Senior Project Final Paper

JPCP Chickadee Project

Author: Ryan W. Frazier

Table of Contents

I – Introduction

The project that I chose to become a part of was one that I had heard discussions of, both through fellow undergraduates and presentations from Dr. Alec Lindsay of the Biology department. The project that I am referring to is known as the JP Chickadee Project (JPCP). The project sounded ambitious and intriguing and had many elements that needed to be tied together in a way that would be easy for people to understand and differentiate the technologies being used to gather data. When listening to the presentation that Dr. Lindsay gave, and hearing what he was asking for assistance with, it sounded like a neat side project that could be accomplished by two or three students within a semester. Boy was I wrong…

Although the majority of my senior project portion would be software orientated, I had also included many hardware aspects into my proposal just in-case I would find the amount of my work lacking. Needless to say, this was never an issue throughout the project in its entirety. In fact, we had to limit the amount of hardware that we included into our first iteration of bird-feeder because we were so overwhelmed with the overall task at hand. That being said, my senior project portion ended up encompassing the JPCP project as a whole, because how else could I test specific softwares and code without having the hardware implemented that would be using them. Essentially this brought me to one final ultimatum: would we be able to push out the product that we had promised Dr. Lindsay?

II – Overview

The Chickadee Project was presented to the computer science department as an opportunity for CS students to get some ‘real-world’ experience. We would be developing a ‘smart bird-feeder’ of sorts that eventually would be deployed into the field and used to collect eating, dominance, and movement behaviors of chickadees in the local Trowbridge Park area. The task that Dr. Lindsay presented us with was to collect information from chickadees that had been previously caught, banded with RFID tags, and then released. The information that he hoped to gather included the unique RFID tag number associated with each bird, a time and specific location of which bird feeder the bird was currently feeding at, and a corresponding weight. This information would be packaged together and uploaded to a live-updating website of which contained the unique bird profiles associated with each tag number. Community

members could then visit the website and see updated information of which chickadees were eating where. The website was already in place thanks to David Germain, a fellow NMU student at the time, and was in a stasis period awaiting data to display.

This is where my current group and I come in. Our job is to analyze, design, develop, test, and implement a bird feeder that can acquire this data from the field. There were obstacles from the beginning that we knew we would have to adapt too such as the drastic weather conditions that our hardware would be exposed too; the location and topography that the Trowbridge Park area presented us with, and how we would get the data to the website in a near instant fashion. The unforeseen obstacles however greatly outnumbered our original estimates. A system such as this needed to be very robust, easily maintainable for the non-technical user, and modular in the sense that Dr. Lindsay could transport and set-up feeders in new locations. This paper covers the hardships that we faced along the way, the software and hardware that we implemented to overcome these hardships, and the reasoning behind our design decision making; not to mention all of the testing that was required in order to integrate each piece of hardware. In addition to each of these key pieces, I will also introduce major hardware and software changes that we needed to make in order to produce a more reliable product.

III – Data Flow

Before I explain in detail about the core aspects of how the data is flowing from the field to our website, I thought it might be a good idea to provide a diagram that will allow you to see the big picture without adding the technicalities just yet. Below is a diagram of how we initially planned our flow of data:

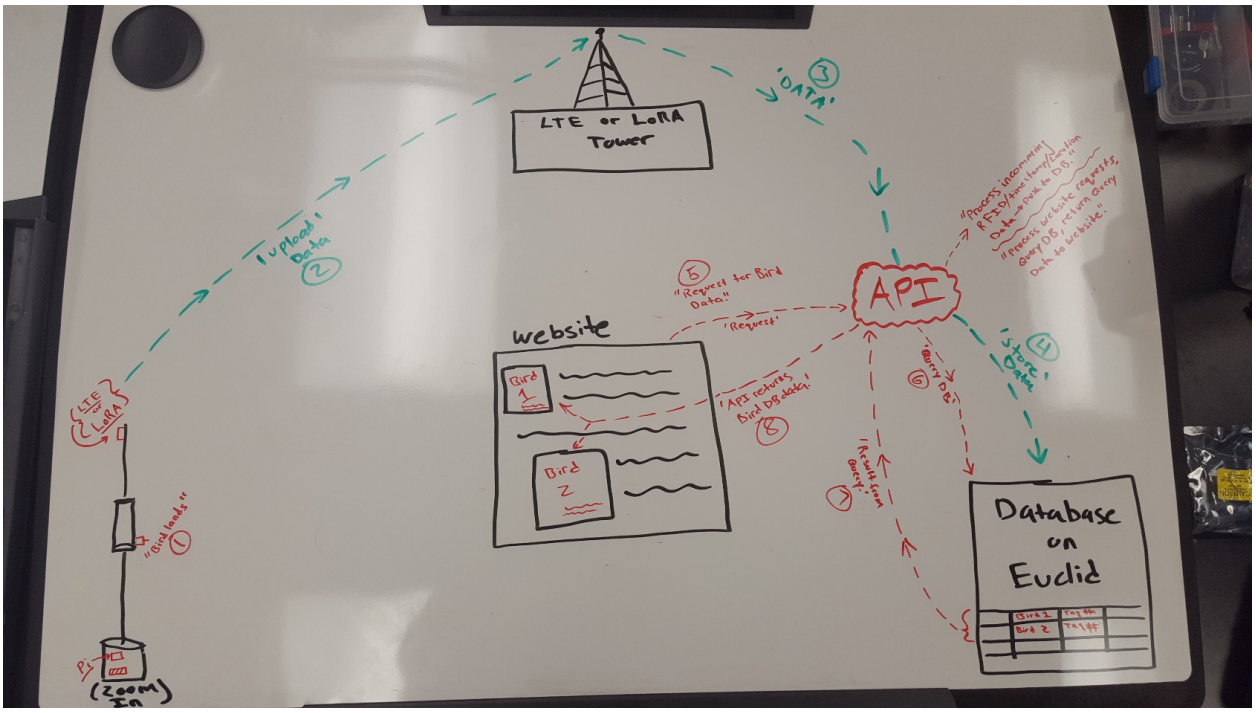

Figure 1: Data Flow using LoRaWAN

Figure 1 shows the 'complete' picture of how the data will flow. The image begins in the bottom left corner when a chickadee lands on one of our feeders. When the bird lands, its unique RFID tag number, weight, and time is recorded by the Raspberry Pi. It then gets packaged in JSON format, and uploaded via the LoRaWAN protocol to our Gateway. The data is then transmitted to our API that in turn stores each incoming segment into our database.

In our initial analysis of the overall data flow, we had decided to use an LTE signal for our wireless connection and transfer of data to the JPCP website. This seemed like a great idea since NMU already has in place their own LTE network with a unique band for transmission. They also have all of the necessary hardware in order to communicate with this network and volunteered to provide our group with LTE hotspot devices for each of our bird-feeder nodes. This was fantastic and it not only saved us the huge headache of figuring out how we would get each bird-feeders' data to our website, but also reduced the overall cost of each bird-feeder. Sadly, this technology would not be used due to

unforeseen circumstances with our design and limitations of the hotspot hardware itself. These limitations will be explained in a later section (section IV) where I talk specifically about our finalized hardware choices and why we chose them.

