5-2018

Designing a Distributed Network of Air Quality Sensors

Kaylee Rauso
University of Arkansas, Fayetteville

Designing a Distributed Network of Air Quality Sensors

An Undergraduate Honors College Thesis in the

Department of Computer Science and Computer Engineering

College of Engineering

University of Arkansas

Fayetteville, AR

by

Kaylee Rauso

This thesis is approved for recommendation to the Honors College.

_____

Christophe Bobda, Ph.D.
Thesis Advisor

_____

Patrick Parkerson, Ph.D.
Committee Member

_____

Alexander Nelson, Ph.D.
Committee Member

Abstract

The deterioration of human and environmental health due to worsening outdoor air quality is a serious global threat. A step towards improving the current state of ambient air worldwide is the increased collection of data for government policy, research, and general public use. Currently, reliable data is only collected at a handful of sites in the state of Arkansas and sparsely across the country. The purpose of this project was to design an inexpensive air quality sensor to saturate entire cities with nodes which transmit air data to a sensor for a web application to display. The prototype node was designed and implemented as a proof of concept with a projection for a node optimized for size, power, cost, and development time. The node uses solar power to charge a lead acid battery in order to support a Raspberry Pi Zero W equipped with air quality sensors using a FONA module to connect to the 2G network to transmit TCP packets to a Python server. The final prototype uses an automated script to collect sensor data, compile the data into a packet, and send to the TCP server to be inserted into the database. The web application uses REST endpoints on the server to execute HTTP requests by the application for data to populate the data and map pages. For future work, system testing should further optimize the prototype node as well as bring the nodal network into the Internet of Things movement.

Table of Contents

List of Figures

1.0 Introduction

The average healthy adult at rest takes 12 to 18 breaths per minute [1]. Over the course of only one year, that can be up to 9.5 million breaths, and at approximately one half of a liter of air per breath, a person can inhale and exhale over 4.7 million liters of air in that year [2]. That volume is nearly twice that of an Olympic-size swimming pool [3]. Given such large quantities of air infiltrating our bodies, it should be indisputable that people have the available resources to monitor the quality of the air in their area. Yet, if one were to investigate what the concentration of ozone was in their city yesterday, they would be challenged to find any collection site or reputable data within a useful distance. In the state of Arkansas, reputable and well-recorded outdoor air quality is only widely available from one source, the Arkansas Department of Environmental Quality, and is limited in locations and pollutants [4]. Ozone readings are available for 7 locations statewide, and particulate matter readings are available for 5 locations. The more comprehensive Air Quality Index (AQI), used to generate a health standard for air quality based on five different pollutants, is only available for Little Rock and Springdale. In both the state of Arkansas as well as across the globe, there is a lack of ample air quality data both for personal and research use. The purpose of this project was to design and implement an inexpensive, scalable distributed network of sensors which can help alleviate this lack of environmental data and bring an open-source quality to the collection of meaningful pollution concentrations. The system could be deployed on a small scale for the citizen scientist with a personal interest in local air pollution levels or on a large scale to create a highly dense nodal network of sensors to make conclusions on higher quality data.

2.0 Background

2.1 Problem

In order to justify a solution for a lack of air quality data, the necessity for obtaining air quality data must be explored. The negative consequences of air pollution can be categorized into two main detrimental effects: the deterioration of human health and the toxifying of the environment. Both outcomes are worthy of a vested interest from audiences ranging from lawmakers to physicians to the general public. Whether directly or indirectly, the effects of air pollution impact all people.

Exposure to air pollutants contribute to both short and long-term health problems in humans. In an instant, the particulate matter (PM) in polluted air aggravates the respiratory tract resulting in coughing and wheezing, and the nitrogen oxide irritates the eyes, nose, and throat [5]. Given prolonged exposure, many pollutants have life-threatening consequences for humans. Sulfur dioxide, nitrogen oxide, particulate matter, and lead all deteriorate human bodily systems in high enough concentrations [5]. Sulfur dioxide, nitrogen oxide, and PM increase the sensitivity of the respiratory and cardiovascular systems making those exposed prone to infections, asthma, chronic obstructive pulmonary disease (COPD), and reduced lung and heart function [5]. Children and the elderly are especially susceptible to the increasing the probability of developing a sensitivity as well as suffering from an ailment due to that sensitivity [5]. Lead is a neurotoxin for infants and children even in low concentrations that severely deteriorates the nervous system [5]. Exposure to high enough levels of lead or even prolonged exposure to low levels accumulates lead in the body resulting in lead poisoning which disrupts the normal function of nearly every bodily system [5]. Additionally, the long-term, toxic effects of the pollutants can manifest in complications of the skin,

immune system, and eyes [5]. According to the World Health Organization, in 2012, one in nine deaths around the world had an air-pollution related cause, and outdoor air pollution was attributed as the sole factor in three million of those deaths [6].

