEXICO: A MIRROR OF POVERTY,” *Mexican Law Review*, 26-Oct-2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1870057816300464. [Accessed: 30-Mar-2021].
[16] A. Industries, “Bluefruit LE - Bluetooth Low Energy (BLE 4.0) - nRF8001 Breakout,” *adafruit industries blog RSS*. [Online]. Available: https://www.adafruit.com/product/1697. [Accessed: 30-Mar-2021].
[17] A. Industries, “MicroSD card breakout board+,” *adafruit industries blog RSS*. [Online]. Available: https://www.adafruit.com/product/254. [Accessed: 30-Mar-2021].
[18] J. Harmer, “How Long Do Memory Cards Last?,” *Improve Photography*, 20-Mar-2020. [Online]. Available: https://improvephotography.com/749/memory-cards-sd-cf-replaced-old-reliability/#:~:text=Almost%20all%20modern%20memory%20cards,decades%20without%20having%20any%20problems. [Accessed: 30-Mar-2021].
[19] G. Tyler, “Why does Bluetooth still suck?,” *Business Insider*, 05-Mar-2018. [Online]. Available: https://www.businessinsider.com/why-bluetooth-sucks-bad-problems-issues-disconnects-2018-2. [Accessed: 30-Mar-2021].
[20] A. Industries, “Adalogger FeatherWing - RTC + SD Add-on For All Feather Boards,” *adafruit industries blog RSS*. [Online]. Available: https://www.adafruit.com/product/2922. [Accessed: 30-Mar-2021].
[21] *WYPH Mini Nano V3.0 Module ATmega328P 5V 16MHz CH340G Chip Microcontroller Development Board USB Cable for Arduino (Pre-soldered Nano 3pcs): Amazon.ca: Electronics*.

[Online]. Available: https://www.amazon.ca/WYPH-ATmega328P-Microcontroller-Development-Pre-soldered/dp/B07KC9C6H5/ref=bmx_6?pd_rd_w=KXjUC&pf_rd_p=74914c26-bceb-437c-85a8-93beb003dfd9&pf_rd_r=FE1AFSZFSGHCQJNHYGYH&pd_rd_r=f7243e14-7695-45f6-b401-f027f8fd394e&pd_rd_wg=WXWJn&pd_rd_i=B07KC9C6H5&psc=1. [Accessed: 30-Mar-2021].
[22] "MicroSD Card Module for Arduino," *MicroSD Card Module for Arduino - DFRobot - Evaluation Boards - Expansion Boards, Daughter Cards | Online Catalog | DigiKey Electronics*. [Online]. Available: https://www.digikey.ca/catalog/en/partgroup/microsd-card-module-for-arduino/70502. [Accessed: 30-Mar-2021].
[23] L. Ada, "Adafruit Adalogger FeatherWing," *Adafruit Learning System*. [Online]. Available: https://learn.adafruit.com/adafruit-adalogger-featherwing/pinouts. [Accessed: 30-Mar-2021].
[24] Atmel, " ATmega16U4/ATmega32U4," *www.microchip.com*. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7766-8-bit-AVR-ATmega16U4-32U4_Datasheet.pdf. [Accessed: 29-Mar-2021].
[25] "Meixco City Climate," *Climate*. [Online]. Available: https://en.climate-data.org/north-america/mexico/federal-district/mexico-city-1093/. [Accessed: 30-Mar-2021].
[26] L. Ada, "Adafruit Feather 32u4 with LoRa Radio Module," *Adafruit Learning System*. [Online]. Available: https://learn.adafruit.com/adafruit-feather-32u4-radio-with-lora-radio-module/power-management#. [Accessed: 30-Mar-2021].
[27] "Heroku Pricing," *Heroku*. [Online]. Available: https://www.heroku.com/pricing. [Accessed: 30-Mar-2021].
[28] A. Hebda, "Heroku vs. AWS - Which PaaS Hosting to Choose?," *Blog by Railsware,* 25-Apr-2019. [Online]. Available: https://railsware.com/blog/heroku-vs-aws-which-paas-hosting-to-choose/. [Accessed: 30-Mar-2021].
[29] A. Abdelaal, "Deploying a Flask Application to Heroku," *Stack Abuse*. [Online]. Available: https://stackabuse.com/deploying-a-flask-application-to-heroku/. [Accessed: 30-Mar-2021].
[30] "Mexican Agreements By Frequency," *Federal Communications Commission,* 07-Jul-2017. [Online]. Available: https://www.fcc.gov/mexican-agreements-frequency. [Accessed: 30-Mar-2021].
[31] "STEMMA," *Adafruit Learning System*. [Online]. Available: https://learn.adafruit.com/assets/979. [Accessed: 30-Mar-2021].

