RF Spectrum Dynamic Map

Final Design Document

By: Lucas Higginbotham, Alexander Kinlen, George Robbins

Senior Design II

4/20/2017

Contents

Executive Summary

Problem
Radio frequency detection across a large area has long been a complex problem. In order to properly determine what frequencies are traveling through an area requires expensive equipment. If a client wants to create a radio frequency (RF) spectrum map over a certain area, they have to pay considerable expenses to utilize this equipment. However, a frequency map of a single point in time is not necessarily helpful as broadcasters stop and start sending signals at irregular intervals. As is such, creating a map over a large area for any large amount of time is wholly unsustainable with these costs. It is this problem that the system will resolve.

Project Objectives
The primary purpose of this project is to create a system for detecting both the presence of radio frequencies that are in use in an area as well as the strength of each of those frequencies using low cost of the shelf hardware. While there are existing solutions on the market for detecting RF signals, these solutions are not suitable for widespread deployment due to expensive hardware requirements. This system will solve this requirement by using low-cost energy detection hardware to collect data from multiple different locations in combination with energy interpolation algorithms to create a dynamic radio frequency spectrum map in real time and store snapshots of the data for historical purposes. If the system implementation is successful, the above data can be used for the end goal of dynamic spectrum access.

Features
A framework to support data retrieval from a wide range of sensors
A server architecture capable of provisioning minimalistic low-cost sensors
A visualization system with signal interpolation capabilities to simulate signal strength
An API that allows for customized historical data access and retrieval for additional analysis
A scalable architecture design to provision a large number of sensors

Technical Approach
The method to achieve this goal is through the use of the most simplistic sensors possible. By removing as much work as possible from the sensors, the development team can strip the hardware to the barest minimum necessary to read RF frequencies. The stripping of the hardware is accompanied by the creation of a provisioning system capable of managing the now simplified sensors. By utilizing cross-platform development technologies, a single provisioning system can manage an entire fleet of sensors on a network and can be deployed inside existing server hardware. This system then transmits the data further along for processing and visualization.

Design Approach
The solution to this problem is the result of a combination of several architectural styles. The data collection is handled in a distributed architecture, with the data collection happening at several different locations simultaneously. The data it transmitted until it eventually arrives to the web front end. The web front-end acts as a client/server architecture, with the web server itself operating under a model-view-controller structure. Lastly, there is a query API that will allow a user to query the database for historical data about the RF spectrum map.

Broader Impacts

The technology industry has utilized the electromagnetic wave spectrum, for transmitting data, for decades. Radios and televisions have long made use of that technology to stream radio and television shows. In today's world, not only do we use television and radio, but virtually everybody makes use of handheld electronic devices which connect to the internet via Wi-Fi. The Federal Communications Commission, FCC, regulates the usage of the radio wave spectrum so that broadcasters know which frequency to use and to minimize noise. These regulations specify which devices can broadcast over which frequency and what that bandwidth is.

The issue with this static design is its lack of flexibility. Today's technology landscape has changed dramatically from when radio waves were first used, but the FCC regulations have not exactly caught up. Broadcasting television is on its way out seeing that much of today's transitions are over fiber optics cable. That portion of the spectrum has not been reallocated and is highly underutilized. In contrast, the rise of cell phones and handheld electronics strains communications due their lack of bandwidth in the radio wave spectrum. Our development will hopefully lead to technology that will allow devices to dynamically allocate their own bandwidth on open frequencies which will elevate strain on the communication system.

Individual Motivation

Alexander Kinlen
Almost everyone uses at least one form of wireless device, whether it be a cell phone, laptop, or radio. When using one of these devices one of the most annoying things I have had to deal with is a poor connection. In my opinion it would be beneficial to be able to see if the poor connection is being caused by large amounts of radio traffic. This project seems like it would help the progress of improving wireless systems.

Due to my lack of computer skills, and the use of database management and web application development, this project seemed like it would be an excellent learning opportunity.

George Robbins
Technology has always had the need to transmit data. Data is being transmitted to be processed or to be displayed for some purpose. Whether the data is transferred through wifi or radio, most mediums involve the electromagnetic wave spectrum. The frequency map project intrigued me for several reasons.

First, this project will give me experience in a wide range of prevalent topics in the software industry. Virtual all modern software is developed to be used over a network or in the cloud. The frequency map will require the use of web server and developing a frontend for the user. The sensor will have to transmit the data via a UDP package, which will give me experience with networking protocols. Programming the microcontroller to control the sensors and transmit the data.

Second, I think this area of study could provide some useful insight into communications. Our study could provide a means to show that current model of communications are inefficient. The study could possibly provide alternatives and the data to back up them up. Furthermore, the study of dynamic frequency variations may provide a means to be able to predict future signal strength or current strength at locations without signals.

Lucas Higginbotham
As a user of several different wireless products that operate across a wide band of frequencies, I can see the value in being able to create a near real time map of frequency use. The primary motivator of this project, creating a map for dynamic spectrum allocation, would help ease network congestion and as a result make mine, and many others, experience using common wireless bands lives easier.

I had some of the skill set necessary to work a solution going into the project and saw the value of learning the rest of what I would need to complete it. Working with an embedded system will be a fairly new experience for me, and determining what architecture to use for the data processing system will give me some valuable insight into performance intensive systems. Finally, it will be my first time working on a project with predictive analytics, so I'm excited to see what I learn algorithmically in the field.

Roles in Development

Lucas Higginbotham

Provisioning System

Lucas has had a passing interest in networking for quite some time. He manages his house network from the access point out in its entirety and has been provisioning a headless SSH server for his remote friend group. This experience has given him experience working with networking firewalls and IP addresses. Additionally, Lucas has spent a considerable amount of time working to pick up the C++ language and has a pretty fair grasp of some of the newer features in the library.

Data Processing System

Due to the need for quick performance requirements, C++ was the language of choice for this system as well. As stated above, Lucas has had experience using the language and was the best fit for this section. However, in addition to his familiarity with the language, Lucas has also dabbled in parallelism in C#, Groovy, and C++, allowing him to have a basic grasp on the important concepts involved with threading. This experience will help him greatly when it comes to receiving the data from the provisioning systems and integrating it together into one complete snapshot.

Alexander Kinlen

Signal Interpolation

Alex was assigned the signal interpolation section because he was interested in seeing the actual math behind the system. Additionally, since the development of the database structure is not anticipated to take a long time, he can begin working on one of the most critical aspects of the system as early as possible.

Database

Because Alex had expressed an interest in learning database management, the ease of learning MySQL, and Alex's lack of experience in web development and programming hardware, we decided it would be best that he be in charge of the database development. Since it shouldn't take too long to create the database Alex will have to be in charge of researching interpolation.

George Robbins

Web Front-end

George was slated for the development of the systems web application. George is a good fit for this section of the project due to experience. He has held a co-op position at a local research and development company and has been heavily involved in developing web application at work. The projects that he has worked on involved implementing java servlets to exposing API that are used in simulations. One deals with plotting and tracking objects on an OpenLayers map which the main reason why our team choose to use that API for this project. George has expressed interest in learning alternative web development stacks. For a project in a different class, George learned and implemented a web application utilizing the MEAN stack. Expanding on that experience, he suggested to use that stack for this projects web application. With his experience in this aspect of

the project, our group should be able to develop a fully function application ahead of schedule which will free up resource to be used elsewhere.

Sensors

The sensor part of this project involves programming custom drivers for our USB dongle radio frequency sensor. Also, some lower level and microprocessor programming maybe need. George is familiar with C and C++ programming which the most commonly used programming languages for these applications. George is very comfortable with Linux having used Linux on his home server as well as with his work. With both considerations, George was a good selection for this portion of the project. A big factor in George's decision of selecting this project was to learn microprocessor and lower level programming. With George's experience in web development, he should have plenty of time to do the research and programming to get the sensor up prior to the testing deadlines established in the team's schedule forecasting report.

Architectural Overview

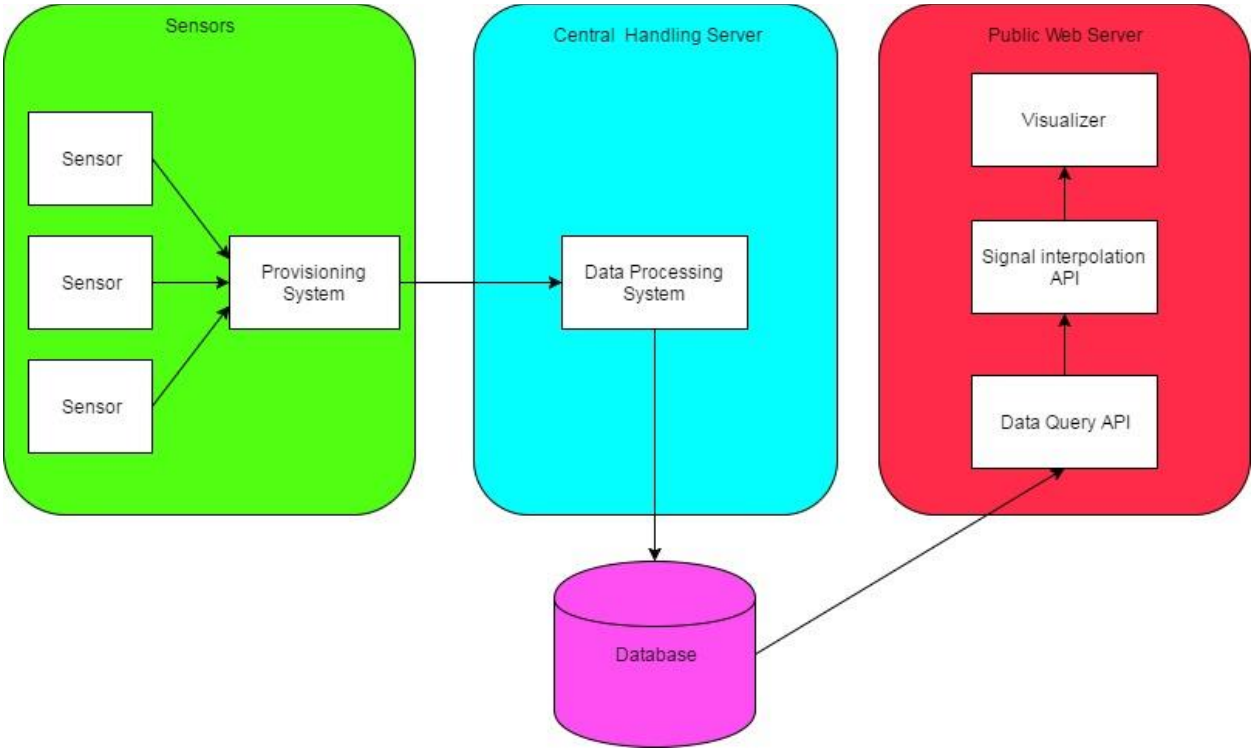


Figure 1: High level overview

The system will be divided into roughly three sections, the network of sensors, a central handling server, and a public web interface. The network of sensors will collect data on the radio frequencies in use as well as their strength and transmit the data in real time to the handling server via a User Datagram Protocol (UDP) connection. From there, the handling server will take the incoming data, collect and send it to the database. The public web server will visualize the data it receives in near real time and also have an interface to allow users to query historical data that has been recorded from the database.

The network of sensors is designed to be a low cost alternative to traditional RF frequency detection. The sensors themselves will be composed of a RF power detector capable of detecting somewhere between 40MHz and 2.4GHz wireless with a sensing threshold of between –100dBm to 10dBm. The sensor will transmit its received data to an onboard CPU that will process the data and convert it into a format to be taken in by our system. This data will then be transmitted via Ethernet to our central server. Additionally, our Ethernet connector will support at least the 802.11at standard, so we can power the device without an additional cable. Finally, the board may require a voltage regulation chip depending on the individual voltage requirements of the other components.

The handling server consists of three components, a provisioning system, a data processing system, and a signal interpolation system. The provisioning system will be responsible for detecting the

individual sensors, establishing a UDP connection stream to the server, and possibly time stamping the data. This system may have to be spun out of the handling server depending on certain hardware requirements. The data processing system will take the time stamped data from the UDP connections, spin it off into a thread, and convert it into a single format that can be passed to the database.

The public web server will consist of three primary components a system for visualizing a frequency map (think heat map) of all of the detected RF frequencies, a method to query the database for historical data collected by the system, and the interpolation system. The visualization system will allow users to customize what frequency ranges they're looking for to get an idea of what frequency bands are free. The method of access for the historical data will be done by an interface on the web portal. The signal interpolation system will take the single format for the data and interpolate signal strengths for areas that do not have a sensor placed based on reading from multiple sensor locations.

Sensors

The goal of the dynamic frequency map project is to display real time signal strength of frequencies across the radio frequency spectrum. As stated above in the project description and the broader impacts section, our project will help with communications by allowing people to study the dynamic nature of radio waves. To study these radio waves, our system is going to need sensors to capture the necessary data and transmit that data to the provisioning system for interpolation and processing. The detail construction of the system is described in early sections. This chapter will outline the sensor technical details and functionality. Also, previous iterations and design decisions will be covered in this chapter too.

Prototype

When our team first heard the pitch to work on this project, we were all under the assumption that the sensors were made and only needed to be programmed. Turns out that the sensor used during the project pitch was a mere prototype and much more work needed to be done in the research and development of the sensors themselves. The work that needed to be done to bring this sensor prototype to life was more computer engineering rather than computer science but our team set out to make this happen. The prototype, that was already established, consisted of a PCB with a voltage regulator and RFM22B/23B transceiver soldered on it. The prototype required a batter for operation and displayed frequency strength on a LCD screen attached to the board. Initially our goal was to program the microcontroller to cycle through the range of frequencies between 413MHz and 930MHz with a sensitivity of -120dBm. The voltage regulator receives the power from the battery and outputs 3v supply for the RFM sensor and a 5v supply for the LCD display. The sensor can transmit data using radio waves at the 915MHz frequency, however this method of transmitting data would likely interfere with the receiving duties of the sensor. Dr. Chatterjee tasked our team with coming up with an elegant solution to transfer the data from the sensor to the server.

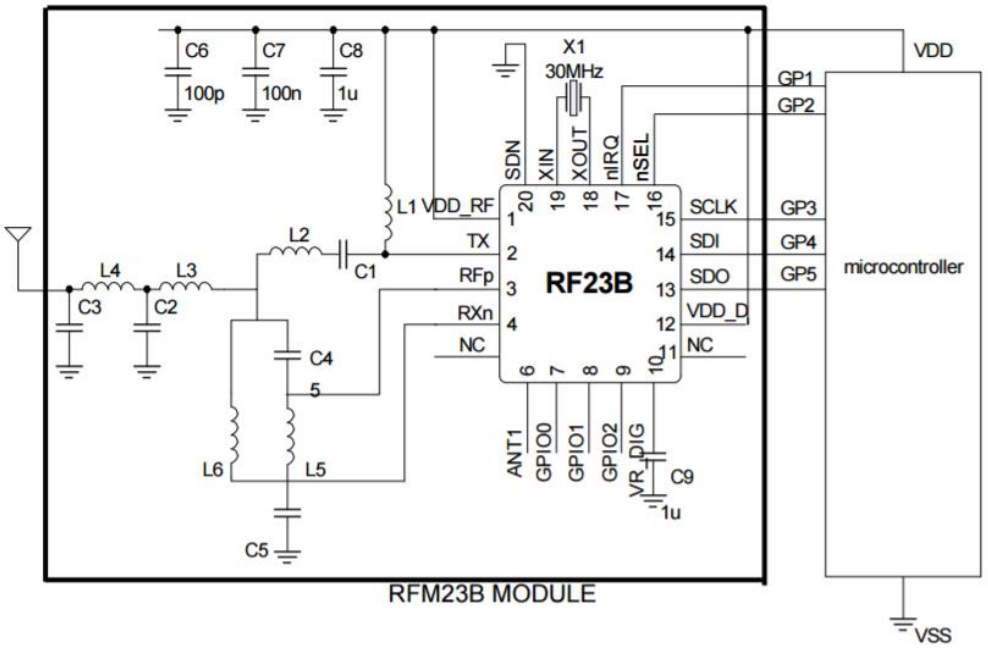


Figure 2: RF23B sensor used in sensor prototype

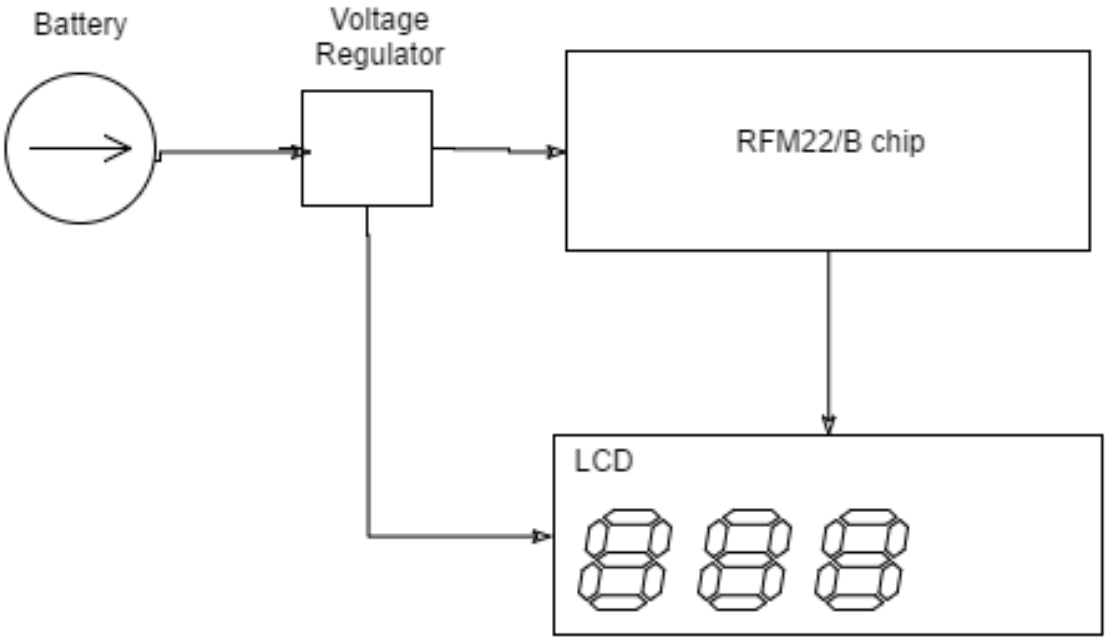


Figure 3: Basic sensor prototype block diagram

Ethernet Transmission

Our spent the next few weeks researching and discussing the alternatives to the problem of transmitting the data from the sensor to the server. During our research our team came across the

W5100 Ethernet chip. The purpose is to transmit data via Ethernet connection which is exactly what we were looking for. The chip fit with Dr. Chaterjee's budget and worked within the bounds of our power limitations. Implementing this chip with a new prototype would require a larger PCB board and re-soldering the components into place. The sensors must be stored in a static location and have access to an Ethernet port. It is not clear who would have been doing the soldering since that is well outside the realm of a computer science project but the W5100 seemed to be a viable option. For this sensor to be stationed in a static location, that location must be stored in the database for retrieval. Furthermore, assistance from UCFs IT department would have been necessary to establish the sensors with a static IP address so that the servers know what to listen for. Programming would have been necessary to package the data from the RF23B sensor into UDP packages for transmission.

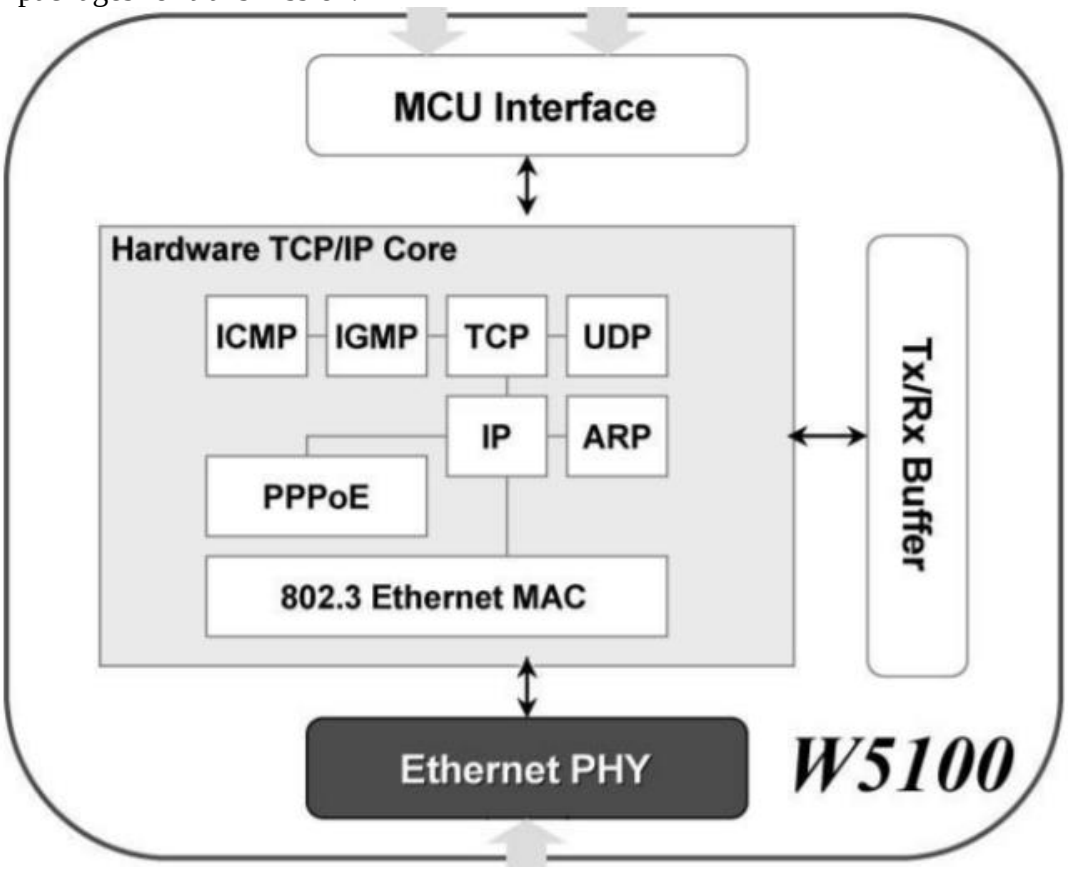


Figure 4: W5100 Ethernet chip block diagram

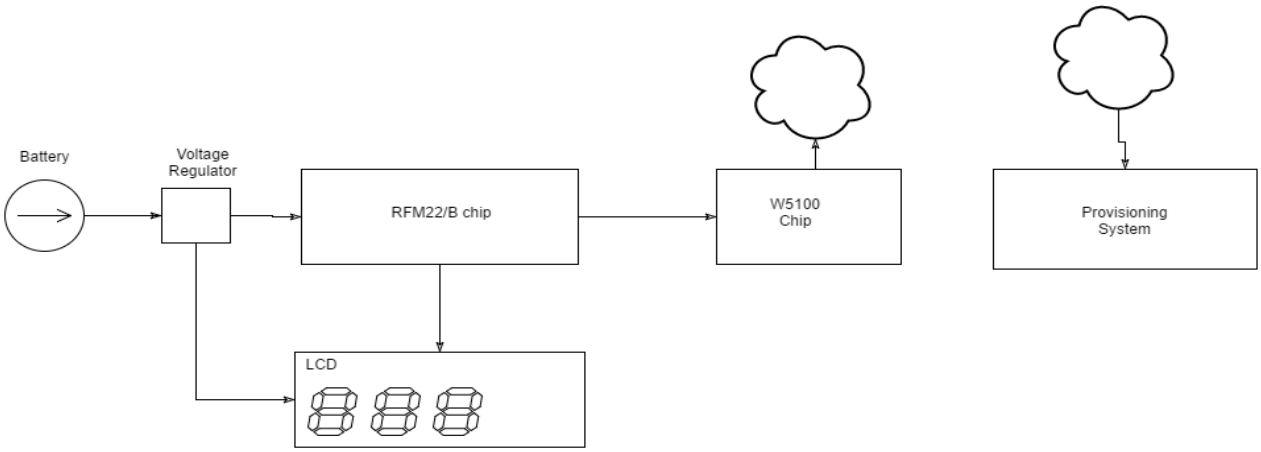


Figure 5: Prototype with Ethernet block diagram

Power Over Ethernet

The W5100 chip could have been used in our solution to the problem, however we were not done with our research. As we continued to research possible solutions, our team came across the Ag9800M chip. The Ag9800M chip offered the ability to transmit data over an Ethernet connection but it came with an added benefit. The chip is a power over Ethernet chip, commonly abbreviated as PoE, which transmits data and powers the device using one Ethernet connection. The appeal to using PoE is that the battery is removed. There will be no issue with replacing the battery of the sensor periodically which would minimize the cost to our sponsor. The Ag9800M chip can simultaneously power our sensor with enough power to operate properly and transmit the data to our provisioning server. The Ag9800M was within Dr. Chatterjee's budget and he seemed very interested in implementing this design. The issues with using the Ag9800M chip are the same with the W5100 chip. The sensor would be stationary which would require a static IP and constant connection to the Ethernet. The sensor would need to programmed to send the data as UDP package over the Ethernet connection.