In addition to the detrimental impact on human health and life, air pollution is a poison to the ecosystem. Sulfur dioxide and nitrogen oxide are gasses that dissolve into the water vapor in the air which fall back to the earth as acid rain [7]. The higher acidity levels of this precipitation are toxic to the plants and animals that live in the soil and bodies of water in which this rain falls [7]. The acid rain pulls aluminum out of the soil which is harmful for plants and animals [8]. Many species of organisms in aquatic environments have a pH that they can survive, and acidifying their water source can eliminate whole populations [8]. This acid rain, in addition to an increase in ground-level ozone, can damage crops reducing yields [7]. Increases in these outdoor air pollutants contribute to the greenhouse effect of the atmosphere which has been cited as evidence for global climate change, a concern for all forms of life on Earth [7]. The cost of outdoor air pollution affects all people. It should be not only pursued but pursuable for concerned citizens.

2.2 Purpose

An important step towards stopping and reversing the sources of outdoor air pollution is the collection of more data. As mentioned before, the Arkansas Department of Environmental Quality only takes regular air quality data from at most seven locations across the state. This generalizes entire regions of the state's air quality in very minimal data sets. When pollutants can be carried hundreds of miles by wind, seven retrieval locations for an entire state is simply not sufficient for making conclusive statements that incite change [7].

Therefore, it is imperative that computer technology be used not only as an end but a means to an end that may not be entirely related to computing technology. While many envision the future of computer engineering will be further delving into specialized fields, there are many necessary and stimulating applications in interdisciplinary realms. This project aims to design and implement a distributed system of air quality sensors which collects data on a local scale and is intended to be public facing to encourage more interest in investigating air pollution.

As an overview, this system comprises a series of nodes that collect air quality data and store the data on a TCP server. A node consists of a Raspberry Pi Zero W, a small, low-cost computer, upon which air quality sensors are attached. The sensors measure the concentration of their respective pollutant in the air and store the values upon the Raspberry Pi. The Pi is used to transmit the data via TCP packets to the server hosted on the University of Arkansas Nebula platform. The Pi accesses the 2G cellular network in order to send the packets to the server. Data is stored in a database using the Cassandra framework which is designed for high scalability. The node hardware is contained in a weatherproof case to protect the electronic components from weather and tampering. However, the air quality sensors must be exposed to ambient air in order to measure pollutant concentrations, so the case must be designed to allow air flow while protecting the physical components. The node will be powered by a battery which should be charged using a solar cell mounted to the case. The amount of power the cell can retrieve and the battery can contain are design constraints that must be accounted for when constructing the node and the size and frequency of packet transmission to the server.

2.3 Related Work

Designing air quality monitoring devices is no new technological feat. Researchers who recognize the importance of safe, clean air are trying to develop devices and methods for accurate and affordable pollution detection. There are several commercially available options which meet many of the design goals set for this project. Major variable factors include cost, available pollutant sensors, and power source options.

At one end of the scale for outdoor air quality systems is the Landtec AQMesh gaseous monitor [10]. First generation monitoring units measure carbon dioxide, nitrogen oxide and dioxide, sulfur dioxide, and ground-level ozone all in one unit using electrochemical sensors and costed approximately $10,000 in 2015 [11]. Newer models have been released since that also measure PM and can be battery operated with an option to add solar power as a power source [10]. The collected readings are available to the owner of the monitor via an online portal which display numerical, charted, and graphical representations of the gathered data [10]. A marketing feature of the AQMesh system is the publication of comparison trials between their pod and air quality monitoring reference stations in the United States and multiple European countries which show that AQMesh pods can provide comparable air quality data readings to established retrieval stations at a lower cost [10]. However, even if a single AQMesh unit is significantly cheaper than a government sanctioned air quality monitoring station, it is still far too expensive to implement as a dense network of sensors for highly localized data.

