Construction of a Data Collection Network using RF95 Radio Modules

by

Ethan Godden

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

B.SC. COMPUTER SCIENCE HONOURS

in

The Irving K. Barber School of Arts and Sciences

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Okanagan)

May 2020

Abstract

IonDB is a system designed to collect and manage data on embedded devices. It is intended to be used within a network of micro-controllers distributed in an environmental setting where memory and power resources are scarce. The goal of this project was to implement a system that can efficiently and reliably transport data from sensor nodes to Raspberry Pi server nodes. This paper describes the structure of this implementation, the technology behind it, and how this implementation could be improved in the future.

Table of Contents

List of Figures

Acknowledgements

Thank you to Dr. Ramon Lawrence who guided me through this project. Without his weekly guidance, this project would have either failed or be far from completed.

Thank you to Dr. Scott Fazackerley who helped me with environment setup issues as well as describing the overall goal of the project. In particular, he helped me with setting up RadioHead on a Raspberry Pi board after weeks of me trying to get it to work.

Thank you to Jon Gresl for helping me get started in the project and for the starter code that he contributed. Much of the final code is just compartmentalized versions of Jon's original code.

Chapter 1

Introduction

Embedded devices are everywhere. They are in everyday devices such as lights, phones, cars, and many other devices. Because they are in many devices, there needs to be a way to store data efficiently on them.

IonDB is a system that manages data collection on small embedded devices with fixed memory and power constraints. This project aims to implement the component that transports data from sensors to server nodes and uploads the data to the database.

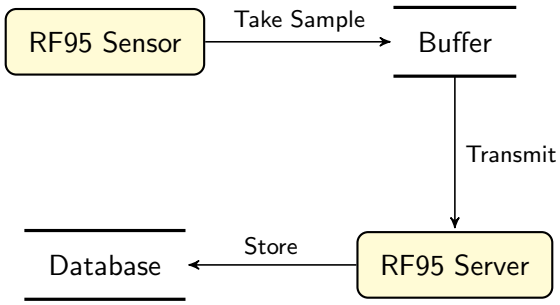

Figure 1.1: Level 1 DFD

1.1 Overview

The system consists of many sensors distributed in an environmental setting and possibly many server nodes. Every so often, each sensor takes a sample and stores the sample in a buffer within the sensor. When possible, the sensor nodes transmit the data within its buffer to server nodes, and the server nodes upload the data to a database.

When and where the sensor nodes take and transmit the samples are dependant configuration of the system. The frequency of samples and the types of samples are customizable at runtime. When each sensor transmits its data is dependant on how many sensors are connected to the same server

node and could auto-adjust by listening to network traffic.

1.2 Embedded Devices

This project is heavily based around embedded devices. These are small, low powered, computer systems.

Any device that can take samples and has a RF95 module can be used as a sensor node. The device used in this project for sensor is the Adafruit Feather M0 with RF95 module [Hop18]. This device uses the popular Arduino framework.

For server nodes, any device with internet access and an Rf95 module can be used. The devices used for this project is the Raspberry Pi 3b and the Rasberry Pi 1b together with the Dragino Pi Hat [Dra19]. This device uses the pigpio framework.

1.3 Motivations

The focus is to have data processing on devices with limited computational abilities. In particular, devices with limited memory and limited power capabilities. The Adafruit Feather [Hop18] used in this project has the additional limitation of having only one thread, but this could be replaced by another device.

1.4 Contributions

This project was built on top of the work of Jon Gresl from Summer 2019. [Gre19]

Chapter 2

Background

Every node in this system has an RF95 radio module. To understand the configuration of this system, one needs a basic understanding of how an RF95 radio module works. The two main technologies involved in an RF95 module are SPI and LoRa

2.1 SPI

SPI, which stands for Serial Peripheral Interface, is a synchronous communication protocol for systems with a single master device connected to many slave devices.