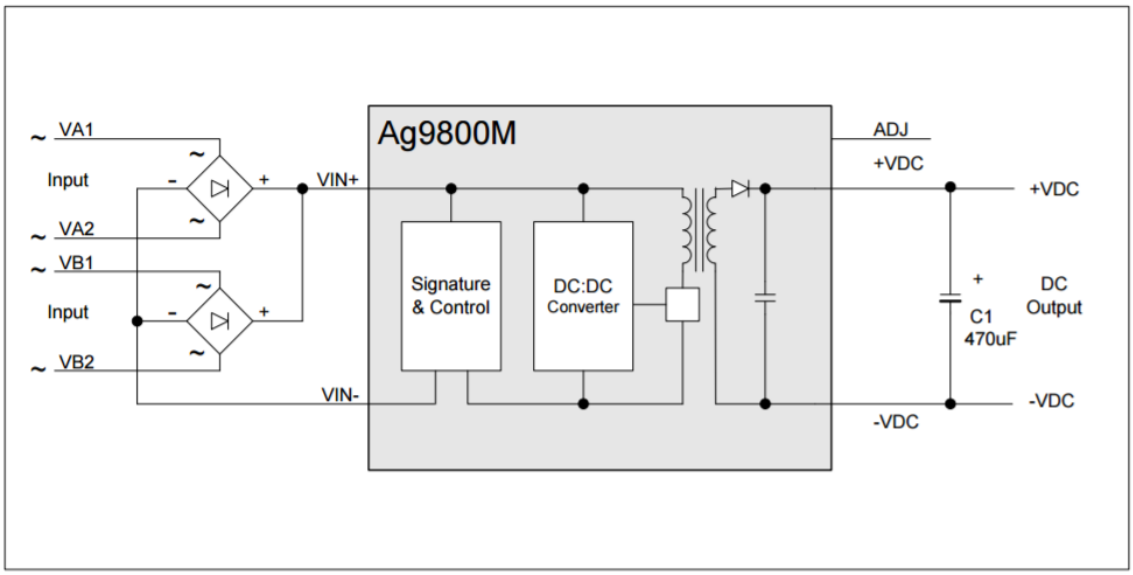


Figure 6: AG9800M Circuit Diagram

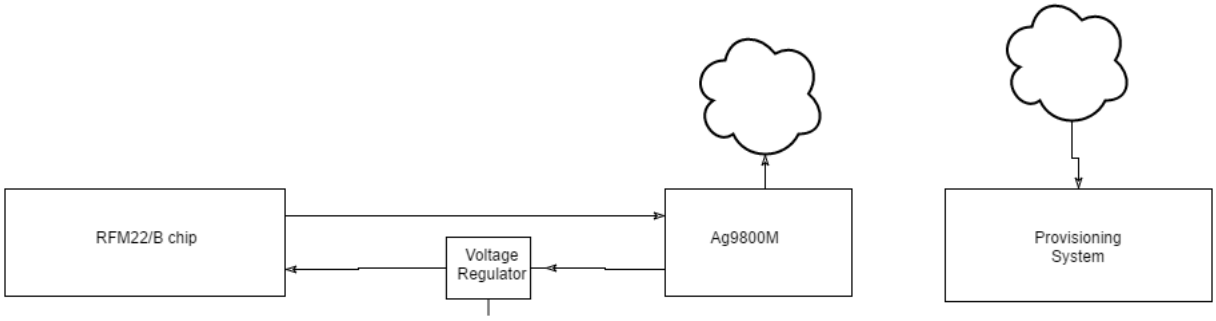


Figure 7: Power over Ethernet Block Diagram

WiFi and Mobile Sensors – "Want Requirement"

Once our team was satisfied that there was no other feasible option for transmitting the data from the sensors to the provisioning system we settled on the PoE Ag9800M chip addition to Dr. Chatterjee's prototype. During our next group meeting with Dr. Chatterjee, he expressed interest with the design that we laid out however there was a concern about the maximum frequency that the sensor could detect. Part of the study Dr. Chatterjee wants to conduct is the study of wifi band usage. The reason for the wifi range study is covered in detail in the broader impacts section. Wifi operates at much higher frequencies than the sensor currently used in the prototype. Wifi devices transmit and receive data on the 2.5GHz and 5.0GHz bandwidth which is well outside of the prototypes range. Our team then set out to find a suitable option for capturing wifi data points. Our team found the LT5538 detector chip which covers a much larger range of radio frequencies. The chip could capture data up to 3.8GHz. The range offered by the chip achieves one of the wifi data points but left the 5.0GHz out. Our team realized during our research that finding a chip that can reach those ranges was not realistic due to budget constraints. We brought this up to Dr. Chatterjee

and he deemed this acceptable given the situation. The chip would have to be soldered onto another PCB board and programmed to transmit the data in UDP wirelessly. The LT5538 chip operates within the voltage range of the battery from the original prototype design. Having wifi capabilities gave the sensor the ability to be deployed as a mobile sensor which eliminates the PoE chip from the mobile design. Having a mobile sensor is not one of teams "must have" requirements established at during our first meeting with the sponsor. Dr. Chatterjee's motivations for wanting a mobile sensor is to study the wifi range which has become one of the most bogged down bandwidths due to technology trends over the last decade or so. Furthermore, Dr. Chatterjee would like to know if it were possible to estimate the location of a mobile sensor moving around within the area between stationary sensors by comparing the mobile sensor signal strength data to the interpolated data from multiple stationary sensors. Our group would like to tackle an interesting problem such as this but we are unsure if this is possible due to time constraints. For that reason, implementing mobile sensors and location estimation has been classified as a "want" requirement that we may attempt but are not obligated to do so.

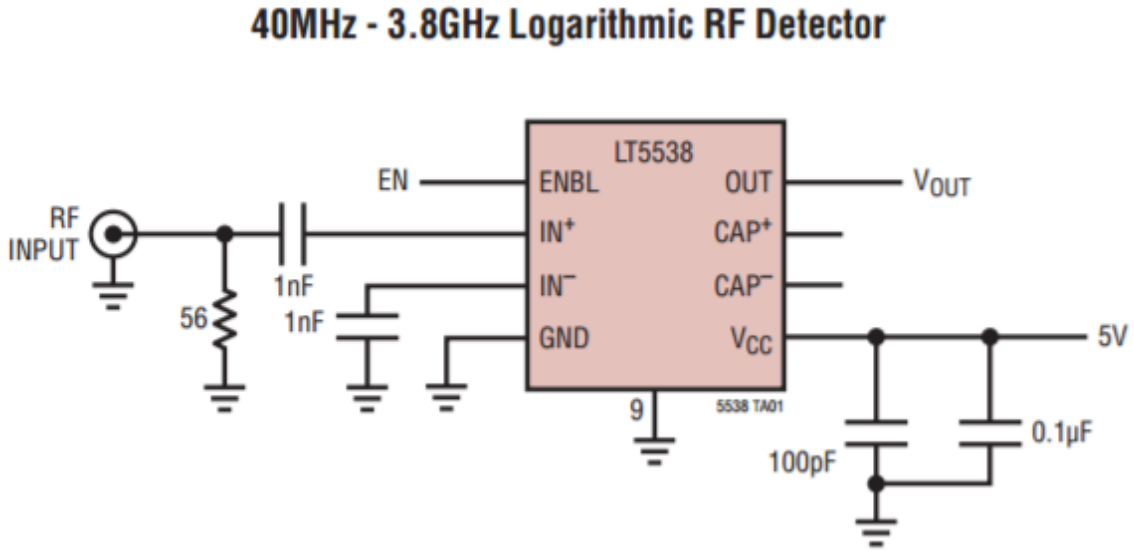


Figure 8: LT5538 circuit diagram

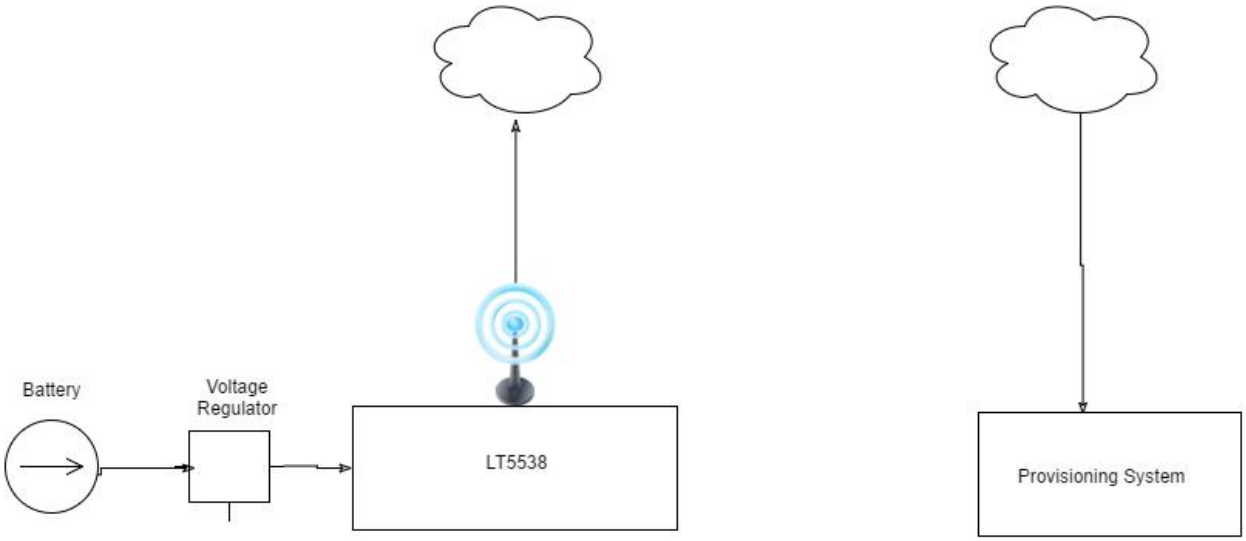


Figure 9:Mobile sensor prototype block diagram

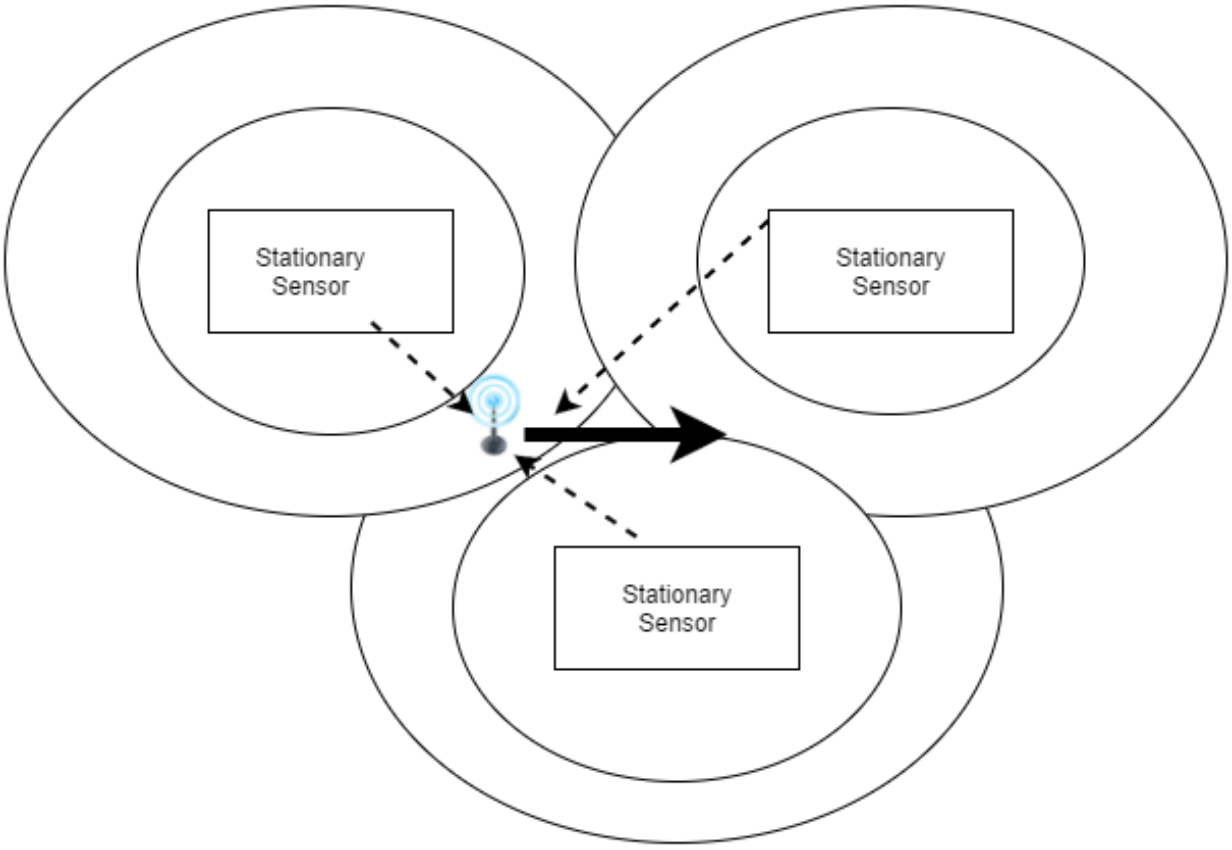


Figure 10: Location Triangulation

Final Design

Despite having research and designed sensors that could meet most of Dr. Chatterjee's requirements, there were some issues that needed to be addressed. The sensors could not study the 5.0 GHz band which was mention in the previous section. In addition to that, the chips would need to be soldered onto a new PCB board and protected by a case before our team could deploy them. The soldering would likely be done on Dr. Chatterjee's end, but our team still need to program the sensors to cycle and transmit the data. The components of the prototype were chosen partly due to cost considerations. During our research, our team noticed that any of the better chip sets were well beyond the budget of the project. We also noticed that the there were two main differences separating the better chip set and the ones we chose. The first difference is the range of frequencies that the chips can monitor. Any chip we found that monitored up to and beyond the 5.0GHz range was much costlier. The other difference is the sensitivity of the sensors. The RF22B chip used in the original design had a sensitivity level of approximately -120bBm which is the target range that Dr. Chatterjee was looking for. Our research turned up sensors that could reach the 2.4GHz band, however did not meet the sensitivity standard. The sensors that we found that did meet the levels of dBms, that Dr. Chatterjee was looking for, were far beyond the budget. For these reason, Dr. Chatterjee changed course and issued our team two of the R820T tv tuners. These tuners are used often by software defined radio hobbyist. Software design radio is a hobby that people get into in which tv tuners and other sensors are used to monitor the radio frequency spectrum. The R820T tv tuner simplified the project because there is no need to purchase microchips or solder them onto new PCB boards. No testing will be required to check to see if the sensor work properly. The R820T tuner also is relatively cheap so Dr. Chatterjee could purchase multiple tuners.

The design of the prototype and our team's tasks changed significantly. The R820T tuner is a USB dongle with and antenna which requires that the tuner be plugged into a device with a USB port. We have decided to use a Raspberry Pi for this purpose. The Raspberry Pi would have the Ubuntu Mate operating system which is a lite weight Linux distribution. The R820T tuner has drivers and software that must be ran on Linux so this was a convent aspect of the Raspberry Pi. Our team has since shifted the approach to the problem due to the drastic change. We now must modify the R820T drivers and open source software to allow the tuner to cycle through the radio frequency spectrum. The tuner will feed the data to the Raspberry that will send the data to the provisioning system. The television tuner offers a little wider of a band, covering up to 1GHz but the benefits of the implementing this design outweigh any negatives. By approaching the problem in this manner, our team will implement the foundation of the system which will allow Dr. Chatterjee time to improve and implement a suitable sensor that can just be plugged into our infrastructure.

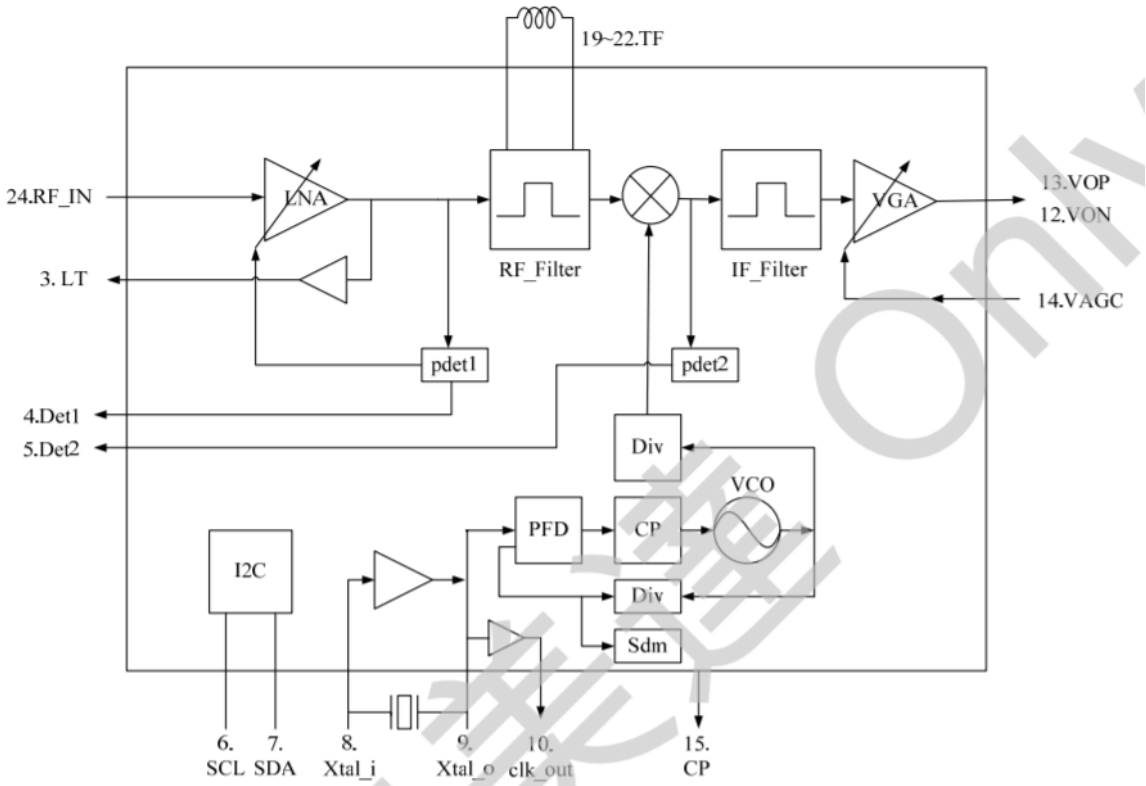


Figure 11: R820T circuit diagram

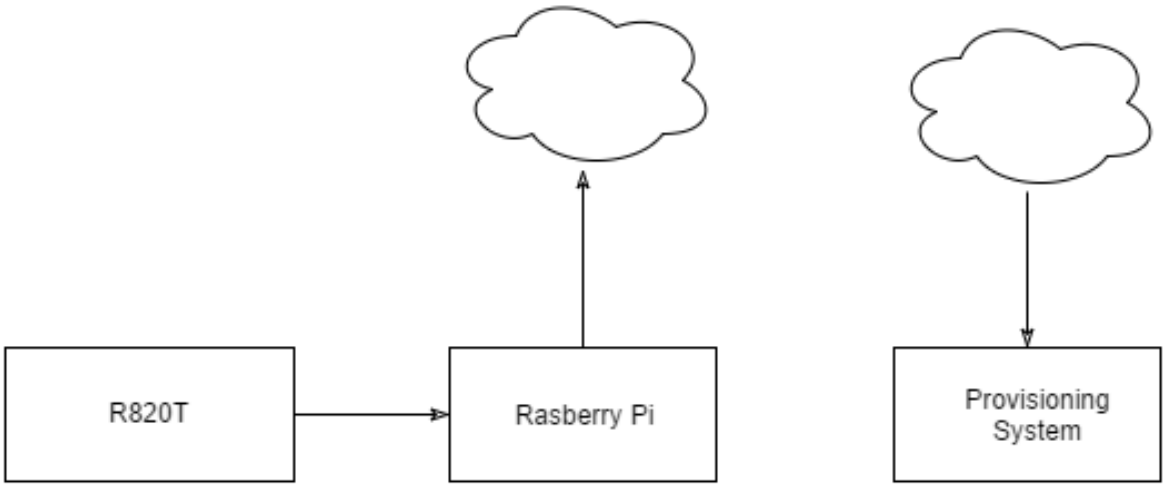


Figure 12: Final design block diagram

Below is the actual appearance of the current radio frequency detection hardware we are using. As you can see, it has the requirement of needing an additional system attached to it and cannot run self-sustained.

Figure 13: Actual R820T USB Dongle

Provisioning System

The provisioning system acts as a bridge for the data being transmitted from the sensors to the server. The intent of the system is to allow the overall architecture to be relatively hardware agnostic. As long as the sensors can transmit data via the format expected by the provisioning system, the actual hardware does not matter. As a result, this will allow the system to be scalable across both multiple frequencies and at different sensitivities. The additional customization that the hardware level also allows for a more flexible budget when it comes to deploying the system.

The provisioning system will exist within each network or subnet that the sensors are deployed (see fig). This is different from the original proposed design where the provisioning system would exist once on the central handling server. The change in the design occurred due to limited hardware specifications. Due to budget restrictions related to the need to mass deploy the hardware, there is a limited amount of processing power native to the sensors. As a result, the decision was made to restrict the sensors to simply broadcasting their data to the network. If we had kept the previous model, the network information would have to have been hard coded in each sensor, which would have resulted in scalability problems.

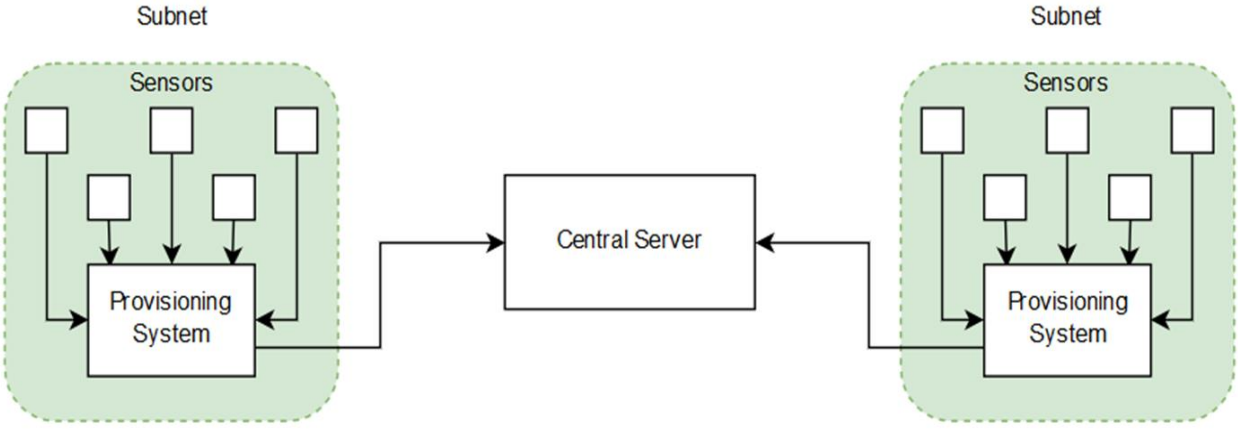


Figure 14:Overview of architecture surrounding the provisioning system

After a provisioning system has been installed onto a network, it will adopt a listening mode where it waits to receive a signal from a sensor. Each sensor will be programmed to begin broadcasting data to the provisioning system when it is installed for the first time. These broadcasts will occur in a fixed-interval time as to not bog down other devices on the network with useless traffic. When a provisioning system detects the associated signal for a sensor, it will begin a process to add the new sensor to its list of active sensors. The provisioning system will add the sensor to a tracking data structure inside the system as well as a tracker that determines when to drop the sensor from the list of active sensors. The IP address will be stored as a command line argument on the sensor, so if the provisioning system is disconnected or moved it will be able to update to a new IP address without having to recompile the client software. The sensor will then begin to stream its collected data to the provisioning system.