6.0 Appendices

Appendix A: Gantt Chart (Author: Yongrui Zhang)

	August	September	October	November	December	January	February	March
Research LoRa	Yongrui, Justin, Jeremy							
Read cottonCandy Code		Yongrui, Justin, Jeremy						
Research required hardware		Yongrui, Justin, Jeremy						
Research required power source		Justin						
Researching Hardware Purchases		Yongrui, Justin, Jeremy						
Assemble Hardware (provided examples)			Yongrui, Justin, Jeremy					
Read manuals for purchased hardware			Yongrui, Justin, Jeremy					
SD Card hardware installation			Jeremy					
SD Card Software			Jeremy					
SD Card testing					Jeremy			
Rain gauge hardware installation			Justin					
Rain gauge software			Justin					
Rain gauge testing					Justin			
RTC hardware installation			Yongrui					
RTC software			Yongrui					
RTC testing					Yongrui			
RTC integration with existing cottonCandy						Yongrui		
Integrate all systems						Justin		
Choose power source							Justin	
Create database from CSV						Yongrui		
Create frontend display						Yongrui, Jeremy		
Deploy Web app to Cloud Service								Jeremy
Full system testing								Justin
Beta Delivery								Justin, Jeremy, Yongrui
Final Delivery								Justin, Jeremy, Yongrui

Figure 6-2: Gantt Chart

Appendix B: Financial Plan (Author: Jeremy Lee)

	1.3 CAD : 1 USD					
	Budget Table (3 module + 1 gatway system)					
			CAD		CAD	
	Consumables	Priority	Cost/Unit	Quantity	Total Cost	Req. Funding
	Adafruit Feather 32u4 RFM95 LoRa Radio	1	45.45	4	$181.80	y
	Adalogger FeatherWing - RTC + SD Module	1	11.64	4	$46.56	y
	WH-SP-RG Meteorological Rain Gauge	1	20.71	3	$62.13	y
	Soldering Wire	1	16.99	1	$16.99	y
	MicroSD Card	1	14.39	1	$14.39	y
	CR1220 3V Battery (RTC Backup)	1	0.96	4	$3.84	y
	3.7V LI-POLY 900mAh Battery	1	13.00	4	$52.00	y
	SMA Antenna	1	9.99	3	$29.97	y
	Additional Wiring	1	8.99	1	$8.99	y
	Heroku Platform Server	1	25/month	2 months	$50.00	y
	Total Consumables				$466.67	
	Total Required Funding				**$466.67**	
	Capital Equipment					
	Soldering Iron	1	40.00	3	$120.00	n
	SD Card Reader	1	20.00	1	$20.00	n
	Electronics Kit (Breadboard, wiring, etc)	1	22.99	3	$68.97	n
	Total Capital Equipment				$208.97	
	Total Required Funding				**$0.00**	