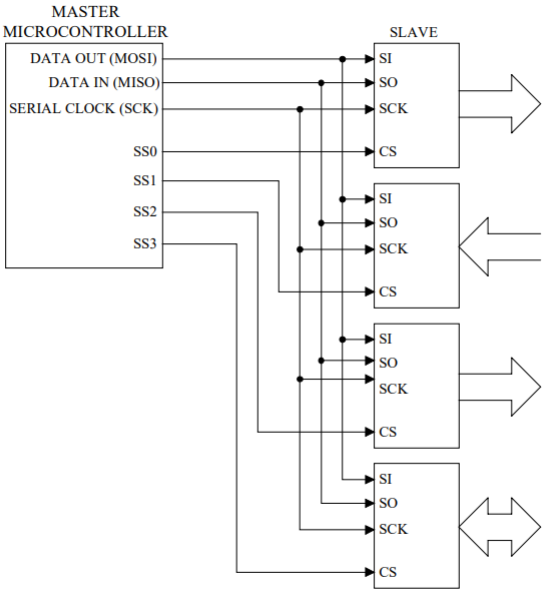

Figure 2.1: A typical SPI system. [Spi05]

There are four main components to every SPI system: a serial clock line, a MISO line, a MOSI line, and many SS lines. The serial clock is a signal generated by the master device to keep data synchronous between all devices in the system. The MISO, which stands for master-in-slave-out, is the line for receiving data from slave devices. Similarly, MOSI, which stands for master-out-slave-in, is for sending out data to slaves. Lastly, every slave device has a SS, which stands for slave select, line. This line is to allow the master device to choose which device to talk to.

In order to configure any device with an RF95 module, you need to know the SS pin number for that RF95 module as well as the interrupt pin number.

2.2 LoRa

2.2.1 Description

LoRa is a layer 1 WAN communication protocol. It is used for long range communication that requires little power. There are a number of techniqu

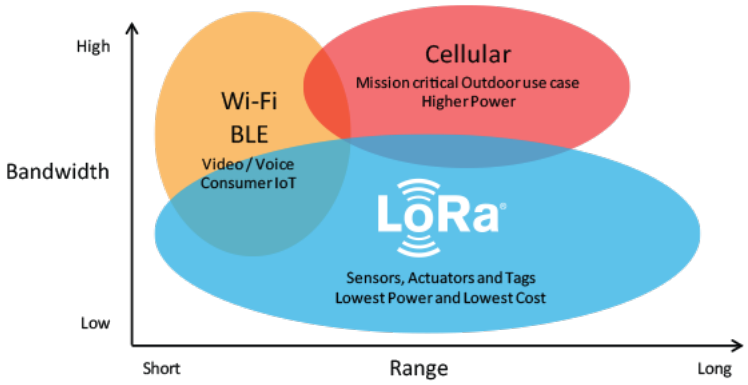

Figure 2.2: LoRa compared to other technologies[LoR]

2.2.2 Performance

For any device using radio communication, the signal strength is proportional to the square of the disance; it is defined by the following formula for free space loss:

$$FSPL = \left(\frac{4\pi df}{c}\right)^2$$

where $FSPL$ if is the frequency of the radio, c is the speed of light, and d is the distance.[Tel16] The way to picture this formula consider the sphere centered at the radio. As the sphere gets larger, the signal is spread out over a larger area. Since the surface area of a sphere is proportional to the square of the radius, it follows that signal strength is also proportional to the distance.

It should also be known that the distance a signal can be acknowledged from is dependent upon the power it is transmitted at. [Tel16]

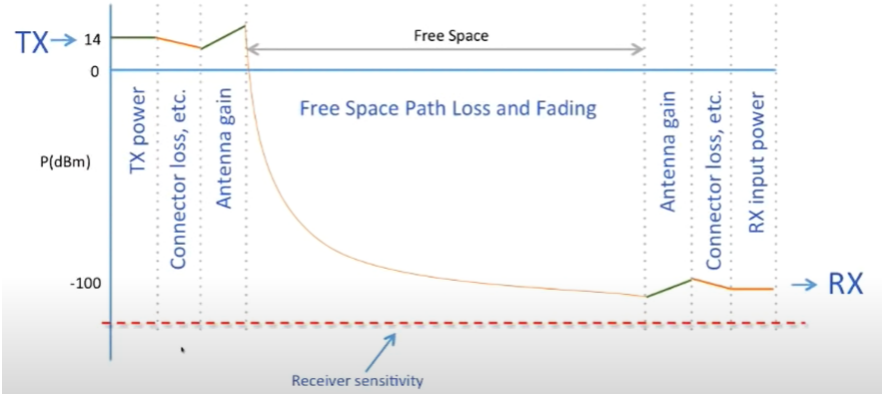

Figure 2.3: Link Budget for any radio

If the signal strength is still above a certain threshold, it can be received. The special thing about LoRa radios is that they can receive packets that have very low signal strength.[Tel16]