Another higher end option is Aeroqual's Ambient Air Monitoring Stations [12]. Products from this company are used by Johns Hopkins University, NASA, Samsung, and other high-profile clients [12]. Their sensors are research and government quality, and they were featured

as one of the few manufacturers brought to attention in the US EPA's 2014 Air Sensor Guidebook [12]. While they are an elite choice for outdoor air quality measurement devices, their products come with an elite price tag. The Aeroqual Series 500 Monitor, which is a portable monitoring system with exchangeable sensor heads, retails for $1,270 to up to $2,425 depending on which sensor head is purchased with the monitor [13]. Purchasing an enclosure for the monitor to convert it to a fixed location sensor costs an additional $415 [13]. While one monitor can measure multiple pollutants, it can only detect one at a time corresponding to the attached sensor head. Multiple monitors would be necessary in order to obtain data on several pollutants at the same time, and at more than one thousand dollars apiece, that does not reasonably fix the budget of the citizen scientist.

A commercial alternative that approaches both the cost and design goals to EarthSense is PurpleAir. The purpose of PurpleAir is to build a community driven network of air quality monitors to upload real-time pollution data to their map online [14]. PurpleAir is an Internet of Things device which uses a control board to average the particle counts of six different sizes of particulate matter between 0.3 and 10 micrometers to estimate the total mass of PM1.0, PM2.5, and PM10 [14]. A single PurpleAir unit costs $229 and includes the PM, pressure, temperature, and humidity sensors on an Arduino with Wi-Fi connectivity for data transmission and the means to power the device [14]. Adding an SD card and real-time clock for offline data storage with increase the price of a single unit to $259, but buying units in bulk reduces the unit price [14]. Additionally, PurpleAir is only equipped with particulate matter sensors. While PM counts are extremely relevant, especially when considering the effect of air quality to human health, EarthSense aims to be a more holistic air quality monitoring system. Supplied power source options are limited to wired 5V DC connections.

The unit must also be connected to Wi-Fi to automatically load collected data to PurpleAir's server, or the user must purchase the additional PurpleAir package equipped for offline sensing and manually retrieve the data from an SD card.

The strengths and shortcomings of existing systems help shape the design goals of EarthSense to create a unique, well-rounded solution for a target audience without a government funded budget or wishes to deploy a highly dense network of sensors.

3.0 Design

3.1 Previous Work

This project concept was proposed last year by PhD candidate Taylor Whitaker which is when initial project design began. After analyzing the problem and how other commercially available solutions have approached and executed their products, we developed the high-level architecture for EarthSense. This included determining what sensors were available and appropriate for this project. It was established that there were nine applicable air quality sensors plus barometric pressure and temperature sensors which would be applicable to include. However, considering power and size constraints, it was decided that 11 sensors on one node would be excessive. Thus, several prototype nodes were designed by grouping three to five sensors per node based on the pollutants' primary negative effects: soil and water toxification, the greenhouse effect, and human health. An additional node was devised to have sensors in a combination of categories for more general-purpose sensing. With these initial strategies, the first node prototype was developed using a Raspberry Pi 2, a breadboard, and sensors for carbon dioxide, propane, carbon monoxide, and particulate

matter. The Pi was powered from a wall outlet using the microUSB port. Data transfer and server connectivity was fulfilled using the Ethernet port.

The server side of the EarthSense was meant to receive and compile the node sensor data into a database which would be retrievable by a web application responsible for displaying the data using charts and maps. Preliminary proposals for node-server communication was to use MQTT protocol to correspond the sensor data to the server. MQTT is a communication protocol built on TCP/IP communication protocol and follows a publish/subscribe procedure for lightweight message passing [15]. The EarthSense server would be written in Python and would be responsible for hosting the MQTT broker Mosquitto which received the MQTT messages from the nodes [16]. The Python server would also insert sensor data into the Cassandra database. Cassandra is a database management system designed for scalability developed by Apache [17]. Scalability is a critical factor to consider since a highly dense network of air quality sensor running for an extended period of time can quickly generate massive amounts of data. Finally, the python server would also host the front-facing web application which would provide node users and the general public with a platform to view and analyze the data gathered by the network. This included numerical tables, line graphs over time, and location-based maps. The web application would originally be developed using the Ionic framework, a mobile SDK for developing web applications [18].