			CAD		CAD	
	Student Labour		Cost/Unit	Hours	Total Cost	Notes
	Student 1		11.06	450	$4,977	Minimum Wage, 15 hours per week for 30 weeks
	Student 2		11.06	450	$4,977	Minimum Wage, 15 hours per week for 30 weeks
	Student 3		11.06	450	$4,977	Minimum Wage, 15 hours per week for 30 weeks
	TOTAL Student Labour (unfunded)				**$14,931**	
	TOTAL Cost of Project				$15,607	
	TOTAL Cost of Required Funding				**$466.67**	
	Funding					
	Students				$146.21	
	Supervisor				$211.77	
	Other				0	
	Request for Design Centre				$108.69	

Appendix C: Testing and Verification Evidence

Requirement ID: 1

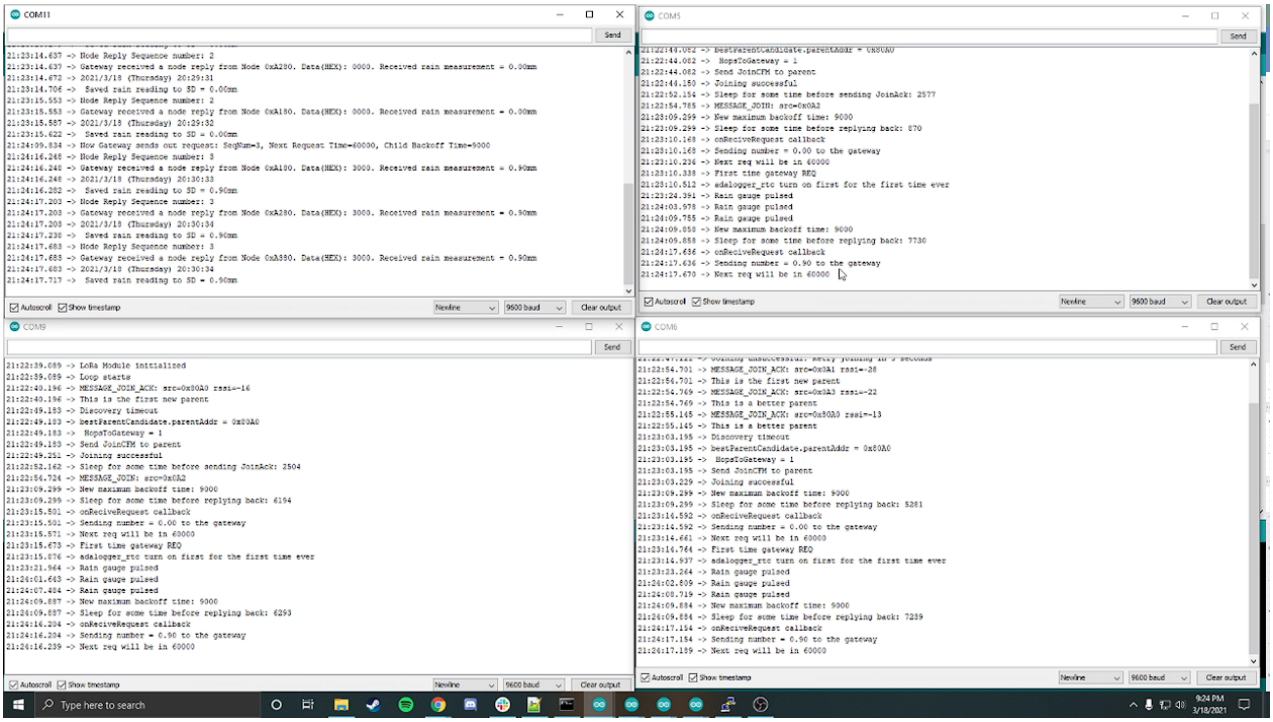


Figure 6-2: Rain data collection in communication period

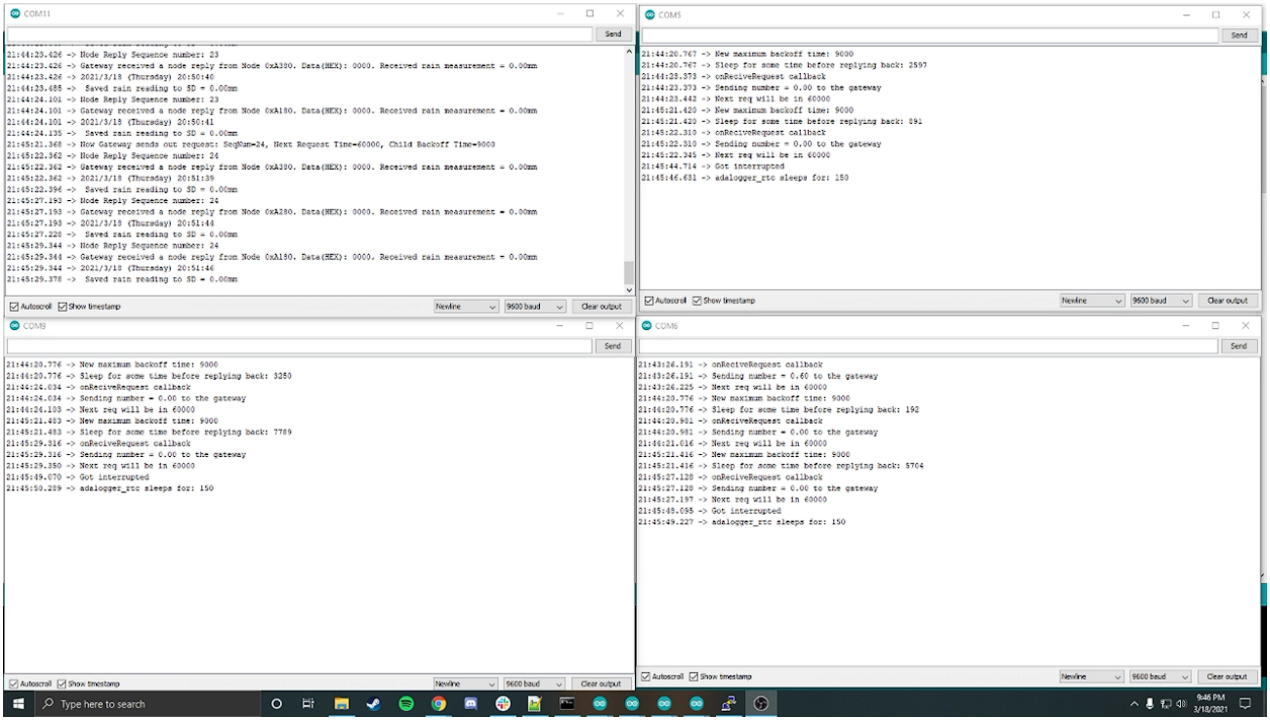


Figure 6-3: All nodes go to sleep

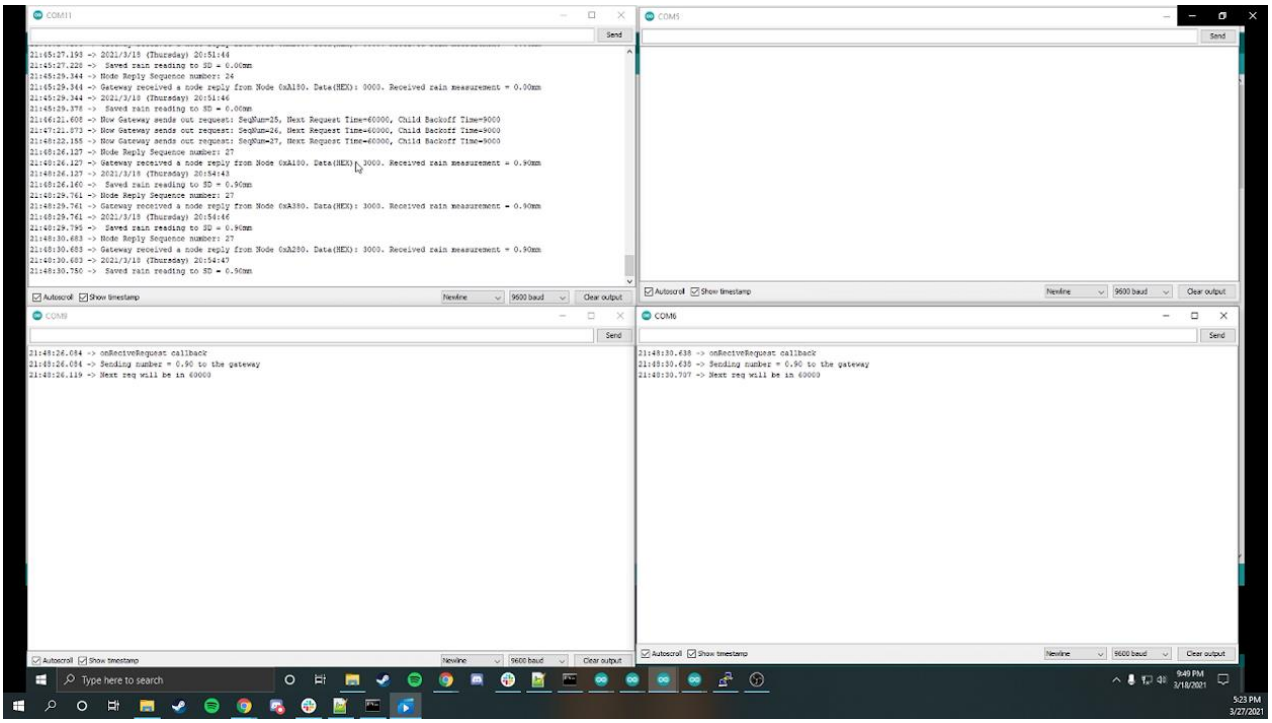
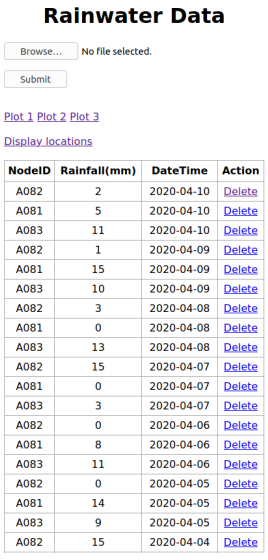


Figure 6-4: All nodes sent rain data gathered in sleep period

Requirement ID: 2



Rainwater Data

Browse... No file selected.

Submit

Plot 1 Plot 2 Plot 3

Display locations

NodeID	Rainfall(mm)	DateTime	Action
A082	2	2020-04-10	Delete
A081	5	2020-04-10	Delete
A083	11	2020-04-10	Delete
A082	1	2020-04-09	Delete
A081	15	2020-04-09	Delete
A083	10	2020-04-09	Delete
A082	3	2020-04-08	Delete
A081	0	2020-04-08	Delete
A083	13	2020-04-08	Delete
A082	15	2020-04-07	Delete
A081	0	2020-04-07	Delete
A083	3	2020-04-07	Delete
A082	0	2020-04-06	Delete
A081	8	2020-04-06	Delete
A083	11	2020-04-06	Delete
A082	0	2020-04-05	Delete
A081	14	2020-04-05	Delete
A083	9	2020-04-05	Delete
A082	15	2020-04-04	Delete

Figure 6-5: Data being displayed correctly - dates and values match what is uploaded by the user.

