1 Overview

For my CS2 Honors Project, I wanted to make a system that could remotely track orders being assembled, using OOP principles.

1.1 Criteria

From the “Outcomes/Product” section of the contract:

> I will have a minimum of two small devices with e-ink displays that will show an order number and some information about the order. The devices will be able to be charged in some way when they aren’t in use (like through solar power or induction). The devices will communicate with a central host over BTLE (Bluetooth Low Energy) or LoRa (Long Range), and the host can update each device with new information. I will be able to push buttons on each device to update its order status. The host can also see a list of all the devices and their order information, including status.

Bin Tags:

- [ ] send messages over LoRa
- [ ] receive messages over LoRa
- [ ] display order numbers and statuses
- [ ] update displays freely (every time an order updates)
- [ ] button press updates an order’s status
- [ ] relies on OOP (classes and methods)

Gateway:

- [ ] sends messages over LoRa
- [ ] receives messages over LoRa
- [ ] sends messages over MQTT
- [ ] receives messages over MQTT
- [ ] relies on OOP (classes and methods)

Central Host:

- [ ] sends messages over MQTT
- [ ] receives messages over MQTT
- [ ] stores boards/orders in a database
- [ ] relies on OOP (classes and methods)

1.2 Hardware

As expected, the hardware I was using changed over the course of the first and second iterations. Listed below is the hardware I am using for the final product.

Bin Tags:

- 2 Adafruit Feather M4 Express boards
- 2 Adafruit LoRa Radio FeatherWing boards
- 2 Adafruit 2.9" Greyscale eInk Displays

Gateway:

- 1 Raspberry Pi 3B
- 1 Adafruit LoRa Radio Bonnet

Central host:

- My own computer

2 First Iteration

2.1 Criteria

Bin Tags:

- [x] send messages over LoRa
- [ ] receive messages over LoRa
- [x] display order numbers and statuses
- [ ] update displays freely (every time an order updates)
- [x] button press updates an order's status
- [ ] relies on OOP (classes and methods)

Gateway:

- [ ] sends messages over LoRa
- [x] receives messages over LoRa
- [x] sends messages over MQTT
- [ ] receives messages over MQTT
- [ ] relies on OOP (classes and methods)

Central Host:

- [ ] sends messages over MQTT
- [x] receives messages over MQTT
- [ ] stores boards/orders in a database
- [ ] has command line interface
- [ ] relies on OOP (classes and methods)

2.2 Process and Outcome

The bin tags in the first iteration were successful in sending messages over LoRa and displaying these messages simultaneously. However, the libraries I was using were limited in a couple of ways.

First, the displays couldn't be updated more often than 3 minutes or the program halted. This was to help prevent the displays from getting permanently damaged (e-ink displays are apparently pretty fickle in this regard), but it did get in the way of debugging while I was programming. It sucked to have to wait 3 minutes before knowing if my code worked or not. I didn't want to switch to more powerful OLED displays, because those used up a lot more battery. Since I am planning to attach these to bins for prolonged periods of time, I needed displays that could function even if they weren't powered. So I thought this meant I needed to find another library that could talk to the e-ink display without the 3-minute hassle.

Second (and definitely more important), I could only send messages over LoRa; I couldn't receive any messages. This wasn't a problem with the LoRa radio itself, just a problem with the library (I think the library was written for sensors, which only need to send data). In this iteration, the gateway I was using operated on a higher-level networking layer for LoRa called LoRaWAN, which had more server-side support. Unfortunately I hadn't found any other CircuitPython libraries that could talk over LoRaWAN, so I thought I was going to have to switch over to a much more versatile C++ library.

3 Second Iteration

3.1 Criteria

Bin Tags:

- [x] send messages over LoRa
- [x] receive messages over LoRa
- [x] display order numbers and statuses
- [ ] update displays freely (every time an order updates)

- [x] button press updates an order's status
- [x] relies on OOP (classes and methods)

Gateway:

- [x] sends messages over LoRa
- [x] receives messages over LoRa
- [ ] sends messages over MQTT
- [ ] receives messages over MQTT
- [ ] relies on OOP (classes and methods)

Central Host:

- [ ] sends messages over MQTT
- [ ] receives messages over MQTT
- [ ] stores boards/orders in a database
- [ ] has command line interface
- [ ] relies on OOP (classes and methods)

3.2 Process and Outcome

For my second iteration, I planned to try out the MCCI C++ library I mentioned above. My goal was to be able to send and receive messages over LoRaWAN, and to continue updating the display simultaneously. Since the boards wouldn't be running CircuitPython anymore, I planned to use a different library to talk to the displays, such as the Adafruit EPD library.