The last thing to consider when measuring performance of a radio-based network is the noise that results from signals reflecting off surfaces. In a vac-

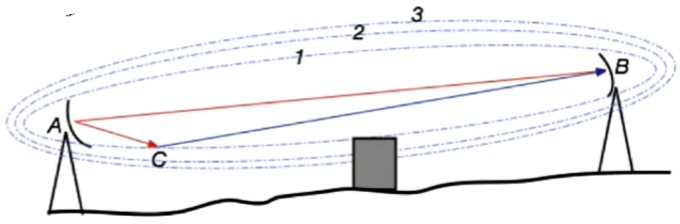

Figure 2.4: Fresnel Zone

uum with no obstacles, a LoRa radio could transmit packets to destinations hundreds of kilometers away. However, because of obstacles, this distance is reduced to, at best, around 30 kilometers. [Tel16].

The performance can be improved if all obstacles in a Fresnel zone are removed.[Tel16]. Because of this, the higher the device is from the group, the longer the distance it can transmit on average.

Chapter 3

Implementation

The current implementation has two main component: the RF95 network manager and the database.

The system consists of many sensor nodes connected to a server node. Every so often, each sensor node takes a sample and stores it in its buffer. The amount of sample and the types of samples are customizable. Based one the size of the network and the types of RF95 clients, the system will decide how often to transmit. However, the transmissions will following something similar to the layer 2 Aloha algorithm.

Once the data reaches the RF95 Server, the data is uploaded to the database. Once there, any application can retrieve it and update the display if it exists.

Along with this, the application can decide to change settings such as the frequency of the radio or the types of samples being taken. This is done by having the application update the database. The RF95 server will then notice the changes in the database and send the changes to the nodes that need to know the changes.

Note that this system had two main goals outside of the IonDB goals. For one, it should be easy to add additional boards to this system. This is done by defining clearly marked APIs. The other is to make it easy to switch to a pure C implementation. This is done using a system's function struction.

3.1 RF95 Network Design

There are three main components in the network design: the RF95 driver, the client API, and the buffer.

3.1.1 RF95 Driver

This code is responsible for interfacing with the RF95 driver. If any other boards are added to this system, they need to implement this driver.

If the new board uses Arduino, this is already covered. This code current interfaces with the RadioHead RF95 driver.

3.1.2 Sample Manager

This code is the main API for any sensor nodes attached to the network.

3.1.3 Buffer

This is an implementation of a regular queue using an array. There are four important properties with this implementation. The `head` property points to the next element in the queue, the `size` property is the number of elements in the queue, the `capacity` property is the maximum number of elements in the queue, and `data` is the actual data.

3.1.4 Dependencies

Currently, the only dependencies are RadioHead and the Ardunio Time library. There are files that interface with these dependencies so as to make it easier to change the dependencies later on.

3.2 Database Design

The database is where all data collected is stored. It serves as the interfaces for which applications can display data collected as well as change settings of the network

3.2.1 Setting Requests

Whenever the application wasnts to change settings, it inserts rows into this table