Requirement ID: 3

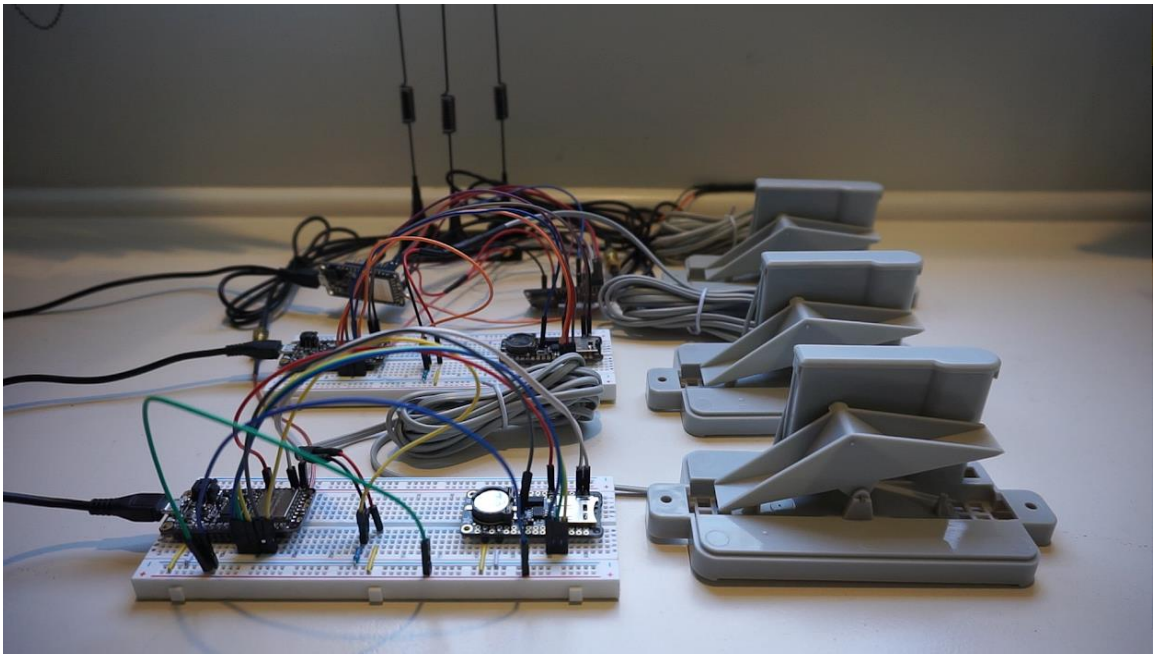


Figure 6-6: All three assembled nodes

Requirement ID: 4

Node Address	Voltage	Date	Time
A180	4.04	3/11/2021	16:01:11
A180	4.02	3/11/2021	16:16:19
A180	4.02	3/11/2021	16:31:25
A180	4.01	3/11/2021	16:46:28
A180	4	3/11/2021	17:01:30
A180	4	3/11/2021	17:16:34
A180	3.99	3/11/2021	17:31:36
A180	3.98	3/11/2021	17:46:41
A180	3.98	3/11/2021	18:01:44
A180	3.97	3/11/2021	18:16:48
A180	3.96	3/11/2021	18:31:51
A180	3.96	3/11/2021	18:46:55
A180	3.95	3/11/2021	19:01:57
A180	3.94	3/11/2021	19:17:00
A180	3.94	3/11/2021	19:32:03
A180	3.93	3/11/2021	19:47:08
A180	3.93	3/11/2021	20:02:12