Additionally, the first time a sensor is added to the provisioning system, the provisioning system sends a check to the database located on a remote server to determine if the new sensor id is currently tracked by the rest of the system. If the database call returns empty, a new sensor is

added to the system at the provisioning level. The reason this check is done at the provisioning level is due to how messages are transmitted to the server at the handling level. At the handling level, incoming IP addresses are treated as new provisioning server so it would require costly checks on each incoming packet to determine if it is a new sensor.

Data that is transmitted from the sensors to the provisioning system will be in a JSON format. This data will be read by the provisioning system as it is received from each sensor. This read will occur across a single thread in an event loop. Each cycle in the event loop will read one message of the radio frequencies being transmitted by the sensors and store those read values in a global buffer managed by the system. The global buffer will either wait for all of the incoming frequencies from one full frame of the sensors or wait until a set amount of time has passed. Once either section is complete, the combined packet will be sent from the provisioning system to the central processing server.

In addition to processing the data, the provisioning system will also have to allow for the management of the sensors. The system will have to detect any untethered sensors currently on the network and begin capturing data broadcasted from that network. Once the sensor has been captured by the system, some rudimentary configuration options can be set to handle basic management of the data being received. The server further downstream may not need all of the wireless spectrum that the sensors can detect, so the provisioning system will have the option to drop traffic to prevent it from being transmitted. This can help to reduce the overall network traffic being produced by the sensors. Additionally, the provisioning system will have to be able to detect when a sensor is no longer processing. When such an event is detected, the system will notify the administrator of the system, either by notifying the system administrator with a logging message in standard output. Additionally, the provisioning system can remove transmitting sensors from its monitoring during a partial decommission to avoid false error reporting and reduce log bloat.

A final critical aspect of the management in the processing step is validating all the data that was received. Since data is being transmitted via UDP, there is built in measure to track packet loss. As a result, each collected piece of data will have to be scanned for validation. This will be done by checking each JSON key value pair as well as the validity of the structure of the JSON. In the event that the data is determined to be damaged, it will be flagged for handling further upstream.

After all of the validation has been handled by the system, the collected snapshot of the data will be transmitted to the next stage in the cycle. At this stage, the speed of data transmission outweighs the need for data integrity, so the data will still be transmitted via the UDP protocol. The transmission location will be in found in an editable configuration file on the machine hosting the provisioning server. Since the provisioning server is a full machine, it is feasible to allow the transmissions to be directed how the client wants. An overview of the process flow is seen below (see figure).

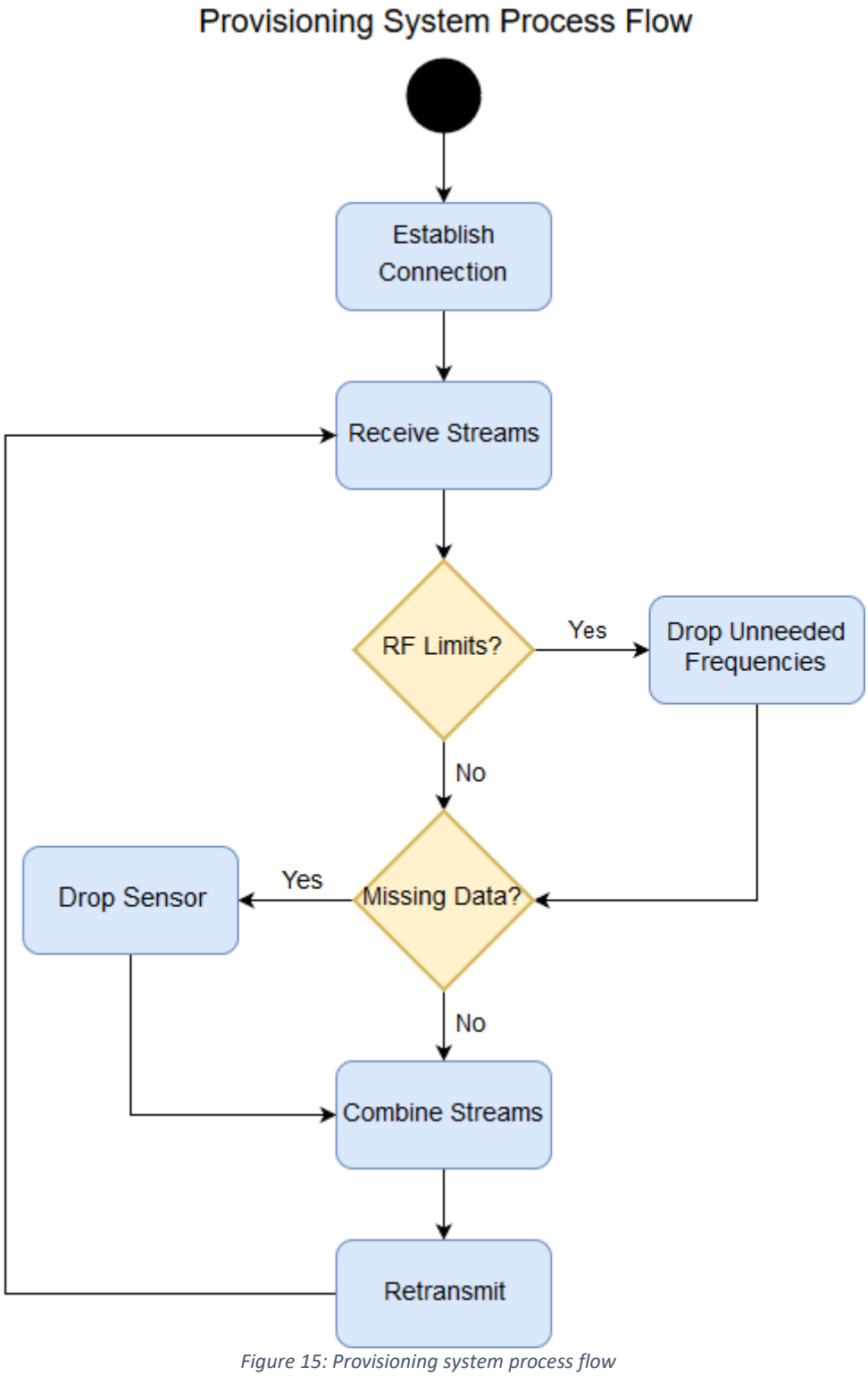


Figure 15: Provisioning system process flow

Data Processing System

All of the data collected by the provisioning systems will be transmitted to one central handling server. This data will be received by the data processing system. Some of the core functionality of the data processing system is very similar to the provisioning system in that it will collect data being transmitted from multiple sources and then combine that date into a single source file. Once the data is collected, it will need to perform a couple different actions.

When the data is collected into one object, it will need to be converted from a string format into a data type to allow for calculation to be performed. The data itself will be converted via JSON deserialization. When this is being done, all of the data in the stream will be checked one final time to determine that it is all valid. Since the data is being transmitted as a UDP stream, there is a real possibility of partial receipt of transmissions. If a stream is determined to be incomplete, it will have to be parsed to determine what data (if any) is salvageable. In this instance, the data will be flagged as incomplete and dropped from the current frame. The data that is not salvageable will have to be marked as a point of prediction instead of a point of reception for the signal interpolation system further downstream. Additionally, if data from a particular source has this happen too many times, it will be flagged for review by the administrators of the system.

In the event of a system failure further downstream, an administrator will need to be contacted to rectify the issue. The method of contact will be via email, with simple single failures sent as standard messages and entire provisioning systems sent with high importance. Administrators will then have access to their own systems and be able to investigate the issue.

The other task the system will be responsible for is handling historical radio frequency images. At a set time interval, the system will take snapshots of the current frequency strengths being read by the sensors. This data will be pulled from the stream, converted into a format that has easily query able fields, and sent to the database for long term storage. If a section of the snapshotted stream is marked as damaged, the system will keep the current snapshot in temporary storage and will attempt to find an undamaged portion of the stream over a fixed interval. If it is unsuccessful, it will store the original snapshot in the database, with a flag that denotes it as incomplete information and additional information about which data stream was incomplete. This task will be spun off into another thread to avoid being a blocking operation for the data combination task.

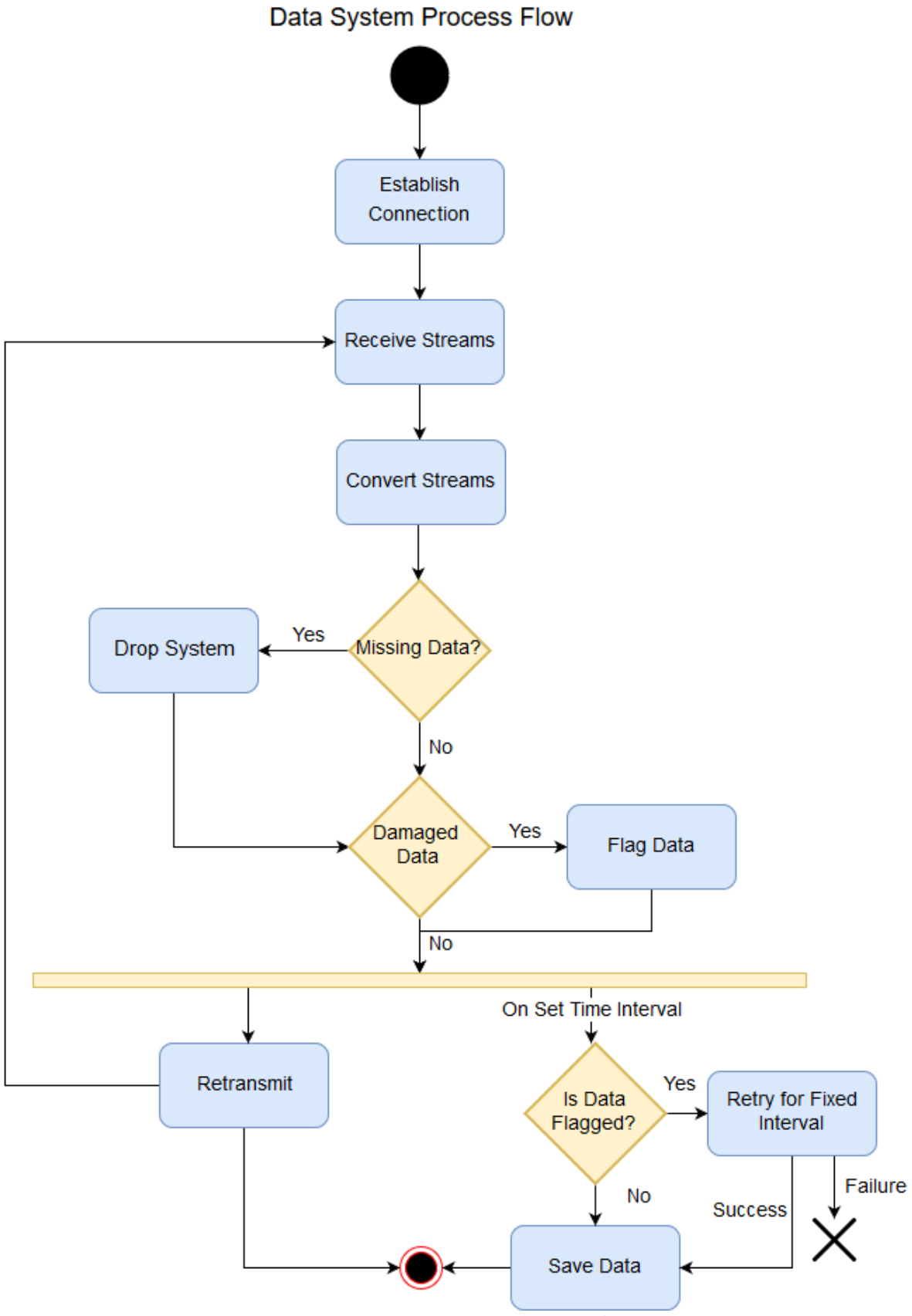


Figure 16 Data Processing System Process Flow

Signal Interpolation

The signal interpolation step is perhaps the most important step of the entire process. This is the step where all the data that has been collected by the sensors and transmitted to a signal location is processed to create the data necessary for the visualization. At this point, every signal a sensor has received is compared to the signal strengths received by all the other sensors in the network. Using this data, the system begins to calculate interpolation points via GPS coordinates to create an estimate for the signal strengths in areas where a sensor is not located.

Each frequency cycle will involve the following:

- Determining the position of each sensor that reported that frequency
- Triangulating the exact positions between multiple sensors
- Using the triangulation to determine the directionality of the frequencies
- Then calculating the actual interpolations in areas where no sensors are located

An example process flow of the system can be seen below (see figure).

Interpolation is a math based tool that's main purpose is to estimate the value of a certain point using existing values from other points. This will aid us greatly in producing values for areas inside where our sensors are on the map. Thanks to our sponsor we now have an equation that will help us with the creation of our heat map.

The following equations supplied by our sponsor Dr. Chatterjee.

$$\phi_q^I = \frac{\sum_{i=1}^{|\Delta|} (d_i^I)^{-k} e_q^i}{\sum_{i=1}^{|\Delta|} (d_i^I)^{-k}}$$

K is a constant that is determined based on the area where are observing. Since we are not in a heavily vertical terrain such as New York City, we will be using 2 for our constant k. d is the distance from the estimated point to the reading we are looking at. E is the measured power form the sensor.

To make sure the estimated points are inside our area of sensors we first run a graham scan on the sensors we are looking at so we can get a 2-dimensional polygon. From there we can iteratively check if a given point is within the polygon. Now that we have a list of points to estimate, we can then insert those points into the given equation and send those points to the front end.

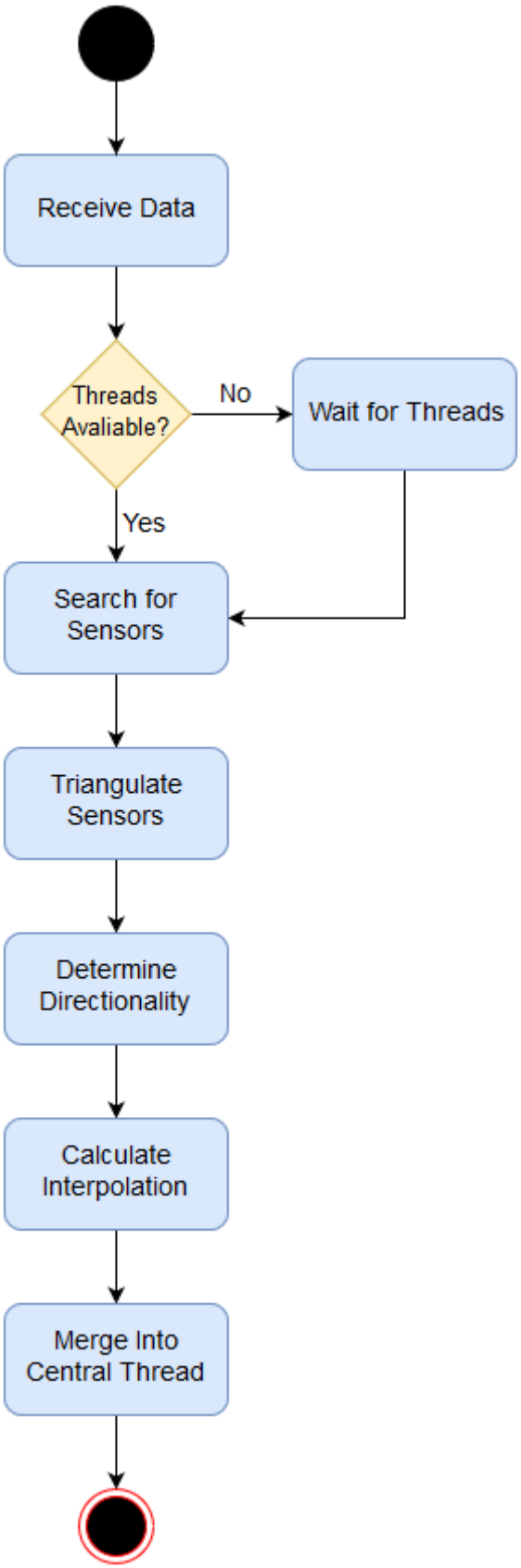


Figure 17: Signal Interpolation Process Flow

Web Application

Overview

Software developers have many approaches and techniques to use when developing a piece of software. Each approach has benefits and limitations which the developer must weigh prior to designing the software. These also are dependent upon the problem that the developer is attempting to solve. The developer must keep this in mind during the research phase otherwise the finished product may not be best suited for that instance. Our group set out to find the most optimal solution for the research that Dr. Chatterjee wants to conduct. Dr. Chatterjee established requirements that dictated which paradigm we were going to use. The most relevant requirement related to the design of our software is that the software must be able to process and display radio frequency data in close to real time. Using this requirement, we determined that our sensors would be streaming data as they cycled through the radio frequency spectrum. To capture these constant streams, we determined that the client – server development pattern most naturally modeled the requirements set forth. The central handling server handles the incoming stream which is cover in previous chapters. The other essential aspect is accessing these streams of data which also requires a server. We decided to separate the data processing and client access into autonomous systems because the data processing requires significant computer resources. The focus for this chapter will be the server handling requests made by clients viewing the data streams.

Building a client – server architecture can happen a few different ways. One option is to build a standalone application that resides on user's machine. The desktop application approach is an older one which becoming less prevalent since the emergence of the internet and the vast improvements to web browsers. A desktop application in our system would have to make the same calls to the backend API but there are several drawbacks which steered our group away from developing one. The biggest problem with desktop applications is that the application would have to be installed on the client's machine. If our team choose this path, then the final product is highly dependent upon the machines hardware, operating system, and the programming language used for development. The build and installation procedures would have to be address for any operating system the client would likely be using. The management of dependencies on the client's machine is also an important concern that our team would have to address.

For these reasons, our team decided to develop a web application to access the API for visualizing the constant streams of data. Developing a web application for the dynamic frequency project has many benefits which appealed to our team. Managing dependencies shifts from the user's machine to web server. This is ideal since we have little knowledge of what the user's machine specifications are. Furthermore, our team did not want to limit Dr. Chatterjee's options in the future, if he chooses to expand on our work, by requiring certain versions of compilers, operating system, or any other dependencies that must be present on the user's machine. The only requirement with the web application is that the user must have access to a web browser but this assumption is a reasonable one given the current state of the Internet age. With our software residing on a remote server, updating dependencies will not affect any machine that is attempting to connect to our service. Another benefit with building a web application is that the user can access the API any number of ways. Dr. Chatterjee will have the ability to access our software on a tablet, phone, or computer with any operating system.

Having decided to approach the project by utilizing a web application, our team faced a few options for development. Web applications are built with a collection of tools that are used in conjunction for the final product. These tool sets are commonly referred to as stacks and are specifically choose together for their ability to be connected into one standalone application. The LAMP stack is a commonly used web stack. It consists of a Linux server, an Apache web server, MySQL database, and PHP. A LAMP stack requires a Linux machine being run remotely that houses an Apache web server such as Tom Cat. Linux and MySQL are both tools that we are using for the dynamic frequency, so the LAMP stack could have been a viable solution. Our team choose not to utilize the LAMP stack for a few reasons but none that really have to do with the limitations of LAMP but more with the benefits of a MEAN stack which will be discussed later in the chapter. Another notable option for the web application stack is Ruby on Rails. Ruby is backbone programming language used to build rail applications. Rails is the framework used to route requests and serve ruby view to the client. Our group briefly considered this stack but quickly discarded it. We did not want to deal with the learning curve of ruby or the rails framework. Furthermore, Ruby on Rails adds a lot of overhead to the application which is not ideal for a responsive map displaying real time data streams. Ruby on rails does have a small niche in the web development industry but is becoming less used due to its added bloat of the rails framework.

After eliminating those options for the web application development, our team decided to use the MEAN stack. Traditionally, MEAN stack combines Mongodb for the database, Express for the back-end application framework, Angular for the front-end framework, and Node for the web server hosting the application. Due to the inherent relational properties of the data streams from the sensors, we are replacing Mongodb with MySQL. Mongodb is a NoSQL database that stores and queries documents. Many applications utilize this methodology for storing collections of documents like discussion threads on a social media website. MySQL offers our application the ability to relate a data stream to a specific sensor which then can be queried based on these relations and other constraints. The database specifics are discussed in another chapter. There are many benefits of a MEAN stack architecture that appealed to our project and ultimately elevated MEAN to the top of our list. One advantage of a MEAN stack is that the entire development is done JavaScript. JavaScript is traditionally used to program functionality into HTML documents. However, Node JavaScript framework allows developers to build the back-end web server in JavaScript as well. This advantage will help the development team because the developers will not have to jump from one language to another. MEAN is also highly scalable unlike many of the other stacks. Most web application stacks rely on the web server to spawn threads to handle incoming requests. The issues with this approach is that the hardware and operating systems place restrictions on the number of threads the webserver can handle. To handle high volumes, developer typically build a separate load balancing server which tracks the loads on servers across many machines and then issues the request to the web server under the least amount of load. Node handles this problem differently. Node receives an incoming request, then passes it off to the appropriate portion of the application for processing. To do this without threads, Node issues promises to the client application which get fired once the response is ready. This approach is unique to Node and is the primary reason to why Node is so highly scalable.

In the MEAN stack, once the node server receives the request it is passed on to the Express application housed by Node. Express is another back-end JavaScript framework that establishes

routes to handle various requests. Express evaluates which route the request must take and then executes that portion of the application. Once Express has finished processing and the response has been built, it sends the response through node to the client which then executes the promise that was initially issued. In MEAN stack, the client will be the Angular application.

Angular is a front-end JavaScript framework that interacts with user, HTML, CSS, and any other JavaScript library used in the clients' browser. Angular applications are built using the model, view, controller paradigm. The developer creates models that make HTTP requests to the Node server. These models are known as services in the Angular application structure and provide a nice way to ensure the modularity of the application. Angular views are the HTML and CSS documents, and in our application these documents will also consists of our OpenLayers map API, bootstrap, and Angular-chart. Angular offers its applications a few nice features for implementing the user interface. Angular uses its own predefined HTML tags so that the developer can access Angular variables and use programming features like loops and conditional statements right inside the HTML. Also, Angular tags can make function calls and pass parameters directly to methods inside the controller. Our team also liked that Angular provides two-way databinding between Angular HTML objects to variables that are set via promises from the Express server. This makes updating data in tables and sending filters through to the server seamless since the developer does not have to build specific objects watching for changes. Angular controllers establish a connection between the HTML documents and the Angular services. Angular establishes these connections using routes which is consistent with how Express applications are built.