3.2.2 Boards

All supported boards are inserted here. This is a largely static table

3.2.3 Setting Config

This contains the current settings for any nodes in the network

3.2.4 Node

This contains all nodes that have been or are currently active in the network

3.2.5 Samples

This table contains all data collected

3.2.6 Collector

This table contains information on which nodes can take which samples

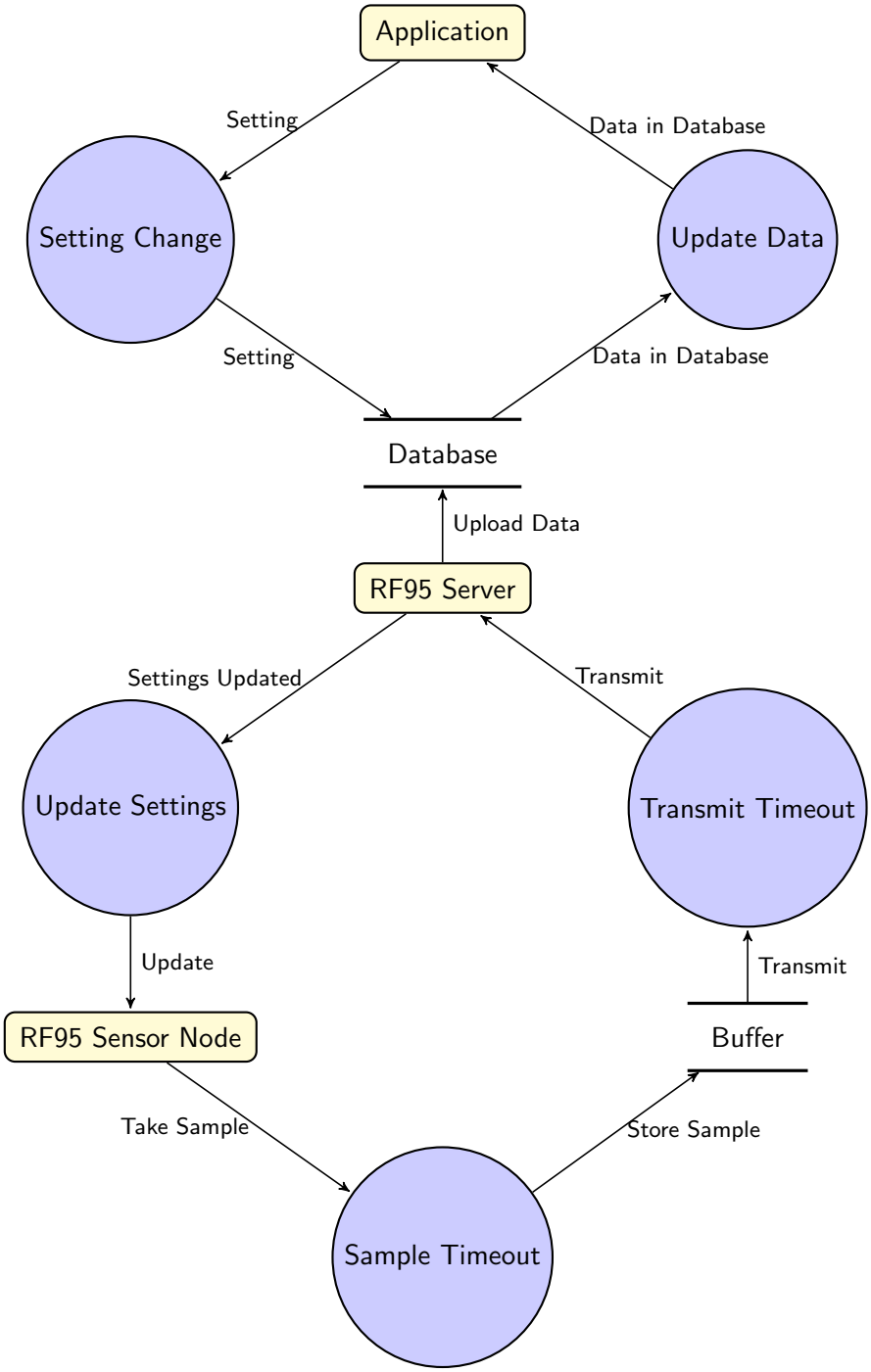

Figure 3.1: DFD Level 2

```
bool rf95_init(uint8_t slave_select, uint8_t interrupt_pin);

void set_rf95_address(uint8_t address);

bool channel_active(void);

bool rf95_send(uint8_t address, uint8_t *data, uint8_t dataLength);

uint8_t rf95_recieve(uint8_t *buffer, uint8_t bufferSize);


void set_cad_timeout(uint64_t timeout);

void set_frequency(float frequency);

void set_signal_bandwidth(long bandwidth);

void set_spreading_factor(uint8_t sf);

void set_coding_rate_4(uint8_t denominator);

void set_transmitign_power(int8_t power);
```

Figure 3.2: RF95 Driver

```
struct lorax_buffer
{
    uint8_t *__data;
    uint8_t __element_size; //Size of each element in bytes
    uint16_t __head;      // Index of next element. The address of
        this element is __data + (__head * element_size)
    uint16_t __size;      //Number of elements currently in the
        buffer. The number of bytes in the buffer is (__size *
        element_size)
    uint16_t __capacity;  //Max number of elements that can be
        stored in the buffer
};

lorax_buffer *create_buffer(uint8_t element_size, uint16_t
    capacity);

bool store_sample(lorax_buffer *buffer, uint8_t *sample);

bool remove_sample(lorax_buffer *buffer, uint8_t *removed_element);

uint8_t* peek_next_sample(lorax_buffer *buffer);

bool is_full(lorax_buffer *buffer);

bool resize(lorax_buffer *buffer, uint16_t new_size);

void print_buffer(lorax_buffer *buffer, char printf_arg);

void free_buffer(lorax_buffer *buffer);
```