Over the summer, initial implementation of the general design was executed by a team of international students the university hosted over the summer while I was at an internship. After coming back from the summer, the progress was evaluated, and the initial design reviewed. Further conversation about the goals of the project, limitations due to design

constraints, and personal preference caused some changes to their product. Some parts were kept, some were modified, and some were rebuilt.

A major distinguishing factor between the EarthSense and other commercial solutions is cost. Keeping expenses low for the production of a node was at the forefront of the hardware design. Therefore, the Raspberry Pi 2, which retails for about $35, was exchanged for the Raspberry Pi Zero W at just $10. Additionally, buying sensors individually started to become costly and spatially inefficient. Therefore, the design which would utilize many individual sensors was replaced with two sensors: a Sensly HAT and a barometric pressure, temperature, and altitude combination sensor. The Sensly HAT is a product by Altitude Technology which hosts three different gas sensors on one small chip the same size as one normal sensor [20]. Furthermore, the Sensly was designed to seamlessly integrate with the Raspberry Pi to easily retrieve sensor data. This had been a problematic drawback of using individual sensors because some sensors required conversions to readable data. The three gas sensors on the Sensly are also non-calibrated, so different HATs can be calibrated to detect different sets of gases replacing the need for multiple node designs.

Another modification from the original design was the node-to-server communication protocol. Originally MQTT, it was switched to TCP protocol which is the packet-based protocol MQTT is built on. The Adafruit FONA cellular module, used to access the 2G cellular network from the Pi, supports seamless TCP integration with manufacturer-provided documentation for all TCP commands [20]. No additional libraries are necessary to execute the commands directly from the Pi. TCP packets are small in size, so using TCP instead of MQTT is still a lightweight messaging option which was the reason why MQTT was originally chosen. Consequently, the MQTT broker Mosquitto on the server would need to be

modified as well. A Python TCP server replaced Mosquitto as the message broker. Its purpose remains the same; the TCP server receives node data via TCP packets, parses the individual segments of data, and inserts each entry into the Cassandra database.

After analyzing the purpose and target audience of the EarthSense network, it was concluded that the web application needed to be rebuilt. Ionic is a framework designed to build native and web applications for mobile devices very well but does not integrate as easily when constructing web applications for desktop. Since the main function of the web application is data presentation in the form of tables, graphs, and maps, it became clear that users were more likely to be at their desktop than their mobile devices. Being large, detailed, and interactive visual displays, it was determined that users would be more inclined to utilize the application on a larger screen. Therefore, the employment of the Ionic framework was dropped in favor of an AngularJS application. AngularJS is a TypeScript framework that augments the web standard HTML-JavaScript-CSS format [21]. It modularizes the application's structure to reduce complexity, facilitate component reuse, and route views. It also expands the capability of HTML by allowing Angular to modify HTML directly. In order to interact with the Cassandra database from the Angular application, a Java RESTful API using Spring framework was added to the server. REST architecture utilizes the HTTP protocol to perform requests from the web application for node, sensor, and user data to populate pages or modify data. In this way, the web client has a simple interface through which to interact with the database without having to understand the underlying structure.