After some grueling attempts to use the Arduino IDE to talk to the M4 boards, and trying to figure out how to use the MCCI library, I realized that I wasn't making as much progress as I would have liked. I also realized that I had a much more straightforward way of getting two boards to talk to each other over LoRa, using Adafruit's RFM9x library. Instead of relying on LoRaWAN, I was just going to deal with the lower-level LoRa layer directly, and write more of my own server-side code for the gateway and central host.

By the end of the second iteration I had a simple gateway working that could receive/send messages to/from the bin tags over LoRa. I also started incorporating more object-oriented programming ideas into the code for the bin tags, starting with a `Message` and `Order` class. I still had a 3-minute display refresh problem, but I figured it wasn't worth switching over to C++ to get around that problem.

4 Third Iteration

4.1 Criteria

Bin Tags:

- [x] send messages over LoRa
- [x] receive messages over LoRa
- [x] display order numbers and statuses
- [x] update displays freely (every time an order updates)
- [x] button press updates an order's status
- [x] relies on OOP (classes and methods)

Gateway:

- [x] sends messages over LoRa
- [x] receives messages over LoRa
- [x] sends messages over MQTT
- [ ] receives messages over MQTT
- [ ] relies on OOP (classes and methods)

Central Host:

- [ ] sends messages over MQTT
- [x] receives messages over MQTT
- [ ] stores boards/orders in a database
- [ ] has command line interface
- [ ] relies on OOP (classes and methods)

4.2 Process and Outcome

In my third iteration, I finally figured out how to get around the 180-second display refresh constraint that was built into the `displayio` library on the bin tags.

I added more code to the gateway so that it could send order updates over MQTT. I also wrote a simple Go program that could subscribe to an MQTT topic and see the messages that the gateway published.

5 Final Iteration

5.1 Criteria

Bin Tags:

- [x] send messages over LoRa
- [x] receive messages over LoRa
- [x] display order numbers and statuses
- [x] update displays freely (every time an order updates)
- [x] button press updates an order's status
- [x] relies on OOP (classes and methods)

Gateway:

- [x] sends messages over LoRa
- [x] receives messages over LoRa
- [x] sends messages over MQTT
- [x] receives messages over MQTT
- [ ] relies on OOP (classes and methods)

Central Host:

- [x] sends messages over MQTT
- [x] receives messages over MQTT
- [x] stores boards/orders in a database
- [x] has command line interface
- [x] relies on OOP (classes and methods)

5.2 Process and Outcome

I refactored the Python code on the bin tags to incorporate a lot more OOP principles. Every physical component of the bin tag is now represented by a class (`Button`, `Display`, `Radio`), and I also refactored my `Message` and `Order` classes to make the code more streamlined. While the code on the gateway still lacks classes and methods, it can be easily refactored with a similar framework (`Message`, `Order`, and `Radio` classes).

Most importantly, I wrote the bulk of the central host code in this iteration, using Go instead of Python. The central host is accessed by two main commands:

- `serve` starts the server, which listens for MQTT messages and sends them, and it also has access to the database.
- `reset` tells the client to update a board with a new order number, and the client in turn tells the server (over MQTT) to update that board. Since the client doesn't have direct access to the database and can talk to the server over MQTT, I can run the `serve` command on my own computer and the `reset` command on a different computer, and it would still work.

6 Final Thoughts

Overall, I am very happy with how this project turned out. Although it doesn't use C++, I learned a lot in CS 2 that I was able to apply to my project. My code has classes with methods and constructors, exception handling with try-catch blocks, and pointers. Since Python is a dynamically typed language, I don't need to use templates for it because it is template-like by default. Go doesn't have classes or constructors, but instead I can attach methods to structs, and I can write functions that work like constructors (returning a struct from some given data).

6.1 Going Forward

Even though I fulfilled my honors contract, I have some goals for continuing this project:

- Refactor gateway code to use OOP (not too difficult, since I already have a similar framework in the bin tag code)
- Write a `list` command for the command line interface on the central host, to be able to list all of the orders without direct access to the database file (also not too difficult, since I already have code that accesses the database)
- Design and 3D print holders for the bin tags, so that they can be attached to bins
- Design and 3D print a case for the gateway