Figure 3.3: Sample Manager

```
bool init_client(void);

void set_time(uint64_t);

bool add_sample_collector(sample_t type, SAMPLE_VAR_TYPE
    (*sample_collecting_function)(void));

void run(void);

void deinit_client(void);
```

Figure 3.4: Buffer Implementation

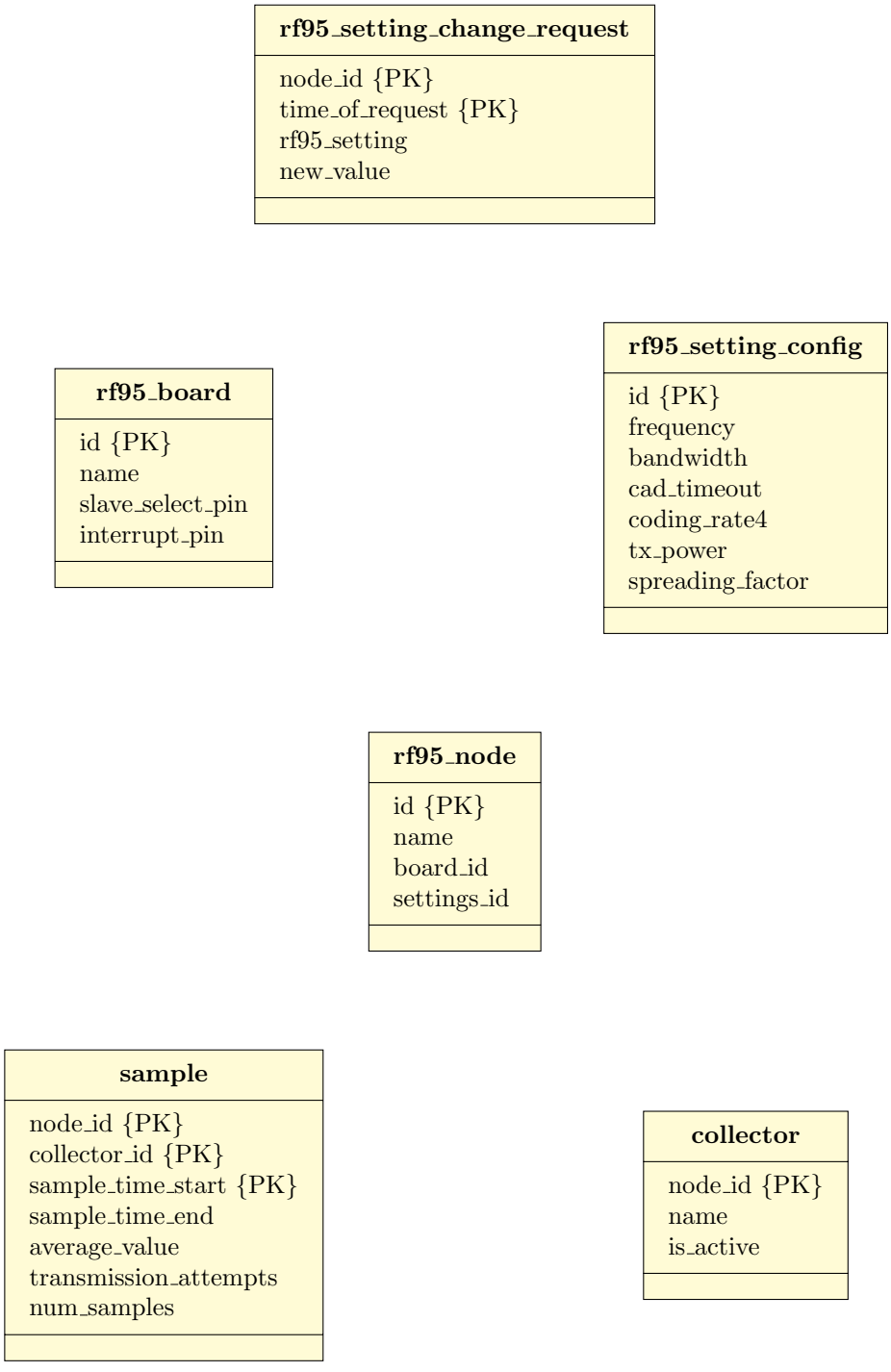

Figure 3.5: Database UML

Chapter 4

Results

Two Adafruit Feather board were tested using the full RadioHead network, and it was found that they could communicate around five kilometers apart

There are number of things to note about this measurement. For one, the devices where on the medium setting. Because RadioHead has a long-range setting, this distance could probably be improved further.