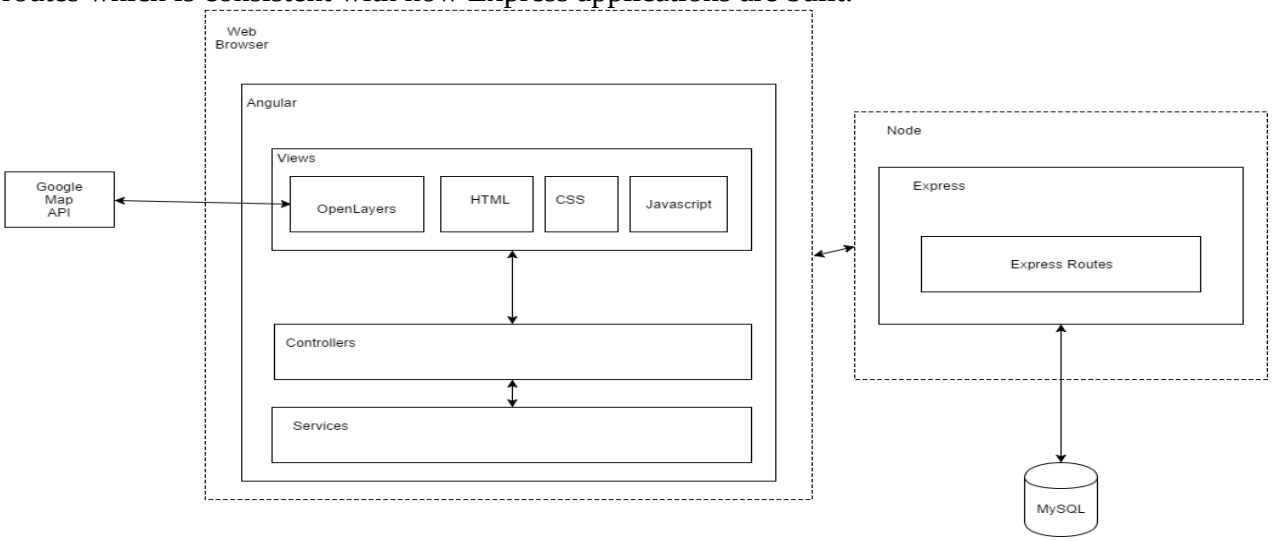


Figure 18: MEAN Stack

User Interface

Web applications are typically developed in a modular approach. Developers aim to keep back end API separate from the front-end code for troubleshooting, maintenance, and security considerations. The front end of web applications consists of the HTML, CSS, and JavaScript file that are executed by the browser. Together these files provide a nice user interface to the services provided by the Dynamic Frequency Map system. The detailed description of how Angular works in conjunction with the MEAN stack architecture is covered in the previous section. This section will cover wireframes, Angular routes for our specific application, OpenLayers, and other dependencies.

User connecting to our API will first be sent to our landing page, see Figure 3.2.1 The connection will occur when the users web browser requests index.html from our server. The web server will automatically route users to the landing page if the URL in the browser's navigation bar is the servers IP address and port that the server is listening on. Tentatively the address to our web application is http://10.171.204.163 which is hosted at UCF senior design lab. The port in which node will be listening on is 8080, however this may change. All communications to the webserver will take place with the HTTP protocol which provides clients with GET, POST, PUT, and DELETE requests for the webserver. Our web application will be utilizing GET and POST since we have no need to allow the client to manipulate the data in the database which is what the other HTTP requests are used for. The clients can navigate to our web application's landing page by navigating to http://10.171.204.163:8080 which Angular will route to index.html. The landing page provides a nice minimalist view welcoming the client to our API. In the wireframe mockups, you can see that the landing page gives the user a brief one sentence welcome to our API. If the user wishes to know more about what the API is all about, then they can navigate to the web application's about page using the "Learn More" button but that will be covered later in this section. Notice at the top of the page is the navigation bar. Using a template for each page in our web application, the navigation bar will always be conveniently located at the top of the page providing easy navigation for the user. The navigation bar is also designed with the minimalist approach in mind. The goal is to not overwhelm the user with excessive clutter so we provide the user with the most important sections via the navigation bar. Currently the user can navigate to the live and historical data sections via the "Live" and "Historical" button located on the navigation bar.

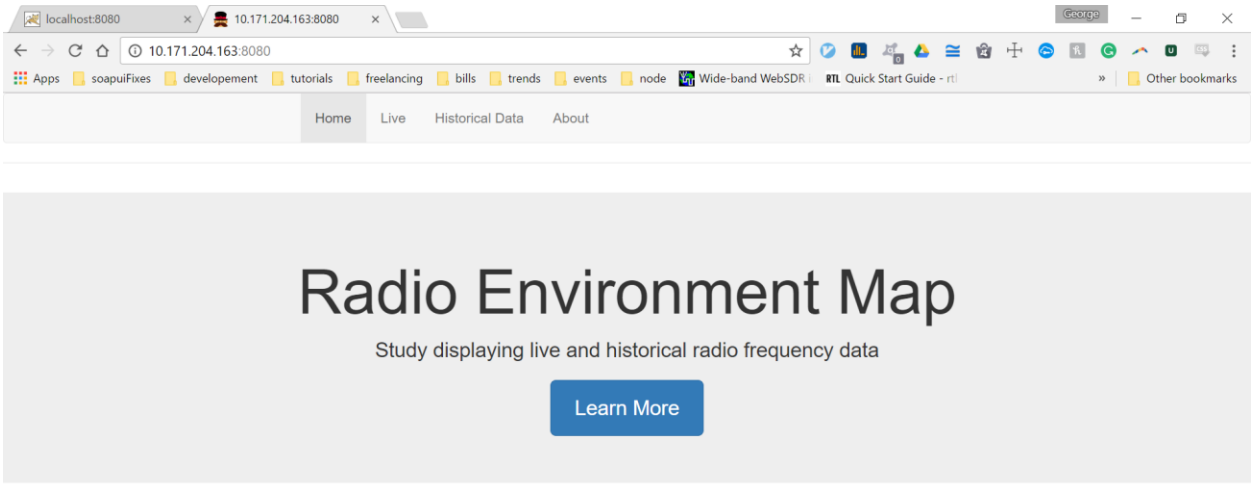


Figure 19: Index.html - Landing Page Mockup

From the landing page, the user is face with a few options to navigate to. Our team thought it would be a good idea to offer some information to any user who may want to find out more information about our project. The "Learn More" button navigates the user to an about page detailing some of this information. The about page can be access through the URL http://10.171.204.163:8080/about.html which were Angular routes the user based on the button click event. The about page uses the same minimalist design and color scheme as the landing page. From the about page users can read about our mission statement, broader impacts, and our sponsor. The mission statement provides a brief explanation of the project. The explanation is not intended to be an in-depth analysis of what we are doing but it's there to offers some insight into the scope of our project. Next the user can read about the broader impacts of our study. The broader impacts section consists of an excerpt from the broader impacts of this document. Our group included this to offer some perspective into why this research could be import and applicable to everyday communications. Lastly, our group wanted to inform the user of our sponsor. The description of Dr. Chatterjee was pull off his UCF page. Also, there is a link at the end of this section to provide the user a means of communication in the case that the user may want to collaborate on this work.

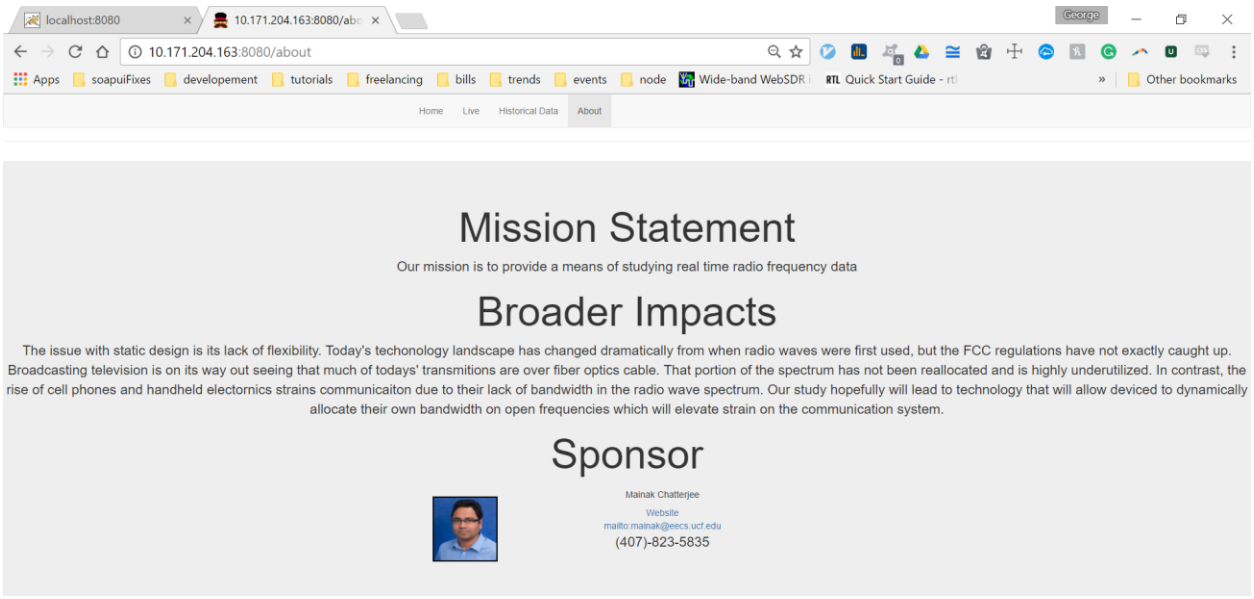


Figure 20: About.html Mockup

Now that the user has been greeted and had the opportunity to learn more about this project, they'll want to navigate to where they can use the data stream portion of the API. The user can navigate directly there from any page using the navigation bar. The two options on the navigation bar are for the live or historical data streams. Angular routes the live button click event to the http://10.171.204.163:8080/live.html which can also be access by entering in this URL directly into the browser's navigation bar. On the live map page, the user will have the option to navigate to the historical data section through the navigation bar. The live page has a lot going on which made it hard to follow the same minimalist design of the previous pages. However, this is unavoidable due to the need to filter data, display the data, and sensors. The color scheme used does match the other pages giving the user a nice consistent interface. On the left section of the page are the filter options. Here the user can select which frequency to display on the map by either using the slider. The next filter allows the user to display data from a single sensor or groups of sensors. By default, all check boxes will be marked and all data will be displayed on the map. It is important to note, that the filters have no effect on the sensors themselves nor on the data collection part of the system. Instead the filters are selecting only what subset of data is being displayed. The entire set of data will be still be available despite the filter settings. The filters are bound to AngularJS data structures that will allow the web server to automatically apply the filters. The rest of the page is designated for displaying the data to the user. Our web application will display the data in two forms.

The first form is the heat map which will be displayed as an overlay on an OpenLayers map. OpenLayers will be covered later in the dependencies section. Angular will maintain javascript variable that are bound to the map. These variables will be updated by polling the node API at regular intervals. The data that will be displayed will consist of the data coming from the sensors and have been stored in the live table of the database. The webserver queries the database and passes the necessary data to the interpolation child process. The interpolation child process uses a

flood fill algorithm to calculate an estimated value within the area between sensors. In figure 3.2.3, the location of the sensors will also be displayed on the map. This may be useful to have if Dr. Chatterjee expands on the project, at a later date, implementing mobile sensors.

The second for of data that is display on the live map page is located at the table below the heat map. This table displays the actual values received from the sensor stream. No interpolation or prediction will be displayed in the table. The table give users a means of troubleshooting or verifying proper operations of the system. For instance, if the user determines the heat map may seem to be abnormally low for the frequency band in question, that user can quickly scan the sensors output to look for abnormalities such as sensor shut off, or stop cycling. The tables also display physical location of the sensors and date of last transmit.

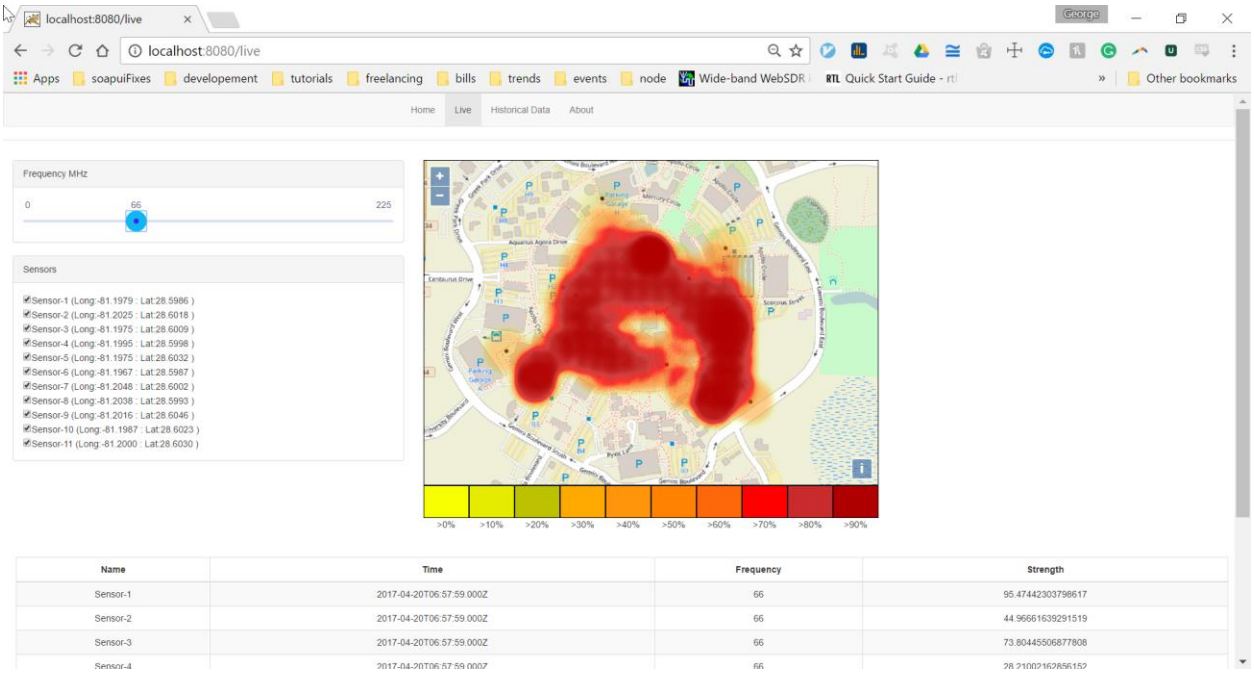


Figure 21: Live Map Mockup

The user will have the option of viewing data that has been recorded from a previous data and time ranges. The historical data page is accessible through the navigation bar, at the top of the screen, by clicking the "Historical" button. Angular routes the user to the historical page at http://10.171.204.163:8080/historical.html upon the historical button click event. The historical data page is almost identical to the live stream page. On the left side of the screen are the filters the user can set to filter out the data being displayed. As mentioned before, these filters do not effect sensor or data processing parts of the system. The historical page keeps the frequency and sensor filters from the live page, and adds additional filters for date and time. The date and time filters have calendar date picker to allow the user to select a date or a range of dates. Next the user can filter the data from the selected date by a time range. The data and time filters provide the user much flexibility to which data they want to view. The user can select a snapshot of frequency strength by setting one date and one time, ie December 8, 2016 and setting both time drop downs to 8PM. Likewise, it is possible for the user to view how signal strength changes at a time over a

date range by selecting a date range and a specific time. An example of this scenario would be December 8, 2016 to December 18, 2016 and setting both time drop downs to 8PM which would then cycle through the data streams applying the filters appropriately. On the right side of the screen are the map and sensor tables which are identical in look and functionality to the live page. In addition to those features, the historical page has added is an Angular-charts histogram that will provide Dr. Chatterjee with extra data.

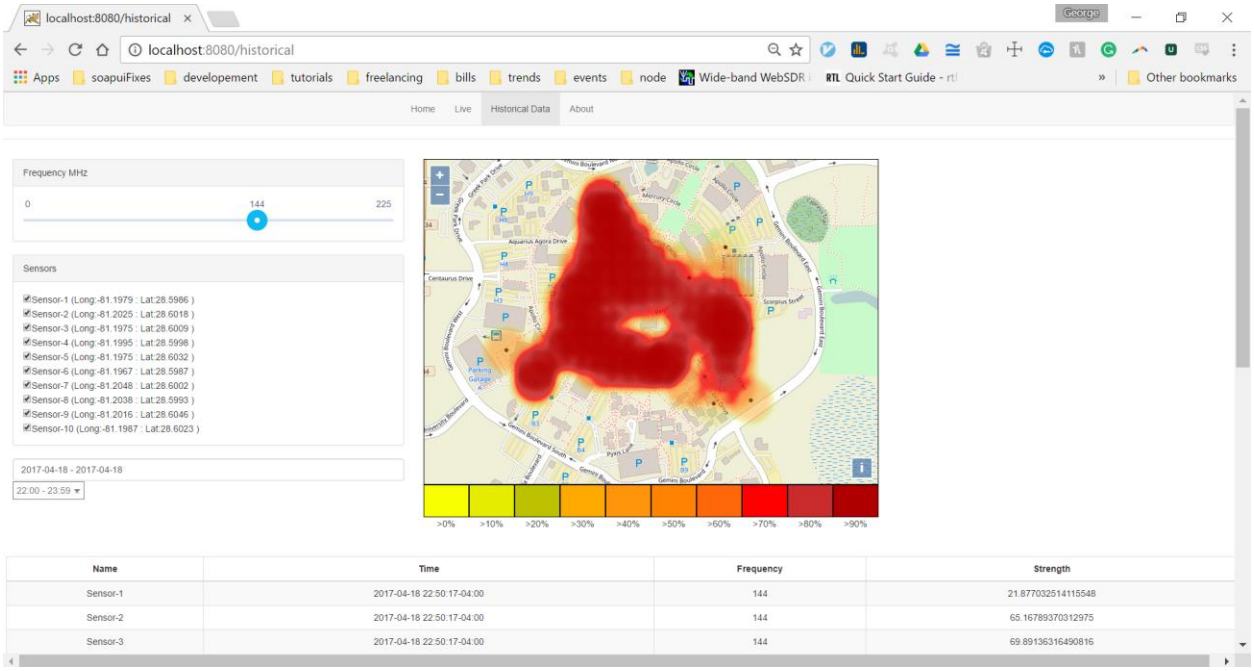


Figure 22: Historical Data Map Mockup

Front-End Dependencies

The dynamic frequency map application will require some dependencies to function properly. These dependencies will be transparent to the user and will be maintained by Node. The dependencies are bundled with the HTML files that are downloaded into the browser. The browser will usually cache these dependencies for quicker access. Using templates to build the user interface also make the browsers access to these dependencies easier since they are loaded when the user first visits our application and not every time the user changes page. Our Node server will utilize a tool called bower to download and update front end dependencies. Bower is installed globally on the Node server by running the command npm install -g bower. NPM is the Node package manager which is used to install tools that the application needs. Once bower is installed on the server, the dependencies can be downloaded and saved in the bower.json file. These dependencies only need to be installed once and will be available to the client's browser when the initial page is loaded. The bower.json file keeps track of the all dependencies installed and which versions are need. The bower.json makes it easy to set up alternative development environments so that some of the development can be done locally and then pushed to the UCF server without working about conflicts between versions. Furthermore, bower.json give our application the ability to update any dependency with easy. By setting the dependency to "latest" in bower.json, every time bower install command is ran bower will retrieve the latest version of that dependency and update our system. Our bower.json may look something like this:

```
{
  "name": "DynamicFrequencyMap",
  "description": "dynamic frequency map application",
  "main": "server.js",
  "license": "ISC",
  "homepage": "http://10.171.204.163:8080",
  "ignore": [
    "**/.*",
    "node_modules",
    "bower_components",
    "public/assets/libs/",
    "test",
    "tests"
  ],
  "dependencies": {
   "angular": "^1.5.8",
   "bootstrap": "^3.3.7",
   "angular-route": "^1.5.8",
    "angular-animate": "^1.5.8"
  "openlayers": https://github.com/openlayers/ol3/releases/download/v3.11.1/v3.11.1-dist.zip
  }
}
```

The Angular dependency has been covered extensively under the previous section. Bootstrap is an open source tool developed by twitter. Bootstrap provide a collection of premade HTML objects. Using bootstraps components. The development team will not have to worry about sizing and functionality. Bootstrap establishes a grid system on the screen which is then used to define the

height and width of their components. This is a nice feature because our team will not be bogged down with ensuring the user interface is aesthetically pleasing regardless of the screen resolution. In the future if Dr. Chatterjee decides to open the application on his cell phone or tablet, we are assured that the user interface will look nice due to using bootstrap components.

The other dependency that is worth noting is the OpenLayers 3 which is the map tool we will be using. Our team choose OpenLayers because it can be used with multiple map tile servers like Open Street Map, Bing Maps, or Google Maps. Setting up an OpenLayers map in the user interface is as easy as creating a new map object and setting its size parameters. Upon initialization, the URL to the map service is passed to the map object. We will be using Google maps service to serve the map tiles to our application. Once the map has been instantiated, layers are applied to the map to display features. We will use two layers in the dynamic frequency map. The first layer will be used to display the location of the sensors. These locations are static and will be retrieved through the API from the MySQL database. The second Layer will be the heat map layer which will be used to display the interpolated data. Once the front end application receives the data from our API, that data will be used to make the heat map features. These features are objects provided by OpenLayes and are then added to the heat map overlay. Additionally, OpenLayers provides a full library of interactions and events that we could use in our application. As it stands now, we will not be utilizing any of these OpenLayers features.

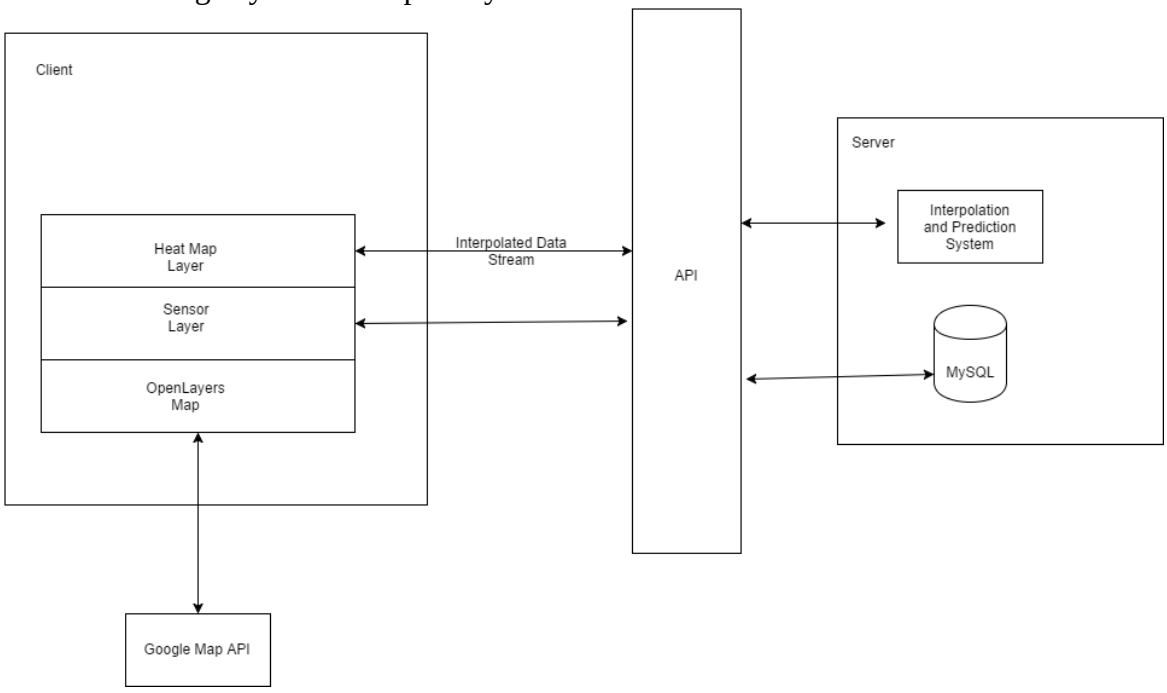


Figure 23: Basic OpenLayers structure

System Functionality Requirements

Requirement ID	Requirement Description
No. 1	System shall have a public web server.
No. 2	System shall have a data handling server.
No. 3	System shall have physical sensors that can scan radio frequencies.
No. 4	System shall have a web page.
No. 5	System shall have a Database.
No. 6	Web page will have a navigable map.
No. 7	Web page map shall display a visualized representation of data.
No. 8	Web page shall have a filter section.
No. 9	Filter allows user to change which frequency will be displayed.
No. 10	Map visualization updates as the web server receives the data.
No. 11	User shall be able to query the database for historical data via query API.
No. 12	Data handling server shall have a provisioning system.
No. 13	Data Handling server shall have a data processing system.
No. 14	Web server shall have Signal Interpolation Prediction System.
No. 15	Provisioning system shall detect provisioned broadcasting sensors and establish a connection to the sensor.
No. 16	Provisioning system shall establish a UDP stream with the sensor to take data from the sensor, time stamp it, and pass it to the data processing system.
No. 17	Provisioning system shall Keep a register of the location of the GPS coordinates of deployed sensors.
No. 18	Data processing system shall receive time stamped RF strength from the provisioning system.
No. 19	Data processing system shall consolidate the data from multiple different sensors into one central stream.
No. 20	Data processing system shall take routine snapshots of this central stream and send it to a database for storage.
No. 21	Data processing system shall This system should be able to handle multiple simultaneous messages.
No. 22	Signal Interpolation Prediction System shall receive the central stream and apply an interpolation algorithm on the received data to create an estimate of RF strength in areas without sensor coverage.