Our final solution to the LTE issue was to use a new transmission technology called LoRaWAN. LoRaWAN (Long Range Wide Area Network) is a very new technology that was purchased by Semtech in 2012. Version 1.0 of the LoRaWAN specification was released in June 2015 and is maintained by the LoRa Alliance which is comprised of more than 500 member companies who are committed to enabling large scale deployment of Low Power Wide Area Networks (LPWAN). As shown in *Figure 1*, the data flow will remain the same with minor changes being made to what type of tower each of our Nodes (bird-feeder) will connect to. Each Node location in the field will have a LoRa module attached to our Raspberry Pi instead of an LTE hotspot device. This LoRa module needs to communicate with a LoRA Gateway before the data can be uploaded to our database. This single LoRa Gateway will replace our old LTE reception antenna that is pictured near at the top of *Figure 1*. For data transmission using LoRaWAN the US standard is a frequency between 902-928 MHz. This is pre-configured at a hardware level within each of the LoRa module boards. Each LoRa node will connect to the Gateway just before starting to transmit its data. As the Node begins to transmit data to the Gateway, each time a transmission is received by the gateway it will push that data to The Things Network (TTN). The Things Network focuses on enabling low power Devices to use long range Gateways to connect to an open-source, decentralized Network to exchange data with Applications. In simpler terms, the TTN provides us with a cloud platform that catches all of our incoming Gateway traffic. We can then forward this data from the TTN using their console web application to our API which then inserts it into our database.

IV – Hardware

The hardware implementations that we have finalized for this project going forward have changed greatly from when we originally grouped up and had begun designing our first prototype. I'll start by briefly describing our first overall design and the hardware it included. Then, I'll elaborate on the introduction of each new hardware piece and what our ultimate node design ended up being. I'll also explain briefly at a high level about why we made the changes that we did, but leave much of the detailed reasoning in my latter section *VII – Troubleshooting and Testing*. What made this project unique is that in addition to coming up with hardware technologies for data collection, we also needed to redesign the physical layout of the existing bird-feeder. The layout of the existing bird-feeder was as follows: a cement-filled bucked that would be buried beneath the ground, a 10-foot metal rod protruding from the bucket with a hollow two-foot PVC tube connected at the top of the metal rod. As you can see below in *Figure 3*, the changes that we made would have exposed wires running the length of the metal rod as well as going into the hollowed PVC tube. Running these wires and keeping them concealed as much as possible was a crucial part for a successful product. We also wanted to make as many components as modular as possible. This would make the task of swapping out broken pieces easier, and also provide a means for expansion later down the road.

~ Electrical Hardware ~

As shown in *Figure 3*, the base of our feeder would house the Raspberry Pi Zero W with a large amount of custom modifications applied to it. The brain of this entire operation, the RPi Zero W, was a great solution to a few of the main design aspects that we looked to overcome right away: a small form factor, great computing power, and a sense of modularity for expansion purposes in the future. A 'complete' RPi, as shown below in *Figure 2*, was paired with a prototyping board that allowed us to interface with the GPIO pins of the RPi, and a LoRa/GPS hat which we would use for transmitting the data using LoRaWAN. Most of the custom work is done on the RPi prototype board. This is where we solder all wires and microprocessors before covering it with the top level LoRa module.