Figure 6-7: Battery Voltage Measurements

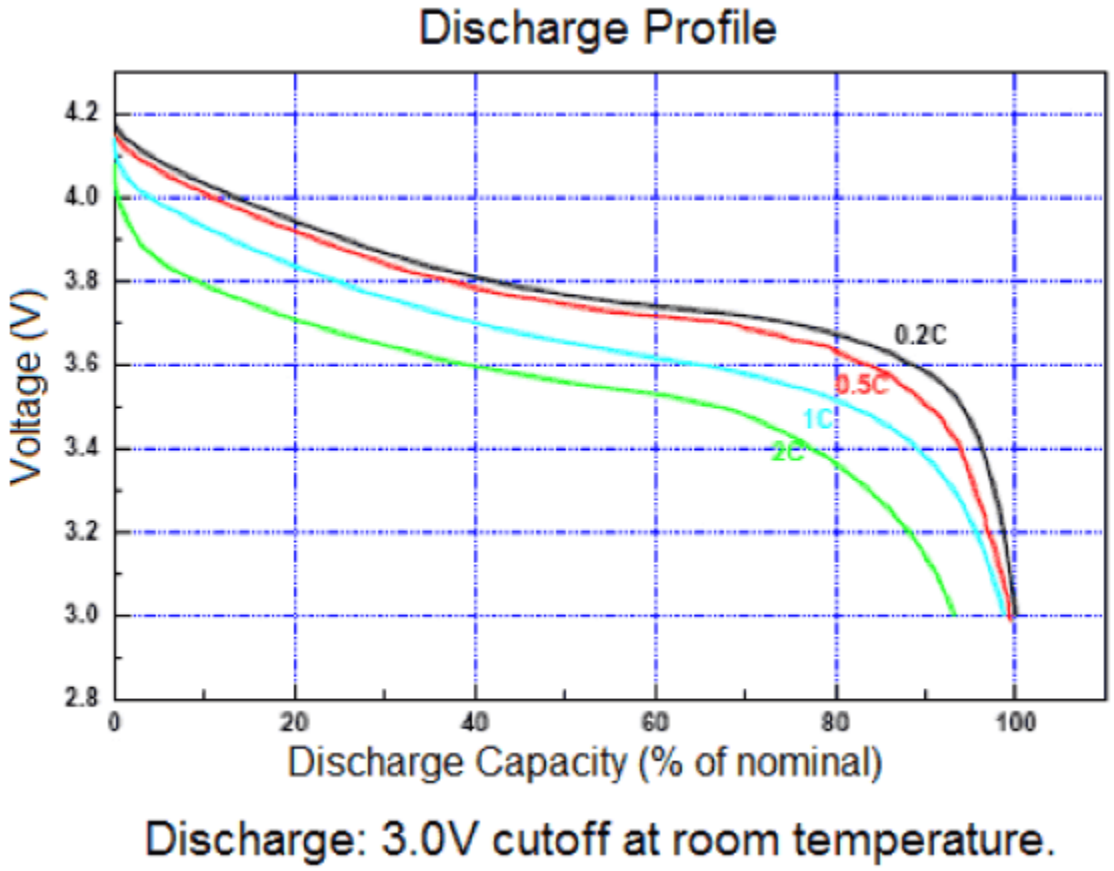


Figure 6-8: Battery Discharge Graph[29]