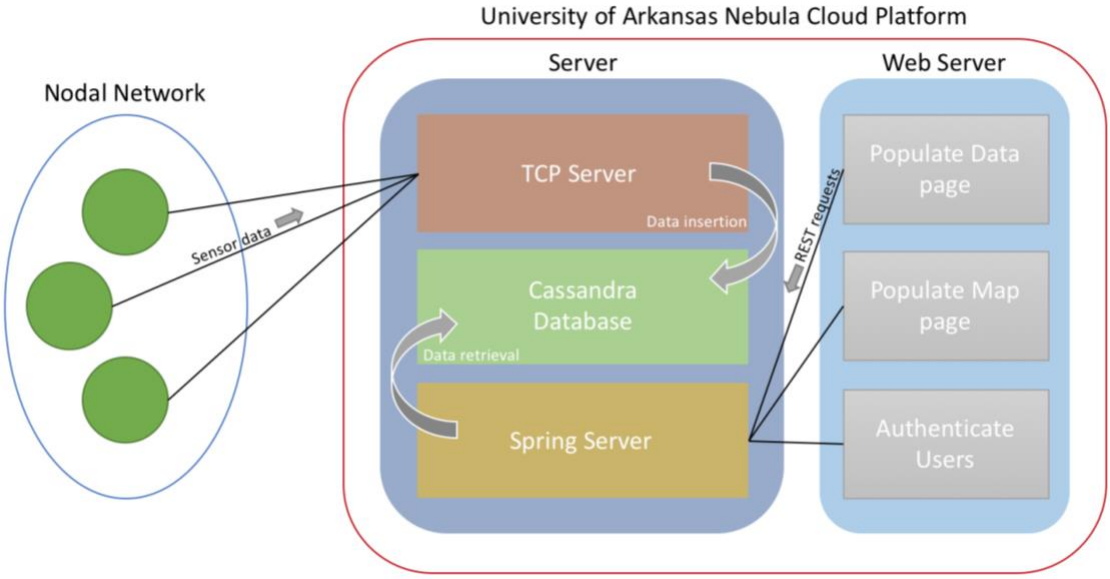

Figure 1: Overall Design Flow

3.2 Design Constraints

Cost

One main restriction on this project's design was cost. The sensors used by the EPA to make official air quality measurements are priced around $20,000 for a single unit [22]. Even supposed "low-cost" air quality monitors available for personal use were more than $1,000 per unit. Only PurpleAir's particulate matter sensor was remotely affordable, and it traded its range of measurable pollutants for its cost and had very little, if any, room for customization. Therefore, many design choices were made to minimize total cost while keeping the node's usability general purpose. Raspberry Pi Zero computers are powerful and adaptable devices at a low price of only $10 which made them an ideal foundation for the hardware [22]. The selection of one Sensly HAT over several individual sensors is one instance of compromising cost and flexibility of design. While it would be possible to buy three sensors for less than the

cost of one Sensly, which was about $90, the design would be less adaptable for the user, in addition to consuming more space and using more pins on the Pi [19].

Connectivity choice was another result of a cost-benefit analysis. The Raspberry Pi Zero W comes with Wi-Fi and Bluetooth connectivity built in, so implementing message passing using one of those networks would not incur added costs. However, the user is then restricted to placing the nodes in areas with reliable and accessible Wi-Fi. Using the 2G cellular network for node-to-server communication allowed node placement to be less restricted to urban areas where Wi-Fi availability would be most prevalent. The Adafruit FONA cellular module costs about $50 to allow the Pi to transfer GPRS data, including TCP packets, over the 2G network [22]. Additional costs to use the cellular network include a SIM card ($9), a lithium ion polymer, or lipoly, battery and microUSB charging cord ($8-10), a GPS antenna ($5), a GSM antenna ($3), and a subscription to a 2G network provider for the cellular connectivity ($9/month for 100MB) [23][24]. These could all be omitted to cut costs if the user had planned on placing the node within reach of a Wi-Fi network.

Due to the modular nature of Raspberry Pi based projects, many costs could be reduced by exchanging components based on need. For the most basic weather monitor, the Sensly could be replaced with a simple temperature and humidity sensor at a fraction of the cost. The FONA chip could be replaced with a Wi-Fi connector for a lower cost or even an Ethernet cable if available. However, for the EarthSense implementation, the final product should be general purpose and capable of reporting real-time data in remote locations.

Size

While there is no requirement of size for a functional air quality monitor, the design was still optimized to be as small as possible. A smaller node is easier to setup, manage, and less of a disturbance to its environment both ecologically and visually. Since a major feature of the node is its adaptability of purpose by user, it should be able to be placed almost anywhere from a window sill to a parking lot to a campground. The less bulky the design, the more placement options it will have.

Choosing a Raspberry Pi Zero and Raspberry Pi compatible sensors attributed to minimizing the space needed by the hardware. A Pi Zero is only 2.6" x 1.2" x 0.2" and weighs just 0.3 oz [23]. Individual sensors, like the temperature and pressure sensor by Adafruit, are smaller than a quarter [23]. The Sensly HAT comes in at around 3.0" x 2.5" and fits directly on top of the Raspberry Pi connected by the GPIO connectors. These components were chosen as their contribution to total increase in size is minimal compared to added capabilities of the node.

However, like cost, sometimes small size had to be compromised for functionality. Using the Adafruit FONA module is a small sacrifice of space due to the addition of the FONA chip, the lipoly battery and microUSB cord, and the sticker-type GSM antenna. The FONA module measures in at 1.75" x 1.6", the lipoly battery at 1.3" x 2.4" x 0.2", and the antenna at 3.0" long and not even 0.1" in thickness [23]. As mentioned before, this tradeoff allows the node more freedom of location, a valuable asset to this design.