Another thing to note is the devices were connected to an LCD display while communicating here, and the antennas were crammed into a little space, so there is a possibility that antenna placement could improve the distance here.

The testing was done near a lot of trees and very close to the ground, so this could also be improved if the height was changed, or a more open field was used.

Lastly, there may have been some noise created by my car running the background together with its Bluetooth system.

Figure 4.1: Distance Testing

Chapter 5

Conclusion

Though the system is not finished, the overall design of the system provides technique for communicate with sensor nodes. It uses RF95 modules to communicate with, possibly many, server nodes. Even so, there are number of things that could be changed with this system.

5.1 Future Work

5.1.1 Completion

The current system is not fully implemented yet; communication between a Raspberry Pi and an Adafruit Feather M0 is not consistent, and it has a decent amount of corruption. This still needs to be figured out in order to continue.

5.1.2 Application

An application could be created that allows users to interface with the current system. One of the main design goals was to allow this system to be used with many types of applications such as a desktop application or a website. None of these things are implemented yet and should be if this is to be used.

5.1.3 Users

The database schema could be changed to account for data for a specific user. Currently, all data collected is stored in one table, data is only associated with a specific device; there is no way to determine whether one device belongs to a specific person. This could be changed by possibly creating a table per user or adding a field that associated data with a user

5.1.4 Versioning

There is currently no way to associate data collected to a specific version of the project. If this project is changed in the future, it might be important to know which version was used when collecting data because of changed to efficiency, accuracy, or precision.

5.1.5 Security

All data transmitted across the network is in plain-text. Further, there is no verification that data received by a server node actually came from one of the sensors in the network. This could be improved by using some form of encryption together with hashing.

5.1.6 Serial Numbers

Currently, the system manually assigns an address to every node, and this address has no relation to the specific device before it is assigned. That is, it is possible to permute the addresses in the network and have the same result. This could cause problems in the future because if the system is updated, it will be hard to remember which device has which address. If serial numbers could be used with the currently implementation, this would not be a problem.

5.1.7 Parallelism

The Adafruit Feathers used in this project are single-threaded devices. Further, no parallelism was used with any of the Raspberry Pi devices. Efficiency could possibly be improved if parallelism was used, but this would require different sensor boards.

5.1.8 Node Coordinates

If the RF95 sensors and servers had the ability to determine GPS coordinates, it would give any user the ability to use maps when interfacing with this system. There is also the possibility of letting the user choose the coordinates beforehand and then storing those coordinates in the database.

5.1.9 Alternative Implementation

The current system uses a algorithm similar to the layer 2 Aloha protocol; sensor nodes transmit data every so often. If the transmission fails, the

sensor node attempts to transmit again after a specific amount of time.

This is not the only approach to this problem. One could use a more round-robin style approach. A server node could broadcast to every sensor node information that would communicate the current stage in the round. This would prevent any two sensor nodes from transmitting at the same time, and possibly increase the number of packets received over time.

There are a number of possibly problems with this approach. For one, it would increase the number of packets travelling accros the network. If a sensor node continually fails to transmit data, the server node would continually send out broadcasts saying that its on the current sensor node. These packets are not present in our current approach.

Another possible problem is that it will make increasing the number of server nodes a challenge. Because the server nodes would manage everything, more server nodes makes this solution more complex.

If only one server node is present, this solution may be worth testing. However, because of these downsizes, it is most likely an inferior algorithm.

Bibliography

[Dra19] *LoRa GPS HAT Single Channel LoRa GPS module User Manual*, mar 2019. → pages 2

[Gre19] Jon Gresl. Improving Sustainability in Agriculture Using Sensor Networks. 2019. → pages 2

[Hop18] *RFM95/96/97/98(W) - Low Power Long Range Transceiver Module V1.0*, aug 2018. → pages 2

[LoR] Technical report. [link]. → pages v, 4

[Spi05] *SPI Interface Specification*, mar 2005. → pages v, 3

[Tel16] Thomas Telkamp. Lora crash course. VLDB '80, pages 212–223. The Things Network, 2016. → pages 5, 6