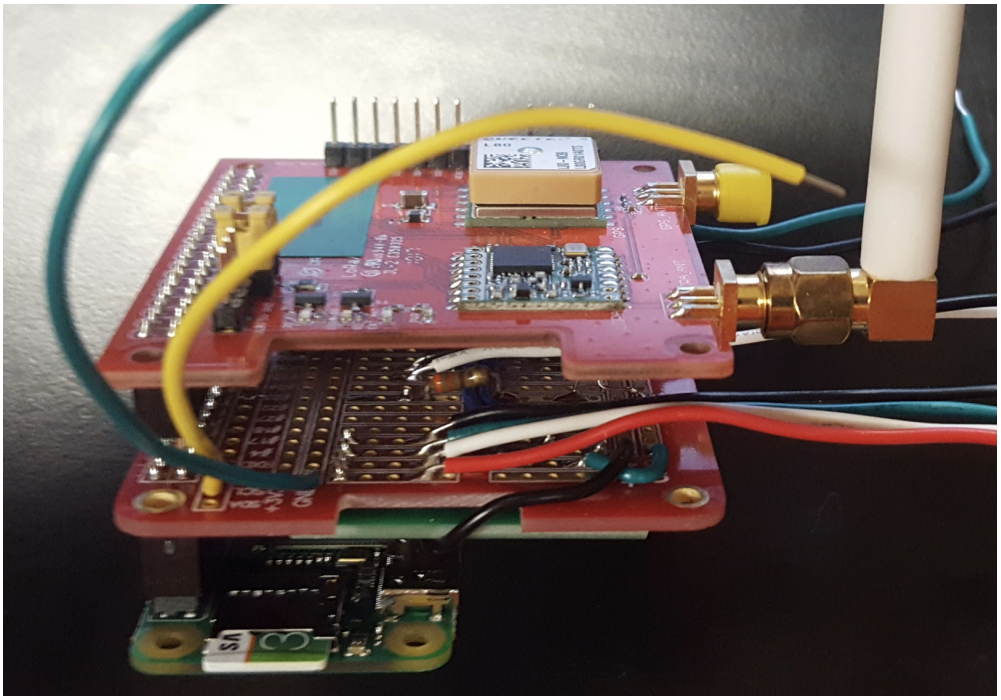

Figure 2: A complete Node. Bottom layer is a Raspberry Pi Zero W. Middle layer is a Raspberry Pi Prototyping Board. Top level is LoRa/GPS Hat.

We decided to use the smaller and less powerful RPi Zero W, as opposed to the RPi 3b because we did not need the extra computational power and due to this we were able to greatly improve our battery life. The prototype board allowed us greater access to the GPIO pins and provided extra space for additional expansion pieces as the biology department sees fit. We also needed the prototype board because we had to add several micro-chips such as the DC/DC converter for incoming power reduction, the hx711 Load-Cell amplifier which allowed us to use the weight sensor, and the RFID UART-chip which enables the RFID sensor. These three micro-processors cannot be seen in *Figure 2*, but they are soldered on the underside of the prototype board. The sensors for collecting weight and the RFID tag will be the actual perch that the chickadees are landing on. The weight sensor will be the 'arm' and the RFID reader will be attached to the end of the weight sensor.