Unfortunately, not all supplemental features of this design have quite as low of an impact on total size of the node. To include solar power as a power source for the Pi required various hardware components to be added to the design. A solar panel, a regulator, and a lead acid

battery are necessary to provide the appropriate current and voltage to the Pi safely and consistently. These additions significantly increase the size of the overall hardware design. Therefore, it is imperative that the final design be optimized for suitable but not excessive power collection so that the complete size of the node is no larger than necessary. Since the Raspberry Pi Zero's supplied power adapter inputs 5V, the lead acid battery should be able to supply the 5V to the Pi as well as charge the FONA's lipoly battery, and the panels should be able to produce enough voltage to charge the lead acid battery.

Power

Without power, there is no EarthSense. The minimum output voltage of the lead acid battery is determined by the power drawn by the Raspberry Pi with the sensors and the FONA module with its lipoly battery. At night or during cloudy weather, the lead acid battery must be capable of powering the node circuitry to maintain data collection and network connectivity. Therefore, the solar panels must be able to provide the charge voltage for the battery. The power adapter supplied with the Pi provides 5 volts, and the FONA module is equipped with level shifting circuitry which allows the FONA to run at voltages between 2.8 and 5 volts [23]. The onboard microUSB ports on the Raspberry Pi and the FONA are both rated for 5V inputs which many projects typically supply using a wall outlet power supply. So, if the battery can provide each of the components with 5V, functionality can be assured. Since lead acid battery nominal voltages are 2 volts per cell, it would be necessary to use at least a 3 cell, 6V lead acid battery to provide the 5V necessary to power the components [25]. The charge voltage limit for lead acid batteries is from 2.3 to 2.45V per cell depending on the particular battery [26]. Hence, in order to charge the battery, the solar cells

must be able to generate approximately 6.9 to 7.35V to the lead acid battery. Allowing for the linear dropout from the regulator will increase these values depending on the specific regulator.

Additionally, the necessary battery run time must be calculated. At an idle state, the Raspberry Pi Zero W draws about 120mA and draws about 230mA at the same voltage using very demanding peripherals [31]. The Pi would spend most of the time idling between taking and transmitting readings, but to err on the side of caution, the estimated current draw for the Pi is the average between the two at 175mA. The FONA can run on standby for 1-2 days on a 1200mAh lipoly battery [23]. Cautious estimates for just 1 day of use result in an approximate 50mA current draw. The FONA can draw 200mA during data transmission [23]. TCP transmission times are on the scale of milliseconds, and data transfers from the nodes are only ever hour. To give estimate generously, if the FONA spent 5% of run time in data transmission, current draw would be approximately 58mA. Therefore, the lead acid battery would need to supply about 233mA total for the Pi and the FONA. Not every day will have the maximum amount of direct sun hours, so the system should be able to run from the battery for at least two days without additional charge from the panels. With a current draw of 233mA, or 0.233A, and a run time of at least 48 hours, the battery would need to provide at least 11.18Ah. Rounded up to conventionally and commercially available amp-hours would be 12Ah. Therefore, a 6V 12Ah lead acid battery would be able to power this design.

However, it may be that all 5V is not needed to run the Raspberry Pi and FONA chip. The processor and SD card on the Raspberry Pi Zero run at 3.3V, and a voltage regulator on the Pi creates this voltage from the 5V input [27]. The Pi is designed to accept a 5V input, more specifically 4.75 to 5.25V input, because that is the standard USB range [27]. Therefore,

it should be possible to power the Pi on as little as 3.3V. As peripherals are added, such as the air quality sensors, some headroom would ensure that the voltage to the processor is always sufficient. As previously established, lead acid batteries are sold at even-numbered nominal voltages, so a 4V lead acid battery should be able to provide the necessary minimum 3.3V to the Raspberry Pi. The FONA module is marketed to function on voltages as low as 2.8V and the accompanying lipoly battery has a nominal voltage of 3.7V. So, an output voltage from the lead acid battery of 4V should be able to power the FONA and charge the lipoly battery to partial capacity. With a 2 cell, 4V lead acid battery and a charge voltage limit of 2.3 to 2.45V per cell, the necessary voltage required from the solar panels is reduced to 4.6 to 4.9V after accounting for regulator dropout. These reductions to the lead acid battery and the solar panels further benefit both the aforementioned size and cost constraints. The 4V battery is smaller and lighter than its 6V counterpart as well as being cheaper. Reducing the solar panel voltage requirement also lowers the size, weight, and price for the solar panel partition of the design as less or smaller panels are necessary. The tradeoff for lowering the input voltage to the Raspberry Pi is current draw. The Pi Zero will have to engage switch mode for its voltage regulator which pulls more current than running at the standard 5V. In this case, the 12Ah estimated battery runtime might not be sufficient to supply the system for two days. System testing at lower voltages would be necessary to determine the magnitude of the effect on current the voltage regulator has since this is not a well-documented implementation for the Raspberry Pi Zero.