No. 23	Signal Interpolation Prediction System shall transmit the data to the web server for public viewing.
No. 24	Sensor shall cycle through each frequency record the signal strength of the of each frequency.
No. 25	Sensors shall convert the data received from the sensor into a UDP stream to the provisioning system.
No. 26	Sensor shall transmit its data via Ethernet.

Project Milestones

Predicted Date	Task	Completion Status
10/6/2016	• Project Design Proposal and Project Requirements Due	Completed
10/15/2016	• Meet with senior design lab coordinator to establish server space	Completed
11/1/2016	• Configure Necessary tools	Completed
11/15/2016	• Web server running • Database management system ready for use	Completed Completed
12/6/2016	• Finish project documentation	Completed
12/7/2016	• Final Project Documentation Due	Completed
12/15/2016	• Setup front end interface • Setup controls and event handlers • Begin programming sensors	Completed Completed Completed
1/1/2017	• Establish communication from front end to back end • Verify proper request and responses for front end events • Begin work on sensor simulation system • Have basic database in place	Completed Completed Completed Completed
1/15/2017	• Begin preparations for test plans • Setup unit tests • Finish sensor programming • Finish simulation system	Completed Completed Completed Completed
2/1/2017	• Launch sensors • Begin taking data • Begin work on signal strength projections	Completed Completed Completed
3/1/2017	• Finish signal strength estimation • Fix and close all know issues and bugs	Completed Completed
3/15/2017	• Wrap up development phase • Make sure that all system requirements have been met	Completed Completed

4/1/2017	• Final acceptance testing • Load Testing • Begin presentation preparations	Completed Completed Completed
TBD	• Final presentation	Completed

Budget

Category	Amount
Sensor Cost	< $30
Server Space Cost	$0
Licensing Cost	$0
Total Cost	< $30
Funding from Sponsor	$30
Student Total Cost	$0

The system requires a database, several sensors to monitor radio frequencies and transmit the results, a public web server, and a second server for handling data. The database and the servers we were told that we could use resources from the senior design lab to meet those requirements. Since this is a proof of concept we will be not be using any licensed hardware or software, therefore there is no funding will be needed for Licensing fees. The only expected cost of the system is the sensors which we told that our sponsor Dr. Chatterjee would be handling those cost. The expected cost of these sensors it to be no more than 25-30$ based on the components that make up the sensors.

Networking in C++

There are a huge number of libraries available for use with C++ for networking. Having to work with networking when programming is a common problem, so many solutions have been implemented over the years. As a result, there were too many libraries to do in-depth research on in any regular amount of time. To that end, I first had to establish some criteria to trim the list of libraries.

The first requirement that cut down on the number of libraries available to choose from was the ability for the library to be cross-platform. Although not a key functionality of our system, the ability to scale the system up would be restricted by being limited to a single operating system. By allowing users to be able to install the system across any operating system, we allow them to easily scale deployments within their own networks without requiring additional software installs.

The second requirements that the team utilized was restricting the libraries in use to C++ code only. A surprising number of networking libraries are still written in C and while you can compile C and C++ code in a single project, it can create some surprising complications down the line. In order to avoid these complications, the development team decided to drop any networking libraries that were still written in C.

The third and final requirement was to try and minimize the number of additional dependencies brought in by the libraries. Many networking libraries are not simply networking libraries but are also designed to complete other tasks. While this may be useful in certain cases, the core of the application will be networking and it doesn't need additional sources of breakage added to the project. Additionally, I wanted to avoid drawing in too many dependencies from the underlying library itself (eg, relying on Boost), so those got the cut.

Among the remaining libraries left, the team looked for a few basic standards. For one thing, the libraries needed to have some form of activity by releasing new version/patches at a semi-regular basis. There wouldn't be much of a point of developing an application on top of a library that is no longer being developed or maintained. The developers also looked to see which libraries have a healthy amount of documentation that looked like understandable. Developers on the team have worked with tools that have poor documentation in the past and there is no desire to repeat that experience. Finally, the team looked for tools that have been updated to be implemented in at least C++ 11. The standard has been available for five years now and primary developer for this section prefers to write code using features from the specification. While older C++ code can be used with C++11, C++11 brings with it some design principles and native support for threads that make a library written in it preferable.

After all of these requirements were examined, the final results were down to two candidates: C++-NetLib and Asio. An important factor between these two libraries was the relative activity of the projects. Of the two Asio seems to be more active, has a much larger community and seems to have better documentation. Additionally, it seems that the C++17 networking working draft seems to be based in part on the current Asio library (Wakely 2016). The fact that the C++ standards committee is looking towards the Asio library for inspiration is what ultimately lead towards the decision to use it in the project.

Unfortunately, the proved to be a decision that was made prematurely and in error. After several weeks of working with the Asio library (and its Boost variant), it became apparent that the documentation applied with the library was insufficient. It multiple cases, example code provided by the library was either directly contradictory of filled with several errors. Functions specified by the core system documents simply didn't exist in the actual library files. These problems using the library forced the developer to reconsider several different options in the middle of the development cycle.

After some additional review of library choices and the associated complexities that came with them, the decision was then made to not use an upper level library to handle the netcode and instead build an interface directly to the socket level API in the operating system. Work started with a socket level class being created to abstract some of the difficulties that come with cross platform socket level development, and the client system and rudimentary server were built in a socket level interface.

However, the sockets level interface that is shared by each operating system uses an outdated method of polling for incoming packets called select(). While an implementation of select() is supported by each operating system, the method itself has been deemed too slow for further new development to occur with it. Unfortunately, each operating system vendor decided to create their own implementation of a more advanced polling implementation. This operating system level fragmentation makes it exceedingly difficult to create a system compatible library that would work across multiple different environments.

As a result, the developer looked again for a library that would provide a level of abstraction just over the polling functions at the operating system level. The library that was selected was libevent, an event polling library that has been in development for many years and had a well-established source of documentation. Work began once again for a short time and some of the systems were converted to use libevent. However, it soon became apparent that while the library supported transmission in UDP, its support for other actions with UDP like buffers was non-existent. Additionally, since the library itself was written in C, there were multiple actions and features of C++ that couldn't be used within the event loop that was hobbling the system.

Due to the above, once more a review of possible options was done and eventually a final result that would result in the completion of the project was selected. libuv was billed as a successor to libevent, with better performance and more frequent support. However, libuv shared one of the same problems that the developer had with libevent, namely the library was written in C. In order to get around this, there was a wrapper library called uvw that acted as a C++ wrapper around libuv. uvw allowed the usage of modern C++ techniques in developing the netcode in the application.

The structure of the server netcode is similar across both servers. All of the server code exists inside a construct called an event loop. An event loop is essentially an infinitely running loop that acts at the main program loop. Different segments of code are registered to the event loop as handle, each with a different triggering condition. The core structure of the system is based around

two events: when a UDP packets is received by the system and when a fixed interval of time elapses.

Risks

As with any project, there are some risks associated with this system. Primarily, they revolve around the hardware that is to be used within the system but there is the potential for software problems as well. The hardware is a concern for the development team due to continual changes in what is available for the team. The software concerns relate to developer experience with the technologies in use. The risks are denoted below:

Risk: The hardware requirements for the project are in flux
Severity: High
Description: The hardware requirements are a risk because the requirements for what is acceptable hardware has changed multiple times throughout the project. The initial design proposal strongly implied that the physical hardware for the sensors was already complete and the development team would just have to program the sensors. Upon meeting with the client, this has proven to be incorrect. Since then, multiple meetings with the client have resulted in changing requirements. This is a problem because different hardware will have different capabilities, both software wise and hardware wise.
Mitigation: There are two mitigating steps that are being taken when dealing with the changing hardware. The rest of the system is being designed in a hardware agnostic mindset. As long as the hardware can transmit data in our expected format, the rest of the system will be able to interpret it without issue. The other mitigating step is to create a simulated sensor that is capable of transmitting data in lieu of sensors. This will allow testing and development of the other aspects of the architecture that are not directly reliant on the hardware.

Risk: Developer unfamiliarity with certain technologies selected for use with the project
Severity: Low
Description: There are several aspects of this project that no member on the team has any experience with. For each developer on the team, there is some aspect of the system that they have never had any exposure to. As a result, picking up these new concepts and technologies will take time.
Mitigation: Already during this semester, the development team has started to explore these unfamiliar concepts and technologies. As a part of this exploration, the technologies that were chosen for this system were chosen in part because of their ease of use or comprehensiveness of their documentation. These factors designated early on in the research process will reduce the amount of ramp time learning the new technologies before development will take place. The less time it takes before development can begin in earnest, the lower chance of this risk becoming an issue and it instead being mitigated.

Risk: Inability to handle the processing of the data streams at an acceptable speed
Severity: Moderate
Description: In order for the system to be considered acceptable by the client, the data needs to be transferred from the sensors to the visualization system that the user interacts with in near real time. The challenge associated with this is that the sensors constantly broadcast data and at each step of the system some calculation or process must be done on this data. The calculations can be

time consuming and the data set to perform it on grows rapidly as more sensors are added to the system.

Mitigation: In order to avoid taking too long at each of these steps, we will have to leverage a couple tactics. The first of which is parallelism. Parallelism is the process of taking one task and splitting it across multiple different threads. By doing this, we will be able to split the work done on each stream across multiple different processes to reduce the amount of time work takes processing a single instance of data. The other tactic that we are using is the use of user datagram protocol (UDP) streams when transmitting the data through the system. If a packet is dropped on transmission, that lost data is simply flagged and ignored due to how quickly new data is generated for the system. By using UDP instead of transmission control protocol (TCP), we avoid having to resend data when a packet is lost. The above action greatly slows down a system, so avoiding it should help improve throughput.

Development Methodology

Our group will be implementing a hybrid waterfall/agile model. Initially, we intended to adopt a strictly agile model but the emphasis of documentation in the timeline forced us to reconsider our approach. Due to the nature of the deliverable structure in regards to documentation, requirements gathering will occur in bulk at the beginning of the project. However, feature development will still occur in an agile manner. As a result of this, developers will have a clearer vision of the overall structure of the system and its integrations whilst still being able to iterate on features that have been finished. This allows for greater development room then what is provided by the waterfall method, as requirements change throughout the process as developers realize there is a better way of coding a system or constructing a user interface.

Work will be assigned and completed in sprints. A sprint is a fixed time increment intended to produce a useable and potentially releasable product in the increment that it is created (Schwaber & Sutherland, 2013). Although it is not always an achievable goal, it is intended to be used as a clear progress metric that can be followed throughout the semester that is still flexible enough to allow developers to focus on other tasks. Our targeted sprint length is currently two weeks. The below figure is an example of a typical sprint lifecycle (see fig).

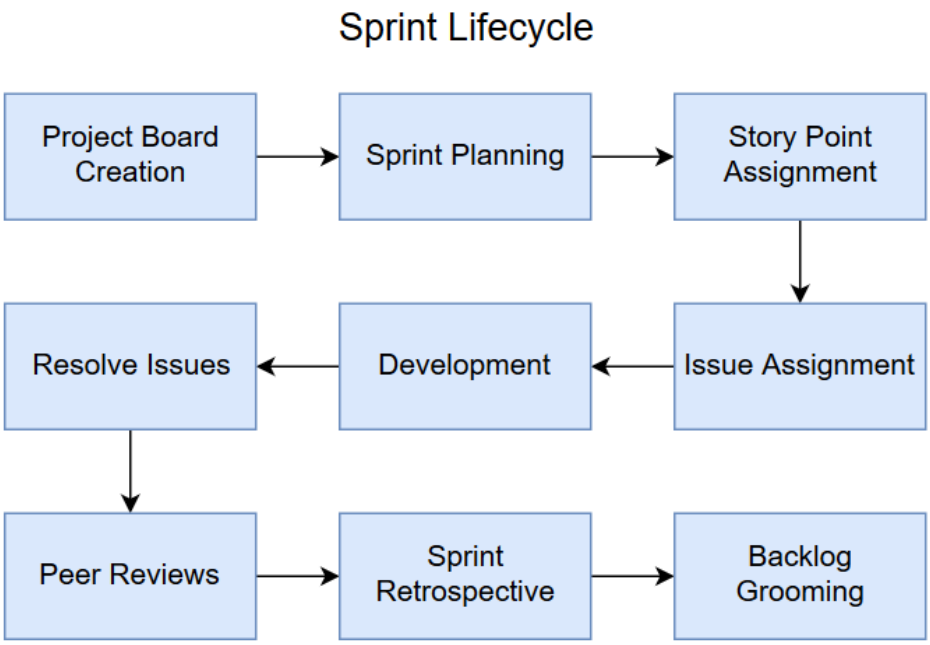


Figure 24: Sprint Lifecycle Diagram

Another important tool that we will be using as part of this process is Github Projects. A Github Project is a board that can be used to track the progress of an issue. A typical board will be split into four sections, each denoting a different stage. The first section is the To-Do section to denote items that still need to be completed. An in progress section to let other developers know that an issue is being worked. A blocked section that a developer can move an issue into to let other developers know that they can't proceed with a task until another one has been completed. The

last section is a completed section, so developers can tell as a glance what has been finished. Each sprint will have its own project board, with each board being assigned its own set of issues.

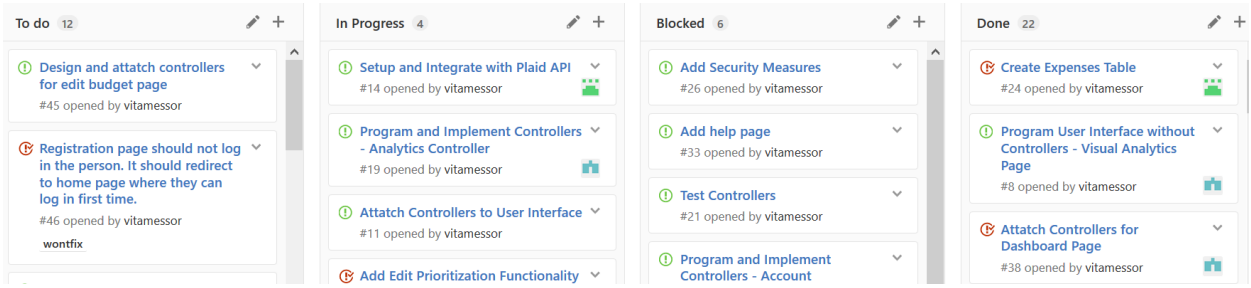


Figure 25: Github Issues Example

With the addition of sprints includes the need to estimate how long tasks will take to be placed in a sprint. Sprint planning is crucial part of this process. Sprint planning is a meeting where all of the developers on the team get together at the start of the new sprint and estimate how long a task will take. Each task is assigned a certain number of story points. Story points are a rough estimation of how long a task will take and in our case, one story point will equal one day's worth of work. During this meeting, the team will also determine the Sprint Velocity, or how many story points are available for that sprint. Using the number of story points available for the week and the estimated amount of story points for the tasks, developers can get a solid idea of how much work can be done in the sprint. Utilizing this method, developers can also begin to get an idea of if they will meet their overall deadlines based upon the time estimates they've given for tasks in the project.

Each task will be assigned an issue in Github Issues. Github Issues is a system for tracking defects and feature development in the project. Issues can be assigned various tags to denote the status of the issue, the priority of the issue, or whether the issue is a defect. Within the system, developers can be assigned to issues to denote that they are planning to work them. The system tracks the amount of time that an issue has been open as well as the amount of time a developer has been working the system. When a developer finishes a task, the issue will be closed by the developer and the rest of the team will be notified that the issue has been completed. A workflow of an issues progress can be seen below (see figure).

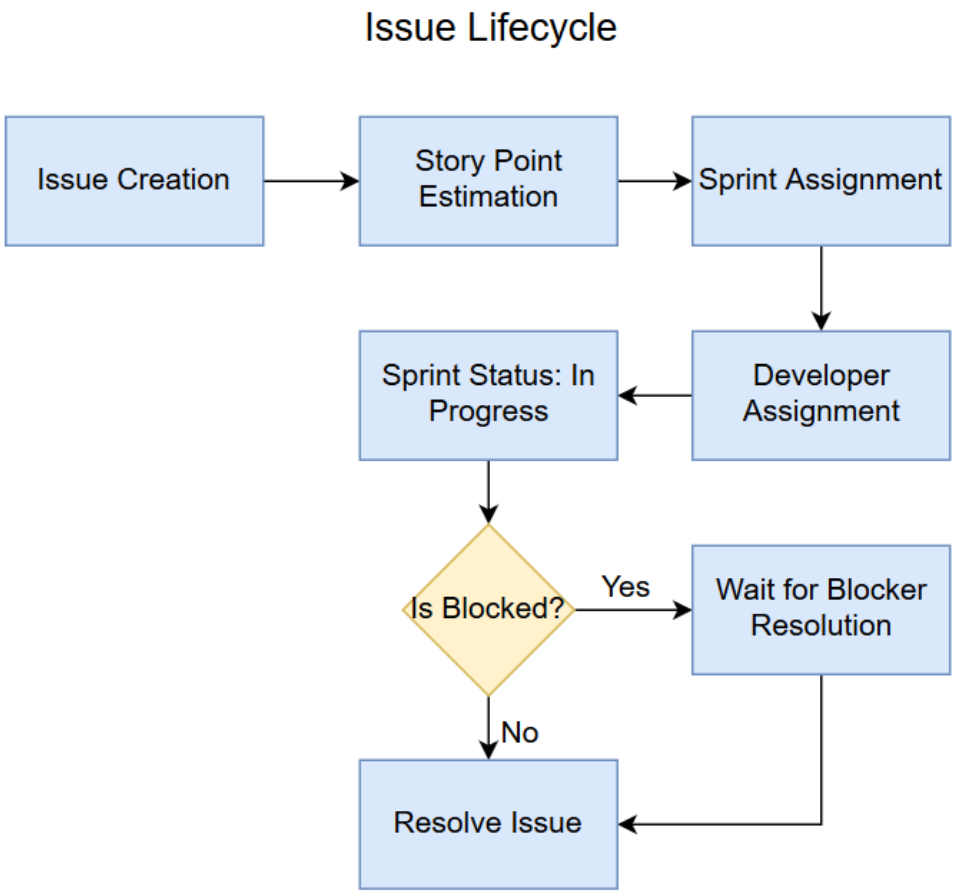


Figure 26: Issue Lifecycle

Once a developer has completed an issue, it will be submitted for peer review via a pull request. Peer review is an important stage in the programming process where other developers on the team examine the code a developer has submitted for any poor design choices or errors. Peer review is intended to be a constructive process, allowing developers with more experience with a particular technology to help a developer that may be less familiar with a tool set. Any bugs that are caught early in development are much easier to fix than later down the line. Once the changes have been reviewed by the team, they are merged with the code base.

At the end of the sprint, the development team has a meeting called a sprint retrospective. A sprint retrospective is a meeting where the development team gets together and has a brief discussion about what happened over the course of the sprint. In this meeting, the team discusses what went well throughout the sprint, any problems that arose during the sprint, and where processes can be improved. The meeting is primarily intended to address any problems early as the arise but is also used to track how the team feels about the progress of the project.

The last step in the sprint life cycle is backlog grooming. Backlog grooming is a process by which you review and tasks that have been place in your backlog (due to not being finished during a sprint or flagged as non-critical) and determine what order they needed to be completed in. You

then allocate a budget of story points in your upcoming sprint to be utilized for completing items in the backlog. This step is important to avoid too many items being pushed into the backlog and milestones being shifted.

The hybrid waterfall/agile methodology best suits the nature of this project. The use of waterfall to prepare our documentation gives the development team a solid framework to be filled in during development. By properly utilizing agile when developing the application, the team will be able to break the system into smaller chunks and iteratively develop pieces in such a way to as to avoid blocking someone that needs data further downstream. The emphasis on having something releasable at the end of every sprint will keep the team on task and continually producing code.

Continuous Integration

An important part of our development methodology is a strategy called continuous integration (CI). Continuous integration is a process by which whenever a developer commits changes to certain branches, a suite of additional tools is run against that code. Most often, this is used for automated testing that is designed to prevent bugs from being introduced into the system (commonly called regression testing). Since the code is built every time a change is submitted, developers can see if something was checked in that would break the build. As a result, they can avoid pulling down the latest copy of the code until the developer that submit the previous change can apply a fix.

Although continuous integration is typically applied for regression testing via the form of unit tests or integration tests, it can also be applied to other types of tests such as load tests or performance tests since CI tools can run command line instructions. As a result, we may leverage these additional features of CI, especially when we begin to simulate the output of the sensors to get performance numbers for the rest of the system. Rather than kick of a load test manually, just run one every single time the project is built and save the resulting performance numbers to see if the changes made benefit the project or not.

Branching Strategy

In important part of the development process is managing the source code of the application. Since our source control system is Git, development for the project will be done on branches. These branches will be denoted as follows:

- master – The master branch is reserved for any code that is considered completed. A master branch is reserved for code that has been fully peer reviewed and ready for production. There should never be code checked into master that has the potential to break the build, it is reserved for release candidates and full released.
- dev – The dev branch is where primary feature integration will take place. As developer resolve issued, they will open pull-requests to the dev branch for fellow developers to review and make comments on. Once a review is complete, the code is merged onto dev and the next feature is prepared for integration. The dev branch would be a branch for nightly builds of the system and is where testing will occur for every addition.
- dev-feature – Each branch where feature development occurs shall be named using the written specification (eg. dev-interpolation). Each feature shall have its own branch to avoid constant conflicts during commits if two developers are working on the same system.

- dev-issue+number – Each branch dedicate to an issue will be named using the written specification with number being the defect number on the issue tracker (eg. dev-issue5). This is done to create a clear separation between defect tracking and feature development as well as the ability to merge the bug fix into multiple different branches if necessary.

An example branch visualization is show below (see figure).

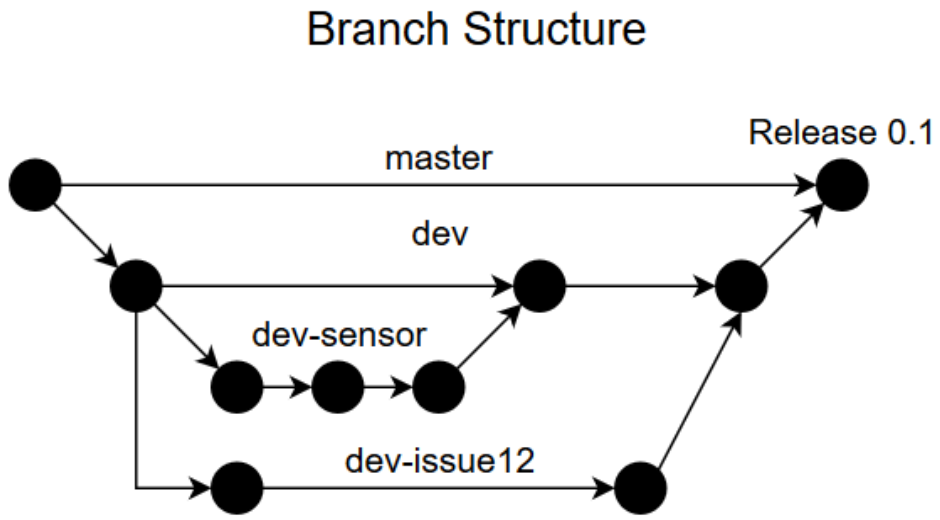


Figure 27: Example Branching Strategy

Sticking to the branch specifications laid out above makes it immediately clear what work is being done on a branch so developer do not have to spend extra time looking through branches for in progress code.

Infrastructure

Our server infrastructure for the class is hosted entirely on the senior design lab server. On the server, we have two VMs, each dedicated to different tasks. Each VM is running the headless version of Ubuntu 16.04 LTS. Administration of the two servers are being handled by Lucas Higginbotham and George Robbins.

The first VM will be dedicated to running the central processing server. This machine will have critical performance requirements due to the need to process and transmit the data in real time. The primary factor for this performance will be both the clock rate on the CPU assigned to the VM as well as the number of threads available to it. These factors are important since the bulk of the calculations should be occurring rapidly on the data as soon as it arrives. The threading is important because each data stream will be received by a new thread.