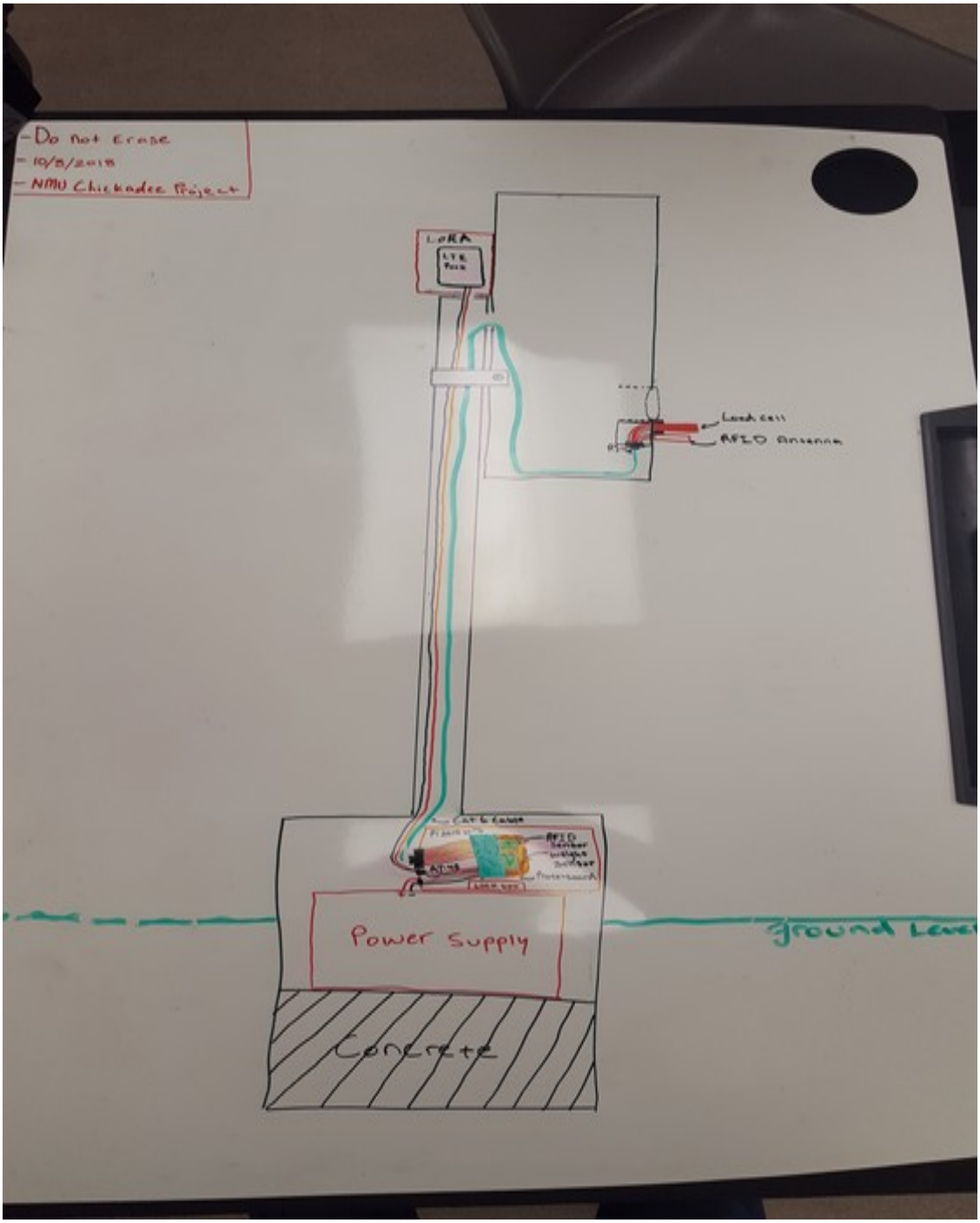

Figure 3: Original Bird-feeder Layout

V – Software

Deciding on what hardware we would use was a difficult task, but the software that I hoped to incorporate to control these hardware pieces proved to be very difficult in its own right. I'll break this section up into several smaller sub-sections, relating each software piece to the specific hardware components that it would control. For the sake of brevity, I'll focus mainly on the final versions of software solutions that I came up with. At a higher level, I will order these sections closely to how the overall data will flow. First, I'll start by describing how I get an RFID 'hit' from a chickadee, as well as determining how I am able to detect banded vs. non-banded chickadees. Next, I'll talk about what I do to log this data. Finally I'll describe how I handle the sending and receiving of log files using the LoRaWAN protocol.

~ Acquiring data from a banded chickadee ~

As a chickadee lands on our custom created perch, we have less than one second to get an accurate RFID read and corresponding weight before it flies off. My code runs indefinitely while the Raspberry Pi is on and is only fully executed when an RFID tag has been recognized. I have a separate version of the code where I incorporate the load-cell which in turn activates the RFID reader, this version will ultimately be used to detect non-banded chickadees later down the road. This version however has an unusual CPU bug where a 'pre-determined' weight is sometimes applied to the load-cell when it tries to tare itself to zero. This causes invalid reads and therefore the code needs to be exited and re-run; this cannot happen hence why it is not in my final iteration of code.

When my final version of code runs, an RFID tag is read from the RFID sensor. This is paired with a timestamp and saved into a log file. I have been able to refine this capture state down to the ability of being able to get tags every 300 milliseconds. I may be able to reduce this further, but from our tests this provides the best results in terms of getting all tags read accurately.