Requirement ID: 5

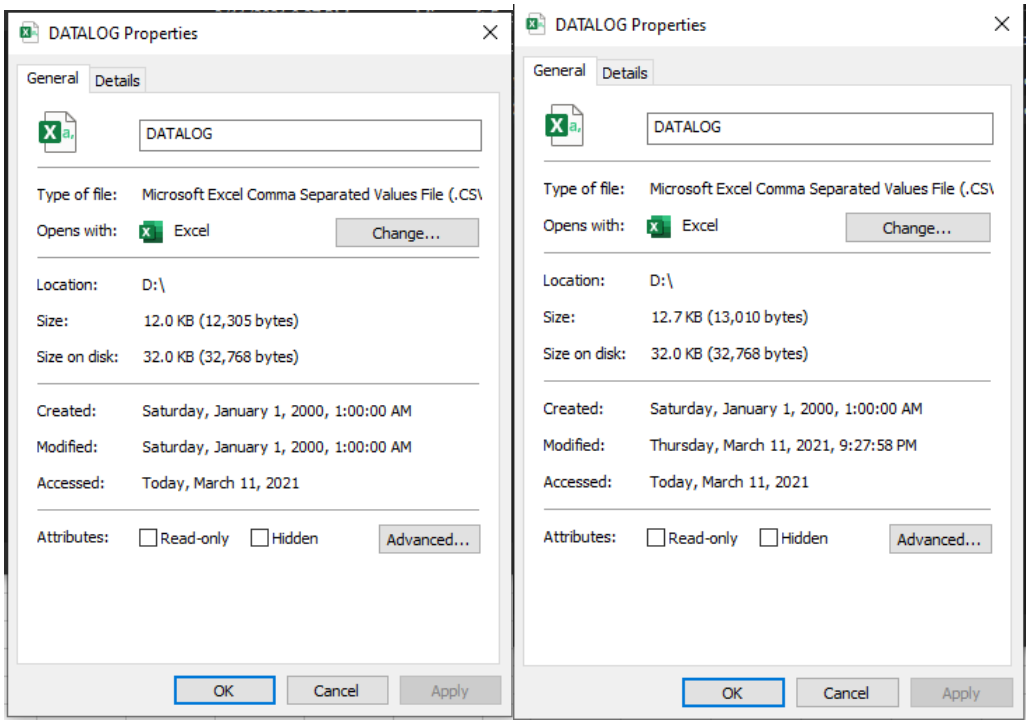


Figure 6-9: Size of datalog file before one communication cycle on left, and size of datalog file after one communication cycle on right

Requirement ID: 14

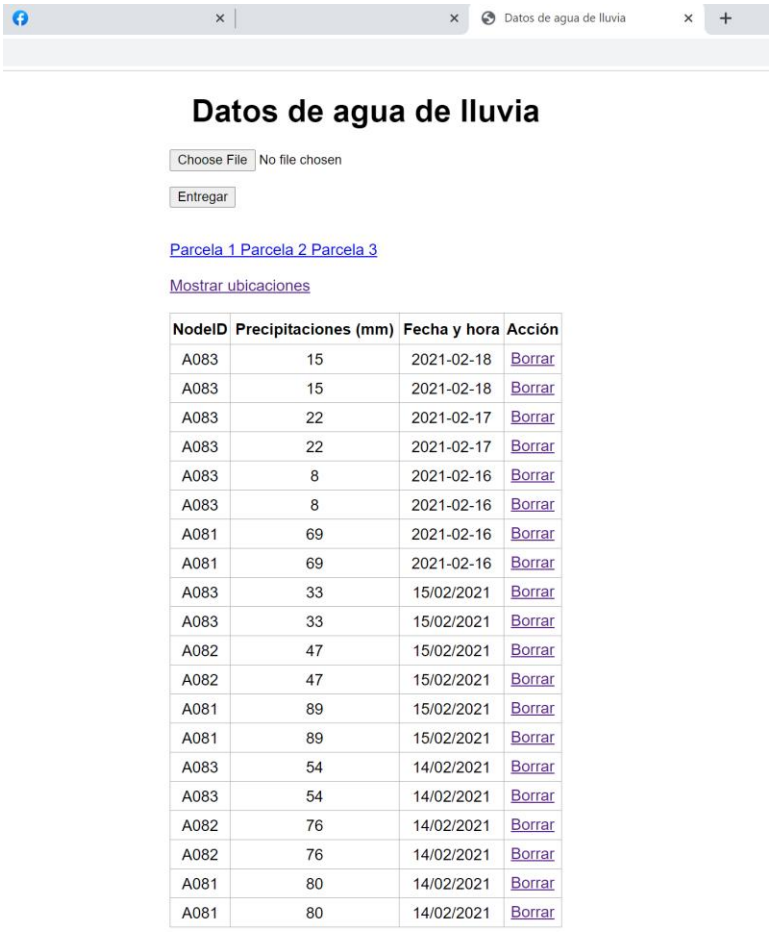


Figure 6-10: Translated web page