The second VM will be split amongst a few other tasks. It will host the database, the web front end, as well as any continuous integration (CI) systems we decide to use. In this case, CPU performance is not as critical for the application since it just has to receive the data streams. The more important performance aspect is data storage, read/write speed, and memory. Data storage and read/write speed are important for database performance and memory is important for both the web server and the CI server. If the server runs out of available memory, it will fall back to page swaps which cause drastic performance degradation.

Data Simulation

Due to the changing hardware requirements that the development team has faced during this semester, we have decided to create some software based simulations of the hardware sensors. These simulations are being created for two reasons. The development of the rest of the system cannot be continually blocked by changing hardware requirements. As we have a limited amount of time to develop the system, components of the system that depend on the data collection must have some data to work with. The other reason is that we have a limited budget for actual physical sensors on the project. As a result, in order to properly load test our system we must be able to deploy are large number of simulated sensors in order to find performance issues that wouldn't be readily apparent with the small number of physical sensors we have at our disposal.

When deciding exactly to simulate the data sent by the sensors, the developers have a few choices. Each of these choices have their advantages and disadvantages and the team may choose to deploy some combination of each of them. At this time, all of these methods are being considered for use.

Models

There are currently two models being considered for use in simulating the data.

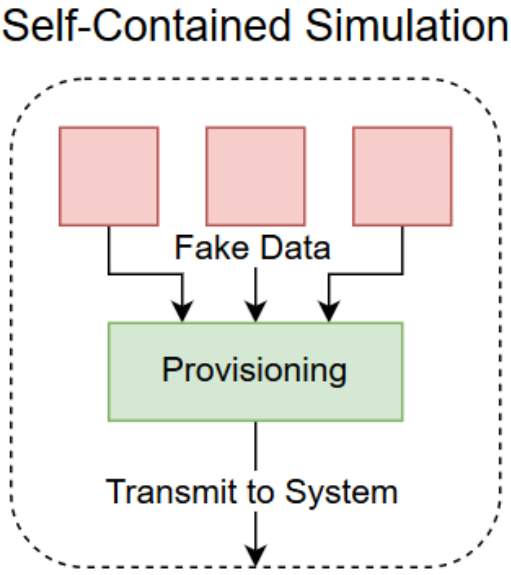


Figure 28: Self-Contained Simulation Model

The first model is that a simulation is written where the provisioning system simulates receiving data. This model is advantageous because it allows for easier deployment of the simulations. Since all the data is created within the self-contained simulation, deploying is as simple as spinning up a copy and putting it somewhere on the network. There is less of an associated hardware cost by spinning up only a single box to run the simulation. However, this version of a simulation is limited by the fact that it isn't a true simulation of the system. Since data isn't actually being transmitted across the internal network to the provisioning system, there are a number of factors that may be testable (such as packet loss, network latency, etc), but are difficult to simulate properly. So while this system may be easier to deploy, it doesn't truly provide a picture of the

end-to-end data transfer for the system. However, the cost reduction provided by using this method may be worth the lost accuracy.

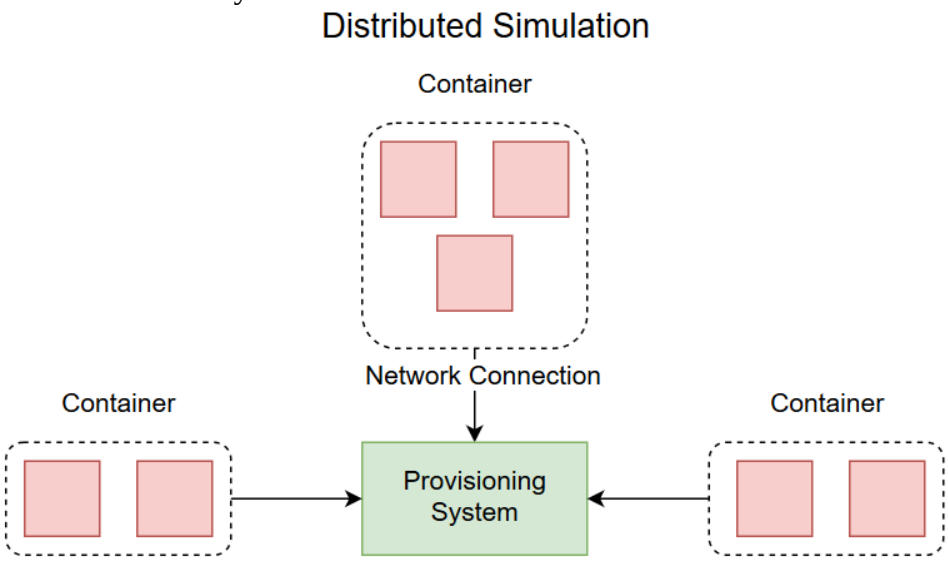


Figure 29: Distributed Simulation Model

The second model is that the sensors will be properly simulated on their own boxes. This model is advantageous because it gives the team a much better idea of how the system will perform in an actual deployment. This method allows for variations in network topography to affect the simulation, letting the development team discover flaws in the code base without necessarily needing to invest in additional hardware for each sensor. This method also allows the flexibility of deploying multiple sensor simulations on the same box, which avoids having to buy hardware for each sensor. However, the problems with this method are that additional hardware is still needed and management/maintenance of the additional simulations can become more complicated. As mentioned above, the simple fact of adding the network into the equation means that more hardware needs to be purchased for the actual sensor simulations. This, naturally, increases the overall cost of the simulation. The other issue is that each sensor will be in their own container on the testing hardware, which means that changing any parameters of the simulation could result in having to manually go to each simulated sensor and tweaking configuration there. So, this model presents the opposite scenario from the first, additional cost for additional accuracy.

Regardless of which of the two models that are picked, there will still have to be a method in place to virtualize the sensors. Towards that end, there are two options that are being examined. The first is to use virtual machines to host each simulation that will be holding either the provisioning system or the sensors. A virtual machine (VM) is a fully contained operating system hosted on another machine and managed by a hypervisor. A VM allows you to have multiple different systems running on the same hardware. The second option is called a container. A container is an instance of a library or application hosted within a single operating system install and managed by an engine. The containers can be run concurrently so you can have multiple distinct simulations running out of the same host operating system. A breakdown of a VM and container is seen below:

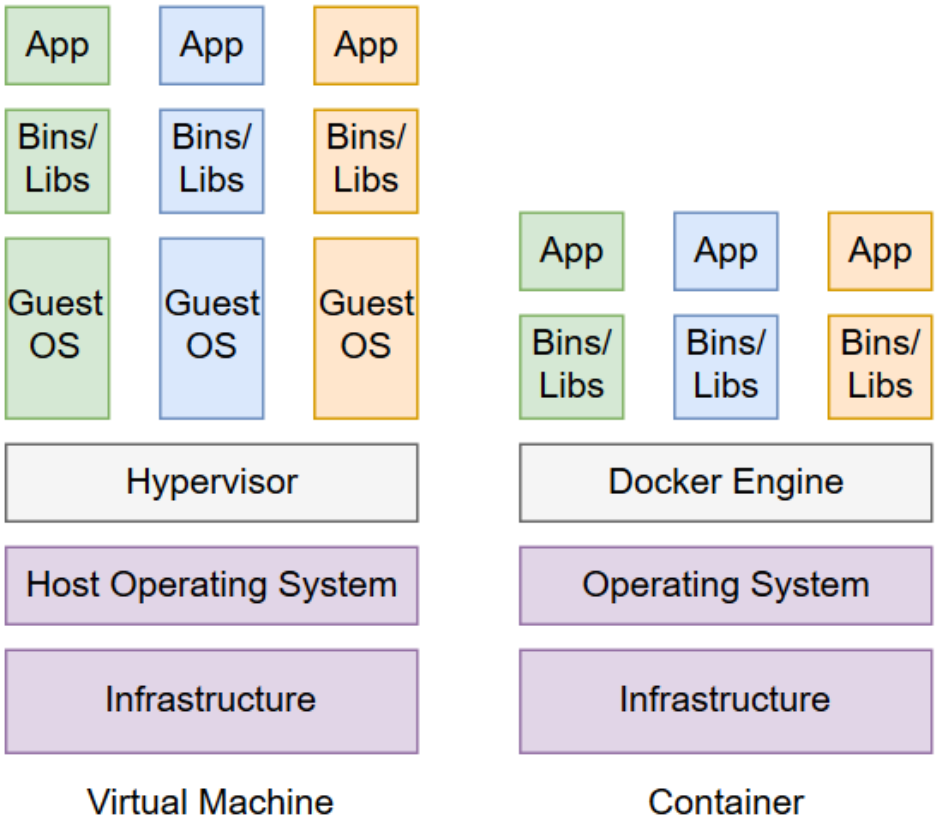


Figure 30: Container vs VM High Level Architecture

The advantages of going with VMs over containers is that the development team will be able to use the same hardware to test on multiple different operating systems. Since each VM has a full guest operating system installed on it, deploying multiple operating systems onto the same piece of hardware will be relatively simple. As a result, we can reduce the amount of hardware we need to purchase for simulating our sensors. However, having a full operating system installed on each instance creates some unavoidable overhead, causing there to be a smaller pool of resources available to each VM on the hardware. Another advantage is that members on the development have worked with VMs before. The fact that there is experience with VMs will allow the team to get the up and running quicker.

On the other hand, however, when working with containers all of the containers are run out of the same operating system. This can greatly reduce the overhead of running multiple containers on a single piece of hardware concurrently. Since the overhead is reduced, the hardware can potentially support additional instances of the simulation software without boosted hardware requirements. This is valuable to the development team because it allows them to generate additional load without paying for additional hardware. Another advantage of containers over VM's is that deploying a new instance of a container is sometimes as simple as running a new command. Since the containers just contain the core requirements necessary to run the application, quite a bit of the complexity is cut out of spinning up additional instance. This will have a great benefit when it comes to reducing development and testing time on certain aspects of the application. However, containers are still a relatively new technology and as such, the development team does not have

much experience creating and deploying them. It is currently an unknown as to how long the ramp time on using containers will be.

In all likelihood, containers will be the tool of choice for our development team. While the team is unfamiliar with the technology, the convenience of spinning up new simulations and the reduced hardware requirements for running those simulations have a significant value when faced with a limited budget. Also, at least for non-Linux testing, being able to spin up multiple instances on a single box is cheaper since server licenses typically cost money. With that said, we are still running all of our primary architecture in VMs, so the developer experience with the technology is not necessarily lost.

Additionally, do to production difficulties, the data coming from the sensors is being simulated at a client level. In order to achieve this affect, the client is structured in such a way to generate a new set of frequencies on a fixed time interval. During each interval, JSON message packets are constructed that contain multiple frequencies. The frequencies levels on the client level are generated at random from a fixed ranged from 0 to 100. Once all the frequencies are generated, the packet is marked with a latitude, longitude, size of message, and a timestamp for further processing by systems upstream.

Receiving Data

Each sensor in the application will be constantly transmitting data to a provisioning server. If the data was being received on only a single thread, it would quickly become choked trying to receive all the information. Each sensor will instead have its own thread associated with it and new sensors being added will spawn a new thread. Each thread will collect enough information from its individual sensor to create a full RF spectrum and then pass that information onto the main thread for processing. When the main thread receives data from all of the sensors in the application, it will forward it for additional processing.

High Level Database Overview

The database only has three main components to it. The sensors and their location, the data that the sensors send out to the live table to be read from before being dropped for new data, and the data that will be stored on the database for an extended period of time.

There are three properties that sensors need to have, identification, location, and whether or not that they are currently in use. Identification could just be represented by an integer. The location will be determined by latitude and longitude, both of which would be used as a type double variable. Finally, the currently in use section will be represented by a Boolean by the name of Active. Since the sensor id has to be unique it will be the primary key of the sensor table.

The recorded data needs to be able to record the findings of the sensor, to identify when the data was taken, what frequency the sensor was reading, which sensor the reading came from, and whether the data recorded was completed. Since some data may be read at the same time we cannot use the time as the primary key. Therefore, we added an index (which would be an auto incrementing INT) to keep each record separate from each other. The sensor ID would need to be the same as the one in the sensor table. Time would be recorded as DATE TIME for the fact that this system could be in use more very long periods of time and users may wish to view records of a given date. Frequency will readings will be a double to be as accurate as possible. Finally, there may have been an error on the sensor's side so there will be a non-null Boolean called Completed that will be used to determine if the record was processed possibly.

The live table is almost an identical to the table for recorded data. We have this table since we removed the direct transfer of data from the handling server to the web server. The way it will be used is the server will drop the table and recreate it and use a batch insert to place all the new data at once. This should be experientially faster than altering each individual record in the table.

The data recorded was kept separate from the sensors because it is assumed that they would be stationary. With this the locations will be the same for each record and would be pointless as well as a waste of space to list the location information each time. The sensor id will need to be the same in both tables in order for the foreign key to associate the records in the recorded data with which location the sensor was at. For future reference if the physical sensor need to be moved it will need to receive a new record in the sensor table as well as a new sensor id.