~ Logging the Data ~

As I mentioned above, saving the data to a log file is straight forward; I just save the RFID tag number with an associated timestamp of when it landed. The log file format appears as *"log#.out"*, where the '#' is an incremented number. The tricky part comes when I need to perform file and directory manipulation. When my log file hits a certain quota, lets say 100-reads or 'hits', I need to save a copy of that log file into both a *backup* and *transmit* directory. The backup directory is just a location of where in the event we have data loss during the transmission phase, a raw log file will be kept. In order to save

into the backup directory, I first do a search in order to determine the highest l*og#.out* file that currently exists. Just before saving into the *backup* directory, I increase this returned log number by one and apply it to the just-about-to-be saved log file; then save it into the directory. As for saving into the *transmit* directory, I perform the same query into that directory just before copying the file over. The catch here is that I only need to perform this search if a transmission state is currently being run. If not, then the directory will be empty and I can save the log file without any restraints.

~ LoRaWAN – Sending and Receiving ~

Let me clear up one thing before I start; LoRa and LoRaWAN are not the same thing. Simply put, LoRa only contains the link-layer protocol whereas LoRaWAN provides us with a network layer in addition to a link-layer. With LoRaWAN we are able to communicate with our base station, the LoRaWAN Gateway, that is already connected to a cloud platform know as The Things Network (TTN). The Things Network is a data relay of sorts that allows us to register our Nodes and Gateway and view incoming connections in real time. The advantage of using LoRaWAN is that we have peer-to-peer encryption on both ends. When I reach the quota on my log file and copy it into the transfer directory I also make a call to some C-code that begins the transmission. The transmission code looks into the transmission directory and will send all log files, line by line and deleting the files when done, until the directory is empty. When the transmission code is called, the C-code generates a file in our top directory. This is what I search for whenever I save a new log.out. The transmission sends the data to our Gateway which in turn forwards the data to The Things Network. The Gateway is connected directly to the internet via a WiFi signal. Once the data reaches the TTN in an encrypted format, it gets decrypted using some generic code that we needed to make edits to in order to suit our data payload. Once the data is decrypted, the TTN then sends the data, via a POST request, to our API. The API that I have written will process any incoming information for errors or duplicates and insert it into our database. I wrote this API in python as well and figured it would be a great starting point to learn about database manipulation since I have hardly any experience with that. Once the database is populated the website can begin requesting information from it. We did not consider the website so much as we did the design, implementation, and testing of our Node-to-Gateway technologies.

VI – Troubleshooting and Testing

I'll start by saying that the amount of testing that was required for each piece of software and hardware easily pushed us back an entire year. For instance, it was only when we began testing our LTE module, after we had implemented it on the software side, that we realized it would be using too much power to be a feasible option. More on this point later though. The amount of testing that was required was tremendous. It seemed that every step we took in the 'right-direction' would fail miserably when we started to test that specific piece. In terms of the amount of testing that was required for software vs. hardware, I would have to say that troubleshooting the hardware portions took much more time. This was largely due to the fact of there being many uncontrollable variables that we simply could not account for in a controlled testing environment. We also wanted to finalize the hardware before I started to include software that would utilize it. All this being said, I'll break down some of the major troubleshooting issues that we ran into such as battery conservation, LTE vs. LoRaWAN, Raspberry Pi 3b vs. Raspberry Pi Zero, RFID and Load-cell designs, weather-proofing, and cost.

~ Battery Conservation ~

To start things out, our biggest battle from the git-go was battery usage. The current system that was in place could already only last about one week, tops. In theory and by our initial rough calculations, just adding a Raspberry Pi into the mix would decrease battery performance down to a couple days, and that was just hooking the Pi up to the battery. One constraint that Dr. Lindsay had requested would be for the new system to last a full week, Saturday to Saturday. There are a few hardware pieces in this sub-section, battery conservation, that have their own sub-sections but play a direct role here as well. To save on the overall cost of each feeder, Dr. Lindsay had hoped to keep using the same batteries that he currently has. We had worked out a few solutions that involved purchasing larger amp-hour batteries, essentially a car battery, but this would be a very large burden on whomever would be swapping them out when they inevitably needed a charge. We looked into using solar power as a means to combat some of the power draw and provide us with a small source of renewable energy but there were too many outliers to account for. The main two reasons being first the setting of these feeders. They will be in a densely wooded, publicly located area which means they are susceptible to people stealing the panels and not enough sunlight is able to reliably penetrate the tree canopy. Secondly, the use of the solar panels would only come into play during a few months of the year, and during the winter months there is very limited sunlight with the added nuisance of having to remove snow from the panels face. Our final design would keep the original batteries and require us to think of alternative solutions to

lower our total draw power (TDP).