For this optimized design, size and cost of a node is likely reduced. Though, implementing these changes might not be suitable for some instances. If the node is placed in a location with a small average amount of peak sun hours, reducing the solar panels and

battery might not be an option. The size of the solar panels would need to increase to gather more energy, and the lead acid battery would increase in capacity to maximize the useable time for energy gathered. Additionally, if the system leaks too much voltage or current, there might not be enough headroom from the solar panels or the battery to deliver sufficient power to the Raspberry Pi with the sensors or the cellular module with the lipoly battery. In these cases, the solar panels and battery would need to be increased to produce and retain more power.

Development Time

Consideration of development time as a design constraint would have likely been overlooked had it not become a problem during the implementation process. Nearly all of the products used in the hardware design can be shipped from domestic retailers such as Adafruit, found on Amazon or other online retailers with short shipping times, or purchased in electronics stores. The exception to this is the Sensly HAT from Altitude Technology. Altitude Technology is based in Britol, England and is the only reputable vendor for Sensly HAT products. It has been several months since the Sensly HAT was ordered online, and it has yet to arrive. Since scalability and replicability are design objectives for EarthSense, obtaining the necessary components for the nodes should be as quick and simple as possible. Waiting several months for a key element meets neither of those conditions. Therefore, for prototype implementation, the Sensly HAT was traded for individual sensors per the original design.

3.3 Node

A single node consists of three components: hardware, software, and protective casing. Components in hardware are the electronic devices necessary to collect and transmit data and the power supply system to support those electronics. The software delegates the collection and transmission of data so the node is self-sufficient and needs very little human interaction. The casing protects the hardware from the environment including weather and tampering.

Hardware

The hardware partition can be categorized as either for data collection and transmission or power supply. Data-related components include the Raspberry Pi Zero with attached air quality sensors and the FONA cellular module with lipoly battery. The power supply components consist of the lead acid battery, the regulator, and the solar panel. Beginning with the data components, Adafruit sensors, such as the BMP280 barometric pressure and temperature sensor and the SHT31-D temperature and humidity sensor, transmit their sensor reading to the Pi using the I^2C communication protocol. I^2C only requires 2 signals, clock and data. The SCK clock input pin on the sensor connects to the Pi's I^2C clock line, and the SDI data pin on the sensor connects to the Pi's I^2C data line. There is only one I^2C clock and data line on the Pi, so multiple sensors using the I^2C protocol must share the pins. All of the sensors can be directly connected as long as they do not share the same I^2C address. In the event of sensors competing for addresses, an I^2C bus multiplexer would have to be added. Regarding power pins, the sensors must be attached at V_{in} for power to sensor and GND for power and logic grounding. Sensors should be powered at the same voltage as what the microcontroller uses for logic. For the Raspberry Pi, this is 3.3V. Due to the Sensly HAT

never arriving, last minute design changes to the attached air quality sensors have to be formulated.

Like the sensors, the Adafruit FONA cellular module must be powered at its V_{io} pin at 3.3V and its GND pin grounded. Added to the FONA is the Key pin which ties to a button on the module to turn the module on and off by holding the button for 2 seconds. The FONA module uses UART protocol for communication with the Pi. UART communication uses two lines on the module to transmit and receive serial data between devices, RX or receive and TX or transmit. The RX pin is the data line from the Pi into the FONA module, and the TX pin is the data line out of the module to the Pi. The RX pin on the module connects to the TX pin on the Pi and vice versa in order to match data flow. Additionally, accessing the full functionality of the FONA necessary for the node requires the use of several connectors and ports. The microUSB connector is used to charge the lipoly battery which inputs to the module at the JST 2-pin connector. The SIM card connector on the back of the module takes a 2G Mini SIM which allows the FONA to access the 2G cellular network. A Ting SIM card