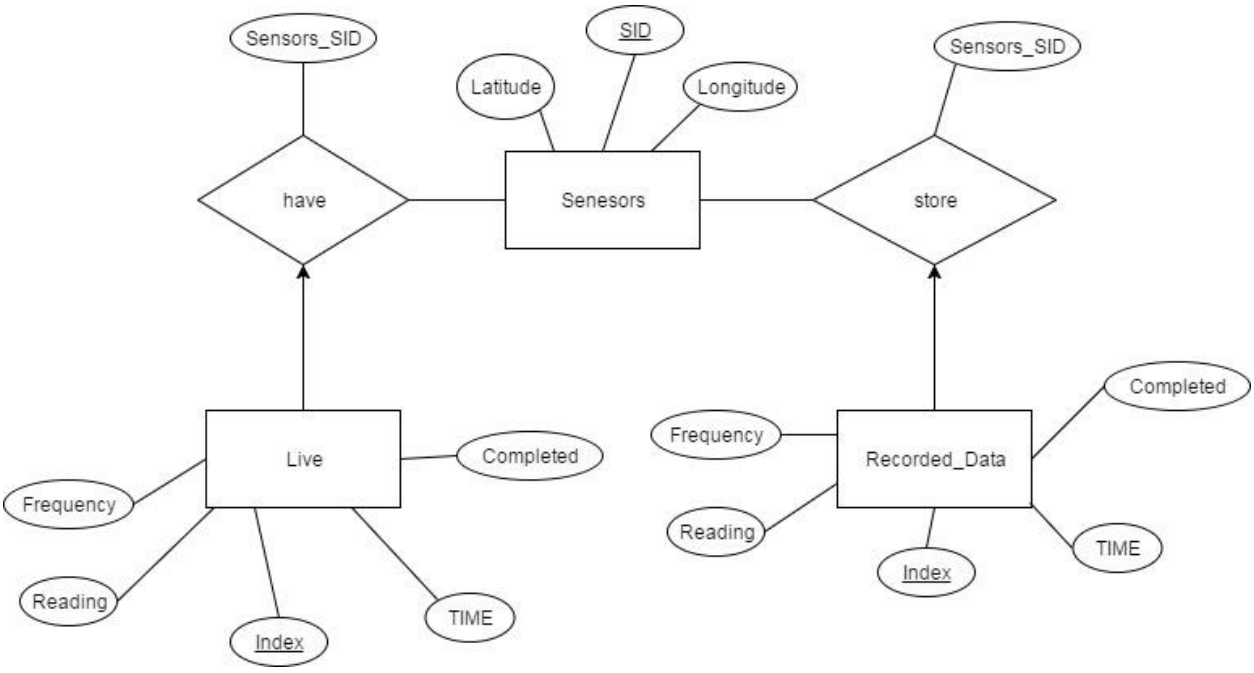


Figure 31: Database ER Diagram

Database SQL scripts

```
-- MySQL Script generated by MySQL Workbench
-- Wed Apr 19 13:34:50 2017
-- Model: New Model    Version: 1.0
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';

-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `mydb` DEFAULT CHARACTER SET utf8 ;
USE `mydb` ;

-- -----------------------------------------------------
-- Table `mydb`.`Sensors`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Sensors` (
```

```
  `SID` INT NOT NULL,
  `Latitude` DOUBLE NULL DEFAULT NULL,
  `Longitude` DOUBLE NULL DEFAULT NULL,
  PRIMARY KEY (`SID`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Recorded_Data`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Recorded_Data` (
  `Index` INT NOT NULL AUTO_INCREMENT,
  `TIME` DATETIME NULL DEFAULT NULL,
  `Completed` INT NOT NULL,
  `Frequency` DOUBLE NULL DEFAULT NULL,
  `Readings` DOUBLE NULL DEFAULT NULL,
  `Sensors_SID` INT NOT NULL,
  PRIMARY KEY (`Index`, `Sensors_SID`),
  UNIQUE INDEX `Index_UNIQUE` (`Index` ASC),
  INDEX `fk_Recorded_Data_Sensors_idx` (`Sensors_SID` ASC),
  CONSTRAINT `fk_Recorded_Data_Sensors`
    FOREIGN KEY (`Sensors_SID`)
    REFERENCES `mydb`.`Sensors` (`SID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Live`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Live` (
  `Index` INT NOT NULL AUTO_INCREMENT,
  `TIME` DATETIME NULL DEFAULT NULL,
  `Completed` INT NOT NULL,
  `Frequency` DOUBLE NULL DEFAULT NULL,
  `Readings` DOUBLE NULL DEFAULT NULL,
  `Sensors_SID` INT NOT NULL,
  PRIMARY KEY (`Index`, `Sensors_SID`),
  INDEX `fk_Live_Sensors1_idx` (`Sensors_SID` ASC),
  CONSTRAINT `fk_Live_Sensors1`
    FOREIGN KEY (`Sensors_SID`)
    REFERENCES `mydb`.`Sensors` (`SID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

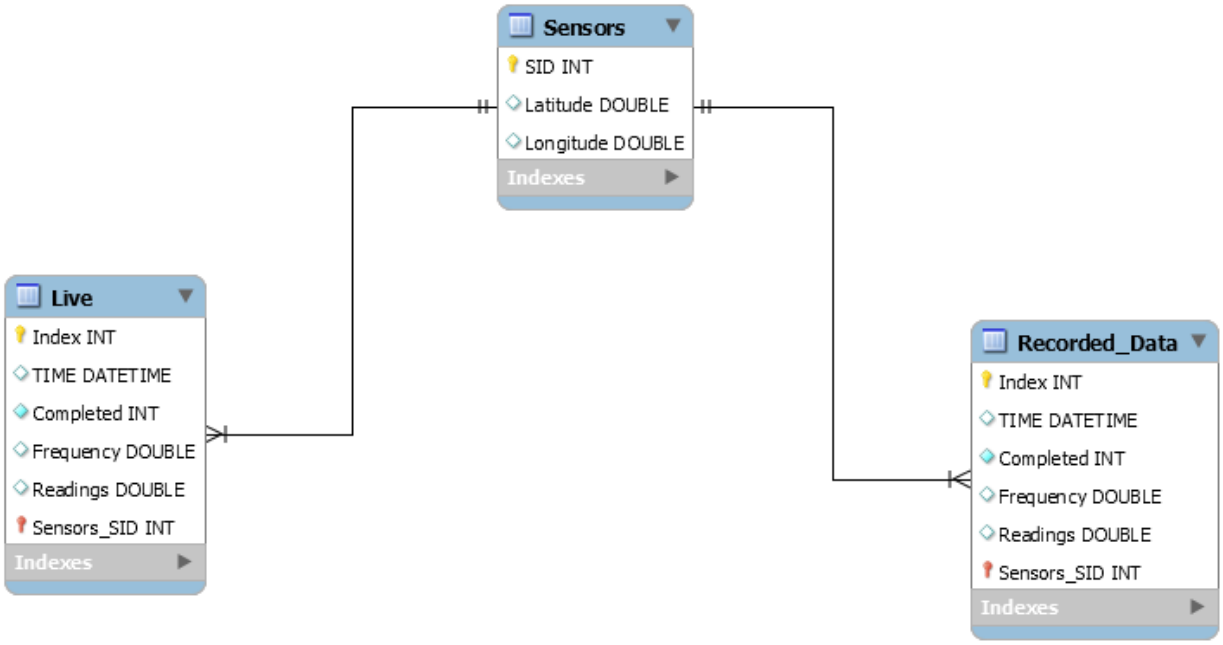


Figure 32: Database Table Structure

Testing

Unit testing

Before we can begin testing of the system as a whole we must insure that the individual components of said system are functioning properly. The components are the sensors, data handling server, public web server, and a database.

Sensors

Requirement ID	**No. 3**
Requirement Description	System shall have physical sensors that can scan radio frequencies.
Testing Method	Requirement No. 2 will be tested indirectly through other tests.
Passing Requirement	Testing of Requirements No. 24 and No. 26 both result in a pass.

Requirement ID	**No. 24**
Requirement Description	Sensor shall cycle through each frequency record the signal strength of the of each frequency.
Testing Method	Have a program controlling the sensors and instead of submitting it to the data handling server have the each of the frequencies and readings be displayed to the tester.
Passing Requirement	The program should display one record of each of the frequencies containing the frequency being read as well as the signal strength being recorded.

Requirement ID	**No. 26**
Requirement Description	Sensor shall transmit its data via Ethernet.
Testing Method	Check to see if there is an Ethernet cable properly
Passing Requirement	Ethernet cable is confirmed to connected to the sensor module by the tester.

Data Handling server

Requirement ID	**No. 2**
Requirement Description	System shall have a data handling server.
Testing Method	Requirement No. 2 will be tested indirectly through other tests.
Passing Requirement	Testing of Requirements No. 12 -14 all result in a pass.

Requirement ID	**No. 12**
Requirement Description	Data handling server shall have a provisioning system.
Testing Method	Requirement No. 12 will be tested indirectly through other tests.
Passing Requirement	Testing of Requirements No. 15 -17 all result in a pass.

Requirement ID	**No. 13**
Requirement Description	Data Handling server shall have a data processing system.
Testing Method	Requirement No. 13 will be tested indirectly through other tests.
Passing Requirement	Testing of Requirements No. 18 -21 all result in a pass.

Requirement ID	**No. 14**
Requirement Description	Data handling server shall have Signal Interpolation Prediction System.
Testing Method	Requirement No. 14 will be tested indirectly through other tests.
Passing Requirement	Testing of Requirements No. 22 -23 all result in a pass.

Requirement ID	**No. 17**
Requirement Description	Provisioning system shall keep a register of the location of the GPS coordinates of deployed sensors.

Testing Method	Check to make sure that provisioning system has a register of each GPS location of each sensor.
Passing Requirement	Each sensor has the proper GPS location associated to it.

Requirement ID	**No. 18**
Requirement Description	Data processing system shall receive time stamped RF strength from the provisioning system.
Testing Method	Have the provisioning system send a test time stamped RF strength.
Passing Requirement	Data processing receives the time stamped RF strength.

Requirement ID	**No. 19**
Requirement Description	Data processing system shall consolidate the data from multiple different sensors into one central stream.
Testing Method	Give the data processing system multiple different records from different sensors.
Passing Requirement	Data processing system successfully combines the into one stream.

Requirement ID	**No. 20**
Requirement Description	Data processing system shall take routine snapshots of this central stream and send it to a database for storage.
Testing Method	Give the data processing system records over a certain amount of time.
Passing Requirement	Data processing makes a copy of a record and is ready to be sent to the database.

Requirement ID	**No. 21**
Requirement Description	Data processing system shall This system should be able to handle multiple simultaneous messages.
Testing Method	Give the data processing system multiple messages at the same time.
Passing	Handles the messages properly.

Requirement	

Requirement ID	**No. 22**
Requirement Description	Signal Interpolation Prediction System shall receive the central stream and apply an interpolation algorithm on the received data to create an estimate of RF strength in areas without sensor coverage.
Testing Method	Send SIPS a test stream.
Passing Requirement	Algorithm sends out the correct estimation.

Public Web Server

Requirement ID	**No. 1**
Requirement Description	System shall have a public web server.
Testing Method	Requirement No. 1 will be tested indirectly through other tests.
Passing Requirement	Tests for Requirements No. 4, and No.6 – 10 all pass.

Requirement ID	**No. 4**
Requirement Description	System shall have a web page.
Testing Method	Attempt to connect to web page via local server.
Passing Requirement	Able to connect to web page and view its contents through web browser.

Requirement ID	**No. 6**
Requirement Description	Web page will have a navigable map.
Testing Method	If the webpage has a viewable map attempt to move the focus around to see another section of the map.
Passing Requirement	User is able to move the focus of the map.
Requirement ID	No. 7

Requirement Description	Web page map shall display a visualized representation of data.
Testing Method	Simulate data that should be given to the web server by the handling server.
Passing Requirement	The proper corresponding heat map is generated over the map.

Requirement ID	**No. 8**
Requirement Description	Web page shall have a filter section.
Testing Method	Check if the web page has a filter.
Passing Requirement	The filter is viewable from web page.

Requirement ID	**No. 9**
Requirement Description	Filter allows user to change which frequency will be displayed.
Testing Method	Change the setting in the filter.
Passing Requirement	A new heat map is generated over the map and corresponds to the data.

Requirement ID	**No. 10**
Requirement Description	Map visualization updates as the web server receives the data.
Testing Method	Simulate a timed set of data and send it to the web server over specific time intervals.
Passing Requirement	The heat map should change at almost the exact same time interval.

Database

Requirement ID	**No. 11**
Requirement Description	System shall have a Database.
Testing Method	Check to make sure the database that was described in earlier sections is integrated to the public web server by using MySQL command to view the tables in the database.

Passing Requirement	Tables are viewable.

Requirement ID	**No. 12**
Requirement Description	User shall be able to query the database for historical data via Public query API.
Testing Method	Store test records in the database and attempt to query the test records.
Passing Requirement	User is able to obtain test record.

System Testing

Assuming that the individual units are all working properly on their own we have to make sure that the units work together in the way that they should. The three main connections are between the sensors and the handling server, the handling server and the web server, and the handling server and the database.

Requirements 16 and 25 cover the connection between the sensor and the handling server.

Requirement ID	**No. 16**
Requirement Description	Provisioning system shall establish a UDP stream with the sensor to take data from the sensor, time stamp it, and pass it to the data processing system.
Testing Method	Sensor sends a UDP stream of data.
Passing Requirement	The data processing receives the corresponding time stamped data.

Requirement ID	**No. 25**
Requirement Description	Sensors shall convert the data received from the sensor into a UDP stream to the provisioning system.
Testing Method	Have sensors perform their scan and attempt to send a UDP stream to the provisioning system in the handling server.
Passing Requirement	Provisioning system receives the stream of data.

Requirement No. 20 tests the connection from the handling server to the database.

Requirement ID	**No. 20**
Requirement Description	Data processing system shall take routine snapshots of this central stream and send it to a database for storage.
Testing Method	Similar to testing this requirement in unit testing, the data processing system records over a certain amount of time. However, the snapshot this time will be sent to the database.
Passing Requirement	Database receives and stores snapshot.

Requirement No. 7 directly needs the connection between the handling server and the web server.

Requirement ID	**No. 7**
Requirement Description	Web page map shall display a visualized representation of data.
Testing Method	Have the Handling server send data to the public web server.
Passing Requirement	The map displays a representation of the data that was sent.

Build and Deployment

Client

Since the client is built using almost exclusive system level libraries, building the client can be accomplished in only a few steps.

Modern JSON

Modern JSON is a header only library that can be downloaded from this location: https://github.com/nlohmann/json/releases

Build

Run the below command in the client source folder.

g++ -std=c++11 -I /path/to/json frequency.cpp client.cpp -o client.o

To run the client, enter

./client <IP> <sensor-id>

With IP being the address you want to send data to and sensor id being an integer representing the id that the database will track.

Provisioning and Handling

The provisioning system and data processing system both operate from the same codebase, so both share the same instructions to build them. However, while the systems are both multiplatform, due to the differences between each platform the build instructions for the two are different.

Linux

The following is the dependencies and their build instructions that are required in order to build the project on Linux:

Modern JSON

Modern JSON is a header only library that can be downloaded from this location: https://github.com/nlohmann/json/releases

The development team used version 2.1.1 for application development. Later versions of the library may work, but use them at your own risk. Since Modern JSON is just a header only library. No additional actions are required to build the application with it beyond static linking it.

UVW

UVW is also a header only library that can be downloaded from this location:
https://github.com/skypjack/uvw

However, this library does not use any form of release schedule, so the development team created a fork of the library as it was when development of the application was taking place. This fork can be found at:
https://github.com/lhigginbotham/uvw

UVW is also a header only library, so no additional compiling is required. However, UVW is a wrapper library for libuv, so libuv must be installed as well.

Libuv

In order to install Libuv, you must first download the source for the latest release. That source can be found here:
https://github.com/libuv/libuv/releases/tag/v1.11.0
In order to build libuv on Linux, you must have the following tools installed:

1. build-essential
2. make
3. libtool
4. automake

On Ubuntu, the tools are all store in repositories can can be installed with sudo apt install <package-name>
Once the above tools are installed, you must run the following series of commands to build libuv:
sh autogen.sh
./configure
make
make check
make install
Once the following commands have completed, libuv will be built and ready for use in the system.

MySQL C++ Connector
MySQL is a common database used on a wide variety of platforms, so the installation process for most Linux Machines is as simple as making a call to their relevant package manager and downloading the library. On Ubuntu, the command is
apt install libmysqlcppconn-dev
After the installation is complete, the systems can be built.

Build
In order to build the system on Linux, you must use the following build command: g++ -std=c++14 -I /path/to/json/ -I /path/to/folder/uvw-master/src/ -I /usr/local/include/ -pthread config.cpp frame.cpp utilities.cpp server.cpp -o server.o /usr/local/lib/libuv.a -lmysqlcppconn

g++	The command to invoke the compiler
-std=c++14	Force the compiler to use the C++14 ISO standard. UVW reauires this standard to compile
-I /path/	Each of these are an include directory for c++ files. Since Modern JSON and UVW can be placed wherever on the machine the system is running, the include directory paths must be altered to compensate.
-pthread	Libuv uses pthread to handle some of its async operations, so the library must be called to build the system
-*.cpp	All of the source files used by the system
-o server.o	The name of the final executable output by the process
/usr/local/lib/libuv.a	A static link the the library file generated by building libuv in the earlier step, required to use libuv functionality
-lmysqlcppconn	Calling the MySQL Connector to be able to access the database

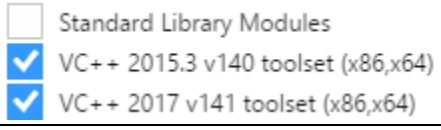
After the following command has run to completion (this can take some time), to start the server you can start the server with the command ./server.o.

Windows

Since Windows doesn't come preinstalled with a compiler, there are a few additional steps that must be taken in order to install the system.

Microsoft Visual Studio (Community-Enterprise)
In order to build the system and its required components, download and install Visual Studio (the Community edition is free to download) from here:
https://www.visualstudio.com/downloads/
In the installer, select Desktop development with C++.

WARNING: At the time of writing, the current version of Visual Studio is 2017. You CANNOT compile libuv version 1.11 on Windows without also installing Visual Studio 2015 build tools (see figure). Due to how the GYP build file is structured, it will not be able to build using just the Visual Studio 2017 build tools.



You can find that option by selecting *Individual Components* in the installer and then scrolling down to *Compilers, build tools, and runtimes*.

Python 2.7
In order to build libuv on Windows, Python 2.7 must be installed in order to run the GYP build scripts included in the project. You can find an installer for Python 2.7.13 here:
https://www.python.org/downloads/
After you've run the installer to completion, you must add the python executable as an environment variable. To do so, navigate to:
Control Panel > System Settings > Advanced System Settings > Advanced > Environment Variables
Once there, select New under user variables and enter PYTHON for variable name and the full filepath to the python executable for Variable value. After you complete this step, you should have something similar to the image below below (see fig).

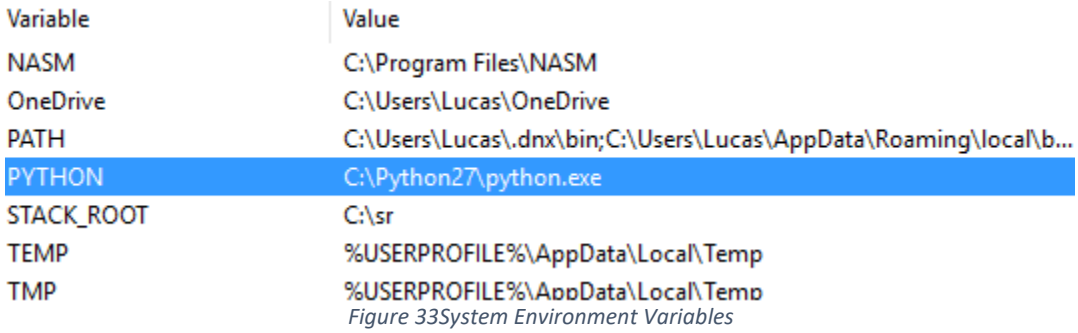


Figure 33System Environment Variables

Modern JSON
Download the appropriate header file and place it in the directory of your choice. Same as above.

UVW

Download the appropriate zip file and extract it in the directory of your choice. Same as above.
Libuv
On Windows, libuv uses the GYP build system to generate an appropriate project emplate based on your system. Once you have installed the previously listed requirements, simply navigate to the libuv directory in command line and run:
vcbuild.bat
After you've run that command, open the generated project file in the same directory called uv.sln in Visual Studio.
Build the project in Visual Studio.

WARNING: Visual Studio may try and default the project build type to x64. Do not attempt to build the project in x64, as it will fail. Ensure that you have x86 selected in the build menu as seen below (see fig).

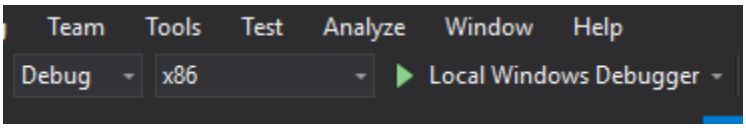


Figure 34 Build Options

After you run the build command you should see a file called libuv.lib in libuv-1.11.0\Debug\lib

MySQL CPP Connector
Download and run the installer for MySQL CPP Connector from here:
https://dev.mysql.com/downloads/connector/cpp/

WARNING: You must download the (x86, 32-bit) version of the connector otherwise the rest of the system will not compile.

Additionally, navigate to the ..\MySQL\MySQL Connector C++ 1.1.8\lib\opt directory and copy the mysqlcppconn.dll file from that directory into the libuvtest\libuvtest directory containing the server source code.

Build
Open the libuvtest.sln file located in the /libuvtest/libuv folder of the source code.
In the Solution Explorer, right-click on libuvtest and select Properties. Navigate to C/C++ and select the dropdown arrow on Additional Include Directories, and hit edit. Verify that all of the required include directory paths are correct for your machine setup. An example is shown below (see fig).

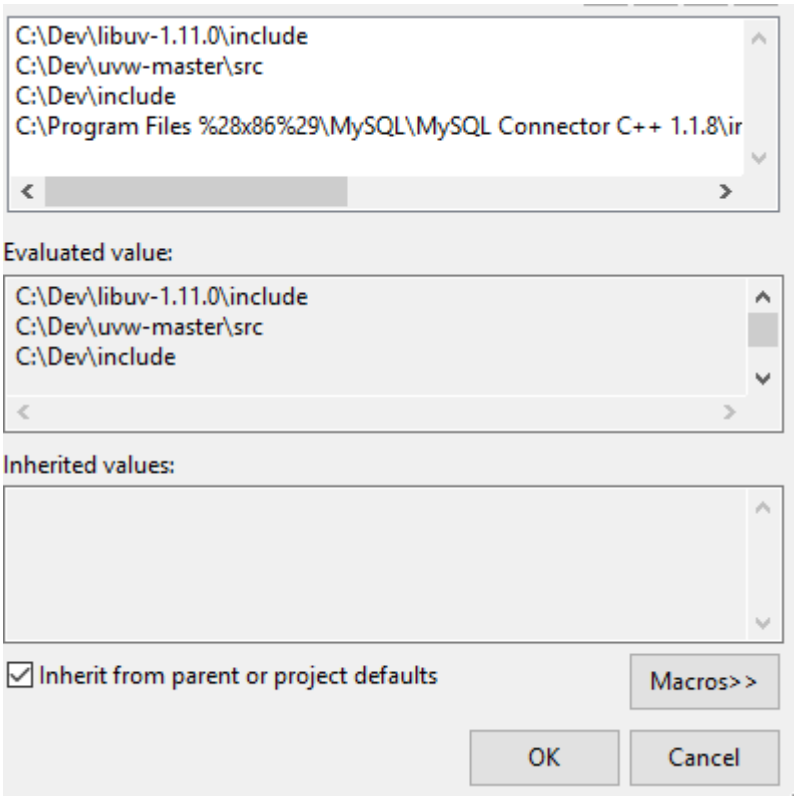


Figure 35 Additional Include Directories

Hit OK and then on the same properties page navigate to Linker, select the Additional Library Directories dropdown and select <Edit>.
Verify that the MySQL Connector lib directory is correctly included. See below for an example (see fig).

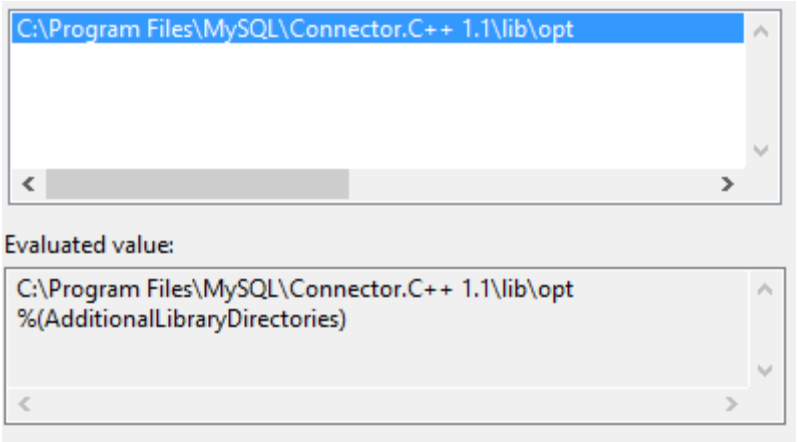


Figure 36 Additional Include Library Directories

Lastly, under the Linker selection navigate to Input, select the Additional Dependencies dropdown, and ensure that you have the following libraries included and that in the case the path matters, that the path matches the directory structure on your system. See an example below (see fig).

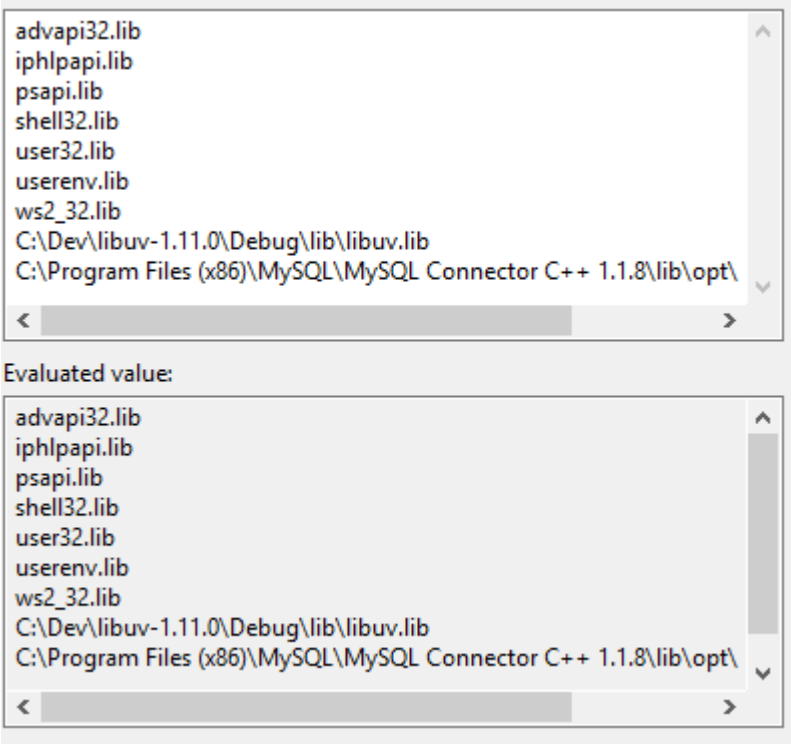


Figure 37 Additional Library Includes

Lastly, close all the settings boxes and build the solution with an x86 target.
The system will then output an executable called libuvtest.exe, which can be run to launch the server.

Configuration
The provisioning system and data processing system can both be configured by the same style of configuration system. The configuration file is formatted in JSON for ease of parsing and the options are listed below. The system looks for a file by the name of config.log in the same directory as the executable.

type	This option is used be the system to determine which mode it should be running in. When its 0, the system is operating as a provisioning system, when its 1 the system is operating as a data processing system.
sendip	This option is used to determine where processed packets are being sent. For the moment, this is opnly used when in provisioning mode
databaseIP	The IP address where database calls can be made
databasePort	The port the database is listening on
databaseUsername	The username used to access the database
databasePassword	The password used to access the database

Web Server

The web server consists of a few parts that require being built and configured. The Server running on Node environment and utilizes a C++ interpolation system. The node environment

depends on NodeJS being install locally on the machine. G++ compiler is used to build the interpolation system.

Node

First install NodeJS by downloading the installer from https://nodejs.org/en/download/. Navigate to the web server directory and run “npm install bower –g” in the command line. NPM is the node package manager that is used to install and manage dependencies for the server. This command will install bower which is another package manager specifically for front end client dependencies. Once the npm finishes installing bower, run “npm install” which will look up dependencies from the package.json file to install and configure. Now, run “bower install” which retrieves the list of front end dependencies and installs them. The library the AngularJS client uses to render graphs of historical data is managed by the npm and needs to be moved to the directory so that the client browser has access. Navigate to the node_modules directory in the web server. Copy Angular-Charts and Charts-JS directories to the <web server root>/public/assets/libs/ directory. Now all the webservers necessary dependacies have been installed and configured properly. To launch the web server, edit the config.properties file in the webserver root directory. Change the port: property to the port that the web server is to be listening on. Typically, web servers run on port 80 or 8080 but this could be any open ports on the machine. Edit the MySQLHost to point to the mysql server where the data is being stored. Use localhost if the web server and the mysql server are running on the same machine. Otherwise, this value should be the IP address of the machine hosting the mysql database server. User and password need to be set to the user name and password used to log into the mysql server. Database property is the name of the database that the webserver will query for data. After proper configuration, the web server is launched by running the command “node server.js” in the root directory of the server. The server will output “server started on port:<port>”, where <port> is the port set in the config.properties file, when it is finished initializing. To verify that the server is up and running, open a web browser and navigate to “localhost:<port>” which should display the project’s landing page.

Interpolation System

Have the c++ compiler installed and use the following command in the command line.
G++ -std=c++ -o interpolate source.cpp
The exe must be named interpolate in order it to be used by the web server.

Potential Future Additions

There are a few aspects we would like to have but we don't think we will be able to implement, whether it be due to time constraints or lack of resources. These aspects are mainly the concept of having mobile sensors, the use of a GPU to aid in the interpolation algorithm, and the design of the mobile sensors themselves and how they operate with the rest of the system.

Signal Interpolation

Since the goal of the sensors is to measure signal strength across a wide range of frequencies, calculating the signal interpolation map will be a complex task requiring thousands of calculations per cycle. In order to reduce the amount of time necessary to run the calculations, parallel programming will have to be heavily leveraged. For each frequency that is being captured, a task manager will spawn a new thread. Within that thread, a number of actions will occur.

The start of the process will remain on one thread, where the system will the sweep across the current GPS coordinates looking for "leftmost" sensor in the system (see fig). When it finds one, it will store it in the system and continue its sweep to the next sensor.

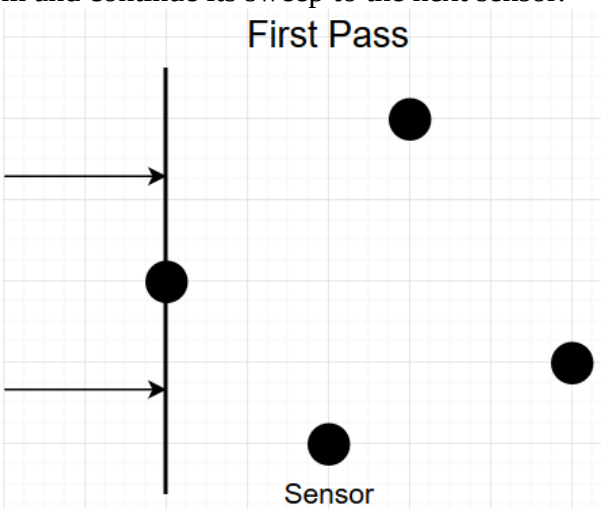


Figure 38: First Pass Sensor Sweep

The system will continue its sweep until it passes two additional sensors. Once the system has acquired three sensors, it will spin up additional threads to perform the actual calculations on the data. The signal interpolation calculations for each sensor will occur on a cardinal plane (see figure).

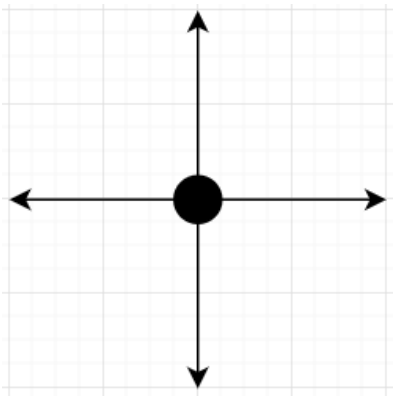


Figure 39: Cartesian Plane

Of the three sensors, a quick calculation will be run to determine their relative positions to one another (see fig). This is done by comparing the GPS coordinates of each sensor and then assigning relative positional coordinates to the sensors. The relative positioning calculated in this step will be used to determine thread assignment for the next step.

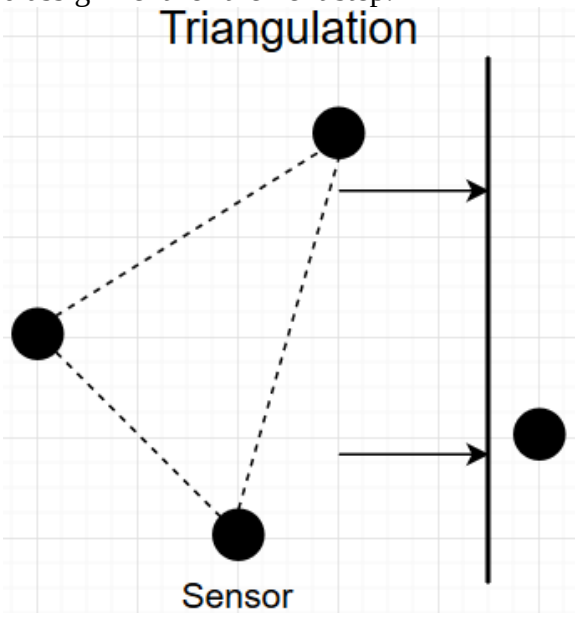


Figure 40: Sensor Triangulation

Once the positions of the sensors have been confirmed, the threads will be assigned as follows. One thread will be assigned to calculate signal interpolation to the left of the system based on the signal strengths from all three sensors. The direction from where the signal was coming from will be determined in this step (see figure).

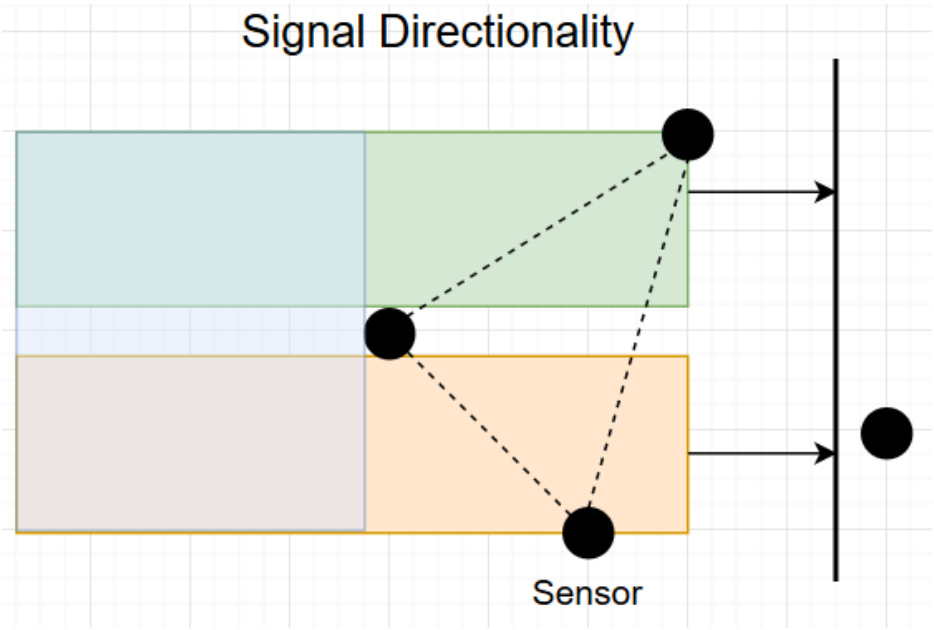


Figure 41: Sensor Directionality Calculation

Once the direction the signal was broad cast from has been determined, the signal interpolation begins on those threads. At that point, each of the four threads is assigned to a respective quadrant on the cardinal plane. Each frequency has a different distance that it is capable of traveling, with some of them easily traveling miles, while others are limited to a couple hundred feet. In order to limit the amount of time spent calculating to a reasonable rate, the interpolation points shall become less clustered as the distance between them increases (see fig). When an interpolating thread returns a distances that would no longer have a signal, the thread terminates and returns the interpolated values.

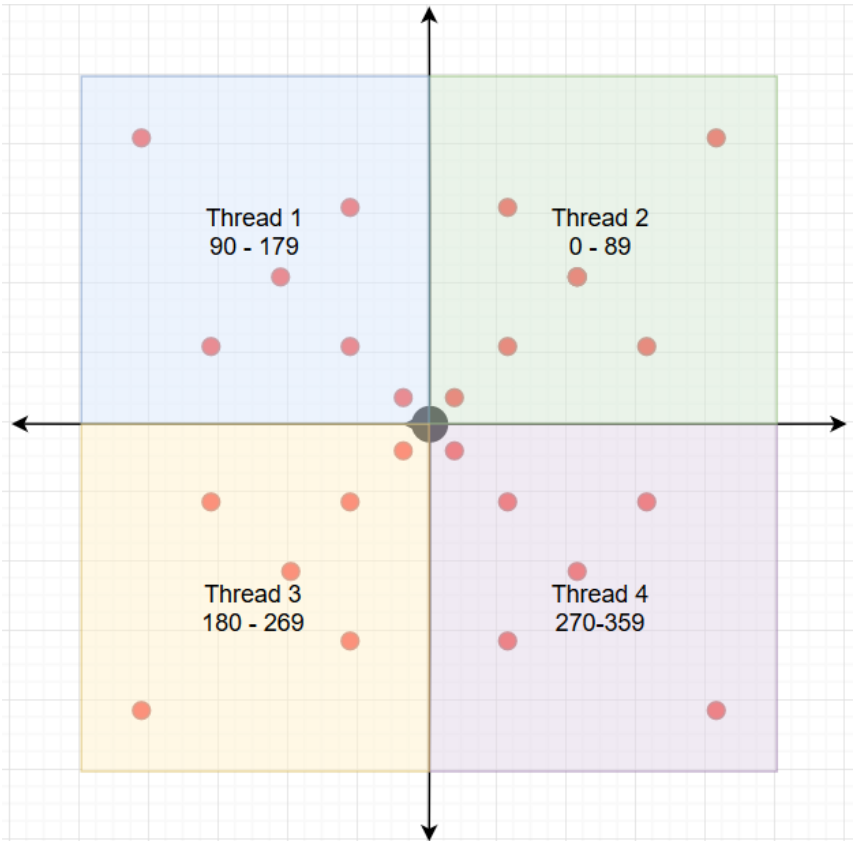


Figure 42: Sample Signal Interpolation

When all four of the threads have returned with their values, to the initial thread. The thread will handle merging the results into a single data structure, which will then be returned to the sweeping thread. The process on the sweeping thread continues, with each time it encounters a new sensor it spins up the triangulation phase again. Once an entire sweep is completed, all of the threads are freed and returned to the task manager at the top level process and the results are stored in a data structure. As the whole process above process has been running, the task manager has been allocating threads for each frequency that needs to be processed (see fig).

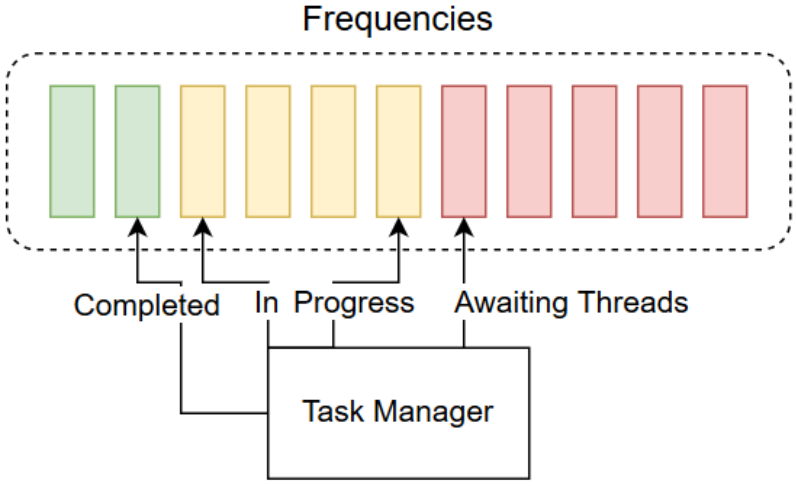


Figure 43: Frequency Management Overview

After all the frequencies have finished processing, the task manager forwards the collected information towards the next step in the system and begins the process anew.

In the event that the task manager reaches a lock state where all of the threads available to the system are in use, the task manager has a couple measures it can take. The first measure is to scale back the number of active frequencies it is trying to process. The task manager will review the number of in progress frequencies that are being processed by the system and cancel an outstanding operation. It will start with canceling an operation that is being blocked by lack of threads and waiting from more threads to be freed. If that proves insufficient, or if there are other systems on the machine that are cannibalizing threads, the task manager will slowly scale back active frequencies until there is only one active frequency being processed in parallel. In the event that there is still not enough threads available to the system, a final measure that may be implemented into the task manager is to fall back into a single operating mode, where all data is handled in only a single thread. If such an event were to occur, the performance impact would be critical and an administrator would be notified.

Mobile sensors

When the original idea was pitched by our sponsor, he expressed an interest in being able to have mobile sensors that could extend the range of coverage of the heat map. Having this would allow users to get a better picture of how the radio frequencies are acting in a given area. Given our resources and time constraints we decided to put the concept of mobile sensors into the “would like to have but probably won’t get” category.

Handling server

However, if we did decide that if a group (whether it be us or a future group) would want to pursue this addition there are a few issues that that group would need to take care of. First, the design of the database and the handling server would have to be altered slightly so sensors didn’t have a hard coded static GPS location. The handling server would have to do one of two things to compensate. Either the sensor would need a GPS unit installed and transmit its location to the handling server, or the server has to determine where the sensors are based on communication with stationary sensors.

Database

The current database is setup in such a way that it only has a single location associated with a sensor. The way we would recommend designing the database for mobile sensors would be to have four tables. First, Sensor table, which would contain the sensor ID, what type of sensor it is (mobile or stationary), and whether or not it is active. Second the recorded data that would remain the same only having the record index, sensor ID, time, frequency, and readings recorded. Third, stationary sensor table which would just have sensor ID and the longitude and latitude. Finally, the mobile sensors table which would contain the sensor ID, latitude, longitude, and the time it was at that location.

The following is a visual example of how the database would be setup and how the tables relate to one another.

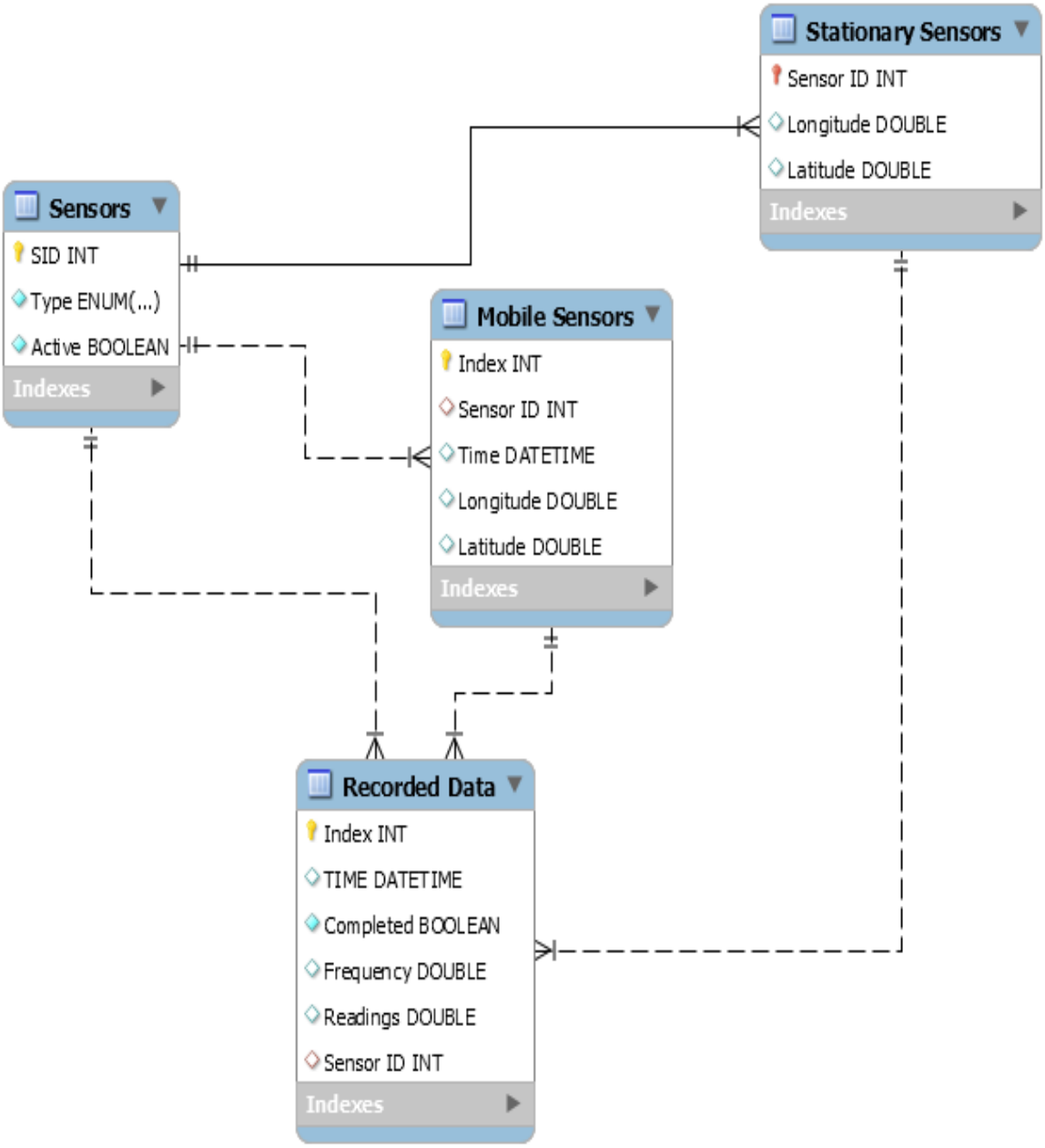


Figure 44: Updated database schema for mobile sensors

The following is an example of what the SQL scripts would look like.