The next two changes that we made that impacted and greatly improved our battery life was the switch from the Raspberry Pi 3b (RPi 3b) to the Raspberry Pi Zero W (RPi Zero W) model, and the switch from LTE to LoRaWAN. Swapping the Raspberry Pi's was purely a hardware change, whereas going from LTE to LoRaWAN involved both hardware and software changes. Clearly put, the RPi 3b had more computational power than what our project required. When idle, the RPi 3b would draw 300 mA (1.5 W) as compared to the RPi Zero W which would only consume 100mA (0.5 W) at idle. During our test attempts at max TDP, which we were never able to actually get to, we saw that the RPi 3b would draw upwards of ~5 W, while the RPi Zero W would only draw ~1.4 W. Making this switch alone saved us more than half of our overall battery life. I'll spare all the technical differences between each model, but the main reason for this difference in TDP was due to the significantly less-specced CPU found on the RPi Zero W. As for making the change from LTE to LoRaWAN, we were able to remove another piece of hardware that was using a lot of our battery: the LTE hotspot device. Our biggest issue with the LTE hotspot device was that it needed it's own internal battery to be charged in order to turn on. This in turn meant that if the internal battery didn't have a charge, it would first need to begin charging itself from our main external battery until it reached 5% charge. Conclusion: the LTE hotspot would continually try to charge itself from the main battery which in turn used more battery life than the RPi 3b at half-max TDP. Another issue with these hotspot devices was their cold weather operating temperature. During our cold room testing, where we subjected each piece of hardware to -20 degree Fahrenheit temperatures, we found that the device itself would not actually turn on if the battery fell bellow a 0 degree Fahrenheit mark for extended periods of time. Considering that this is a norm for the winters in our area, it simply did not suffice. Using LoRaWAN, all we needed was to add a LoRa/GPS hat on top of our RPi Prototype board. The LoRa hat only increased our overall TDP by ~10 mA and even then it would only draw power when we needed to transmit our data.

~ LTE vs. LoRaWAN ~

LTE is almost in every way an upgrade from LoRaWAN. It was faster, more reliable, and more cost efficient due to us being able to acquire the hardware for free. These reasons still were not enough to outweigh the issue of using it too much battery. While the speed of data transmission was greatly improved over that of LoRaWAN (local field tests showed: LTE ~1 mbps, LoRaWAN 300 kbps), we were only sending small text files which averaged 2.1K. 300Kbps would be more than enough to send the amount of data that we would be collecting in a timely fashion. One of the hardest parts was figuring

out how to correctly use LoRaWAN and implement that on both our Gateway and each Node. Since the technology is relatively new (2015), the documentation for it was in a sense lacking. That's not to say that there wasn't any documentation on it, more so it was talked about in a general sense of *"how things should work"* when using it. The majority of code that we found when looking LoRaWAN libraries was written in C. This was largely due to the fact of its direct communication with the hardware. I spent many hours looking through pre-written C code just to determine how exactly we could send a simple custom packet. Eventually I was able to find the C-code that was actually doing the sending. Jon and I modified it so that it would now look into our transmit directory for eligible log.out files to send. This was significantly more difficult compared to using LTE where we could do direct POST requests to our API from each individual Node and bypass the Gateway altogether.