```
-- MySQL Script generated by MySQL Workbench
-- 12/06/16 11:56:49
-- Model: New Model    Version: 1.0
-- MySQL Workbench Forward Engineering
```

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';

-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `mydb` DEFAULT CHARACTER SET utf8 ;
USE `mydb` ;

-- -----------------------------------------------------
-- Table `mydb`.`Sensors`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Sensors` (
  `SID` INT NOT NULL,
  `Type` ENUM('Stationary', 'Mobile') NOT NULL,
  `Active` TINYINT(1) NOT NULL,
  PRIMARY KEY (`SID`))
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `mydb`.`Mobile Sensors`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Mobile Sensors` (
```

```
  `Index` INT NOT NULL AUTO_INCREMENT,
  `Sensor ID` INT NULL,
  `Time` DATETIME NULL,
  `Longitude` DOUBLE NULL,
  `Latitude` DOUBLE NULL,
  PRIMARY KEY (`Index`),
  INDEX `sensor to mobile_idx` (`Sensor ID` ASC),
  CONSTRAINT `sensor to mobile`
    FOREIGN KEY (`Sensor ID`)
    REFERENCES `mydb`.`Sensors` (`SID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `mydb`.`Stationary Sensors`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Stationary Sensors` (
  `Sensor ID` INT NOT NULL,
  `Longitude` DOUBLE NULL,
  `Latitude` DOUBLE NULL,
  PRIMARY KEY (`Sensor ID`),
  CONSTRAINT `Stationaryto sensor`
    FOREIGN KEY (`Sensor ID`)
    REFERENCES `mydb`.`Sensors` (`SID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
```

```
-- Table `mydb`.`Recorded Data`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `mydb`.`Recorded Data` (
  `Index` INT NOT NULL AUTO_INCREMENT,
  `TIME` DATETIME NULL,
  `Completed` TINYINT(1) NOT NULL,
  `Frequency` DOUBLE NULL,
  `Readings` DOUBLE NULL,
  `Sensor ID` INT NULL,
  PRIMARY KEY (`Index`),
  UNIQUE INDEX `Index_UNIQUE` (`Index` ASC),
  INDEX `S_idx` (`Sensor ID` ASC),
  CONSTRAINT `S`
    FOREIGN KEY (`Sensor ID`)
    REFERENCES `mydb`.`Sensors` (`SID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `mobile`
    FOREIGN KEY (`Sensor ID`)
    REFERENCES `mydb`.`Mobile Sensors` (`Sensor ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `stationary`
    FOREIGN KEY (`Sensor ID`)
    REFERENCES `mydb`.`Stationary Sensors` (`Sensor ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

Sensors

By having the sensors be mobile we basically would have to scrap the original design. The sensors would have to have a different power source, the radio frequency scanner with the same capabilities as the stationary ones, be able to transmit its findings to the handling server, and figure out the location of the sensor.

By the sensor being mobile we cannot power the sensor using traditional power cords or by power over Ethernet. The powering of the sensor could only be done by setting up an on board power system whether it be by using a battery or by solar panel.

The sensors would either have to have to have a built in GPS unit to determine its own location or a way to communicate with the stationary sensors to determine how far away it is and triangulate its position.

The mobile sensors would have to be able to send the readings it found. So the sensor would need a transmitter and to send it to the handling server. This might need to be through a web of sensors to make its way back to the stationary ones if the sensors have no way of connecting to the internet.

Technologies Used

This section will list the tools and technologies used throughout the development lifecycle of our project. The below will include development tools, administrative tools, product frameworks, and design tools as well as a brief explanation of their usage in the project. Version listings are for informational purposes only and are subject to change during development.

Libuv

Version: 1.11.0
Libuv is a multiplatform support library with a focus on asynchronous input/output. It is most commonly known for being the back-end that powers Node.js, but it is also used in a variety of other software. In addition to asynchronous I/O, Libuv is also capable of running a fully featured event loop with support for operating specific polling functions, asynchronous file and file system operations as well as support for events, and fine grained signal handling. The library is intended to be a replacement for the older libevent and libev libraries.

UVW

UVW is a wrapper library for libuv written in C++. It supports all the major functionality provided by Libuv as well as providing an interface to use the features and functionality of modern C++14. The UVW wrapper allowed me to use library elements such as specific as std::unique_ptr<T> and as generalized as object oriented programming features. The ultimate goal of the library is to not expose any of the underlying C interfaces behind the C++ API, allowing developers to not have to write any C code.

JSON for Modern C++

Version: 2.1.1
JSON for Modern C++ is a JSON client library that's goal is to treat JSON as a first-class datatype in C++, similar to how JSON is treated in languages such as Python. The goal of the library is to provide intuitive syntax, trivial integration to existing codebases, and a low memory footprint. The

JSON object provided by the library is designed to behave like a standard template language object, even satisfying the ReversibleContainer requirement. The library was integral in developing the entire message system for the network packet transmissions, greatly easing the burden of developing a simple messaging format and allowing for a common format to pass data between the different developer components.

Connector C++/MySql

The C++ connector is Oracle's official connector for interfacing with MySQL servers in C++. It adds the ability to perform CRUD operations on the database, prepare statements, and perform most native SQL commands. The library is statically linked in Linux installations of the system and dynamically linked on Windows installations of the system. This particular connector does not support batch calls unfortunately, so that extra feature had to be written manually.

Git

Version: 2.10.2

Git is a distributed version control system and change management system. Unlike other common version control systems, Git operates on a distributed "branch" model, where developers can perform feature development without interfering with one another. This allows for an improvement in both developer's performance and flexibility. Additionally, the prolific use of Git has resulted in the creation of multiple tools and plugins that increases the ease of use and reduces the developer burden when using Git. Git is a development tools that will be used project wide for code base management.

Microsoft Visual Studio Enterprise

Version: 2016 Update 3

Visual Studio Enterprise is an IDE for developing a wide variety of web, desktop, and mobile applications across a multitude of languages. Visual Studio Enterprise was chosen for its maturity, stability, and wide feature set. In addition to development tools, Visual Studio Enterprise also ships with a suite of testing tools to aid in application development. The maturity of the platform has also allowed for a plethora of support for third party tools (such as Git), which helps to avoid breaking the developer workflow. The IDE will be involved in development in the majority of the application, with the exception of the database development and the sensor code.

MySQL

We chose to use MySQL for database management for a multitude of reasons. First of all, it is one of the most popular relational database management systems to date and thus MySQL has a vast amount of resources to aid us in the development. Also MySQL has a reputation of being one of the easier options to learn which helps us since Alexander Kinlen whom has very little experience with database management. The final reason for MySQL is that it is that it is inexpensive.

Slack

Our team has decided to approach the project from an Agile Scrum paradigm. The agile approach to software development provide many benefits which will be discussed in later chapter. A key component necessary for the agile to be successful is constant communication. The need for reliable communication lead our team to use Slack as our primary means of communication. Slack is a nice communication tool that allows team members to communicate effectively. Our team uses slack to store links, general spur of the moment design discussions, and as reference for each team members progress. Slack also integrates well with other tools our group will be using, such as git. Slack will automatically broadcast team member commits to the entire group. The commit broadcast will help the team track progress and stay on course with our progress projections.

Jenkins

Version: 2.7.4 Debian for Ubuntu

Jenkins is an open source server that our team will use to automate our workflow. Our team will set up a Jenkins server running on one of our two Ubuntu servers hosted by UCF. Jenkins will allow our team to streamline the build and testing process. The Jenkins server will be connected

to our github repository. Every night on a strict schedule, Jenkins will pull down the latest copy of our baseline code. Once Jenkins has the latest code, Jenkins will build the entire system logging any errors or issues that arise. If the system compiles and builds correctly, then Jenkins will run the entire unit test suite and log any issues. Using Jenkins this way will ensure the integrity of the master branch on github. This is also a great way to verify unit testing is being done properly which is necessary for effective testing of our system.

Ubuntu

Version: Server LTS 16.04

Ubuntu is a free open source Linux distribution which we choose to use on our servers on campus. We choose to use the long-term support version to ensure the stability of our system. Ubuntu server comes with the necessary server software our team will need to develop the web server and provisioning server. The most important reason why we chose Ubuntu is that the operating system is well documented and offers plenty of support online. Being one of the most popular Linux distributions, Ubuntu online community offers plenty of forums and examples that our team may need to overcome any issues in the future.

OpenLayers

Version: 3.19

OpenLayers is an open source map framework that can be used to connect to multiple map tile servers. OpenLayers framework provides all the necessary layers our team will need to display the data in a meaningful way. One requirement given to the group by Dr. Chatterjee is to display the live dynamic data as a heat map of signal strength for the give frequencies. OpenLayers provides a heat map overlay which our team will use to meet this requirement. Another benefit to using OpenLayers is the availability of their documentation and the examples on their website. OpenLayers posts working examples using all the features available.

Node JS

Version: 4.5
Node is a JavaScript asynchronous web server that our group decided to use for our project. Typically, webservers handle http request by spawning threads to handle the processing. Node is different because node issues promises which are callback functions that the client will call when the processing is done. This paradigm shift offers a few benefits over traditional web servers. Nodes approach allows Node application to be stateless which means that the backend application does not hold any information for each client. Being stateless allows Node to be highly scalable. We choose Node for our backend of our web application because of its high scalability which would be a necessity if Dr. Chatterjee moves forward with his research. Another reason why we choose Node is because using Node is a JavaScript library which is desirable since now JavaScript will be the language use at every layer of our web application.

Express JS

Version: 4.14
Express is a JavaScript framework for developing Node applications. Express will be used to build the middle layer of our web application. Express applications are written in JavaScript and are ran inside the Node server. When an HTTP request comes in Node handles that request and passes it the Express application. Express consists of routes which are used to serve up tokens for authentication or webpages. Express is commonly used in conjunction with Node because of how well they work with each other. Express and Node keep the functionality of the web application modular which will help with maintenance, testing, and debugging.

Angular JS

Version: 2.1.0

Angular is the next layer of our application. Angular is also a JavaScript framework used in developing websites and web applications. Angular is commonly used with Node and Express, and is an important part of the MEAN stack. Angular framework is used for developing the frontend of web applications. The framework utilizes the model, view, and controller structure for the frontend of web applications. The model part of our angular app will be responsible for communication between the frontend and the backend. The controllers are part of the Angular application that is responsible for serving up views and controlling the model. Views are a collection of html pages that will be served up the client's browser. The modularity of this design helps with maintenance and troubleshooting issues. We choose Angular because of how well it works with the other layers of the MEAN stack. Angular also provides a large collection html structures. The benefit of using Angular's structures is that the they provide a convent two-way data binding. When the user makes changes on the front end the underlying data structures are automatically updating and pushed to the back end.

JQuery

Version: 3.1.1

JQuery is another JavaScript library that our team will be using the web application. JQuery will be used along with vanilla JavaScript to build the functionality of front end html views for our web application. JQuery is not necessary to use for event handling and front end DOM manipulations but we decided that jQuery would be used for its convenience. With jQuery, the syntax of our JavaScript is more intuitive and clean. JQuery is easier to read and, therefore, more easy to maintain and debug. Also, jQuery provides cleaner way to make ajax request to our back end API. Open Layers also has dependencies that rely on jQuery.

Docker

Version: 1.12.3

Docker is a tool for wrapping software applications into a complete filesystem that contains all the necessary files, libraries, and source code necessary to run the application. These filesystems, called containers, operate on a single operating system kernel so they are more lightweight will still retaining their sandboxing similar to a VM. Additionally, since each f the environments are identical, the team can avoid inconsistencies between the installed libraries on the system allowing

for more accurate testing. Docker will be used as a way to rapidly spin up and deploy our simulated sensors for testing.

Bibliography

Schwaber, K., & Sutherland, J. (2013, July). *The Definative Guide to Scrum: The Rules of the Game*. Retrieved from Scrum Guides: http://www.scrumguides.org

Rafael Micro R820T High Performance Low Power Advanced Digital TV Silicon Tuner [Digital image]. (2011). Retrieved from http://superkuh.com/gnuradio/R820T_datasheet-Non_R-20111130_unlocked.pdf

Linear Technology LT5538 [LT5538 40MHz to 3.8GHz RF Power Detector with 75 dB Dynamic Range]. (2008). Retrieved October 01, 2016, from http://cds.linear.com/docs/en/datasheet/5538f.pdf

RFM22B/23B ISM TRANSCEIVER MODULE [Digital image]. (2006). Retrieved from https://www.sparkfun.com/datasheets/Wireless/General/RFM22B.p (Saptarshi Debroy, 2015)df

Saptarshi Debroy, S. B. (2015). Spectrum Map and Its Applicationin Resource. IEEE TRANSACTIONS ON COGNITIVE COMMUNICATIONS AND NETWORKING, 406-419.

Silvertel Ag9800 Miniature PoE Module [Power over Ethernet module]. (2014, December). Retrieved October 1, 2016, from https://www.codico.com/shop/media/datasheets/Ag9800M-datasheet.pdf

W5100 Datasheet v1.1.6 [Digital image]. (2008). Retrieved from https://www.sparkfun.com/datasheets/DevTools/Arduino/W5100_Datasheet_v1_1_6.pdf