~ Raspberry Pi 3b vs. Raspberry Pi Zero ~

As I stated in the previous battery sub-section the greatest reason for changing to the RPi Zero W was because of battery constraints. That being said, we also did not need the unnecessary computational power that the RPi 3b provided. The RPi 3b has a quad-core 1.2GHz processor where the Zero W model only has a single-core 1GHz processor. During our testing phases of gathering data, saving log files, and transmitting the data, we found that the single-core 1GHz option provided more than enough speed and threading capabilities to handle all data collection and file manipulation in a concurrent nature. When we added the prototype board and began to implement each micro-controller, we had to go through and fully re-test everything at every step. We needed to be absolutely sure that anytime we added a new piece all other pieces would continue to work exactly as we had planned.

~ Weather Proofing ~

Throughout this entire time of thinking about what hardware choices we would have to make, we also needed to design enclosures that could withstand the year-round conditions here in the U.P. Since we were trying to make everything as modular as possible this actually helped when it came time to create our weather proof boxes that we designed. The RPi Node enclosure was created out of a plastic Tupperware container. We created cutouts in the side, sealed with epoxy, as an access point so that we could insert our power cables from the battery. As for the connections near the top of the bird-feeder we opted to use a type LB conduit electrical box with a removable back panel. This would allow for the wires to be detached and the 'bird-house' (PVC tube) to be removed from the pole so that it could be refilled with birdseed. Essentially anything that has a wire coming out of it needed to be contained in

some type of enclosure. Each container that we decided to use was tested in the cold room here on campus and also subjected to water tests to check for leaks. We were able to purchase all pieces in bulk which also helped with reducing the overall cost. We were only able to test everything in a controlled environment so we are aware that many pieces may actually fail when they are put into the field. This also was a leading factor that caused us to make each section as modular as possible; so that we can build individual backup pieces without having to create an entire new bird-feeder.

VII – Conclusion (What I learned/hoped it would be)

In a sense, everything about this project was a new experience for me. The amount of learning that I was subjected to was exponentially larger than what I had originally thought and easily incorporated everything and more that I have learned in my entire computer science career while at NMU. I had never worked in a group environment on a project that was starting from scratch. I've never programmed in Python before this project. Other than one semester in Micro-computing Architecture I had zero experience with any sort of circuitry, power consumption equations, hardware design, and the implementation of an actual product for a client. With my final version of code that I will be considering for this senior project I must say that I had assumed I would be writing more than what I ended up with. This was a very naive thought at the time and looking back at everything that I have learned, writing code was a supplement to the project as a whole and not the other way around. I could have written extra code that may or may not have served a purpose depending on the hardware that may or may not have been added. My finalized code already includes portions that will not be necessary for the first initial product release. This code that I wrote for the Load-Cell has been tested and does in fact work, but there are specific instances that cause the code to crash due to faulty weight readings. We did not have enough time to fully test what was causing this whether it be hardware or software related. This is just one instance where I provided steps and a means to incorporate future modifications that the Biology department might need. It was also a great experience to be able to communicate with a client and relay any short-comings that we were facing, such as the load-cell. Though I had written less code than I had originally planned; I believe that in conjunction with building our Raspberry Pi Nodes, coming up with solutions for battery consumption, modifying the overall bird-feeder to incorporate new hardware, and planning for the longevity of this project going forward that I was able to acquire a great

perspective of how product creation in the real world actually is. This will be a fantastic experience that I will be able to continually draw from in my later endeavors in my professional career. I am glad that with this project I was able to design software, build hardware, and push out a product that people will actually use.

Throughout this project I was able to actually practice the higher level elements that are taught in our Computer Science curriculum. We mainly followed a mixed version of agile and waterfall methodologies where we valued constructive feedback, respect, simplicity, and communication. Since we had a small team a waterfall style approach suited us well because we were all located in the same room most of the time. We had, unknowingly at the time, started to introduce agile practices such as stand up meetings and whole-team inclusion whenever we met. This greatly helped us to lock-in and gear ourselves towards driving the project to completion. I'm glad to say that we were able to produce the product by the requested time and even though we have more to do, my group is very pleased with this first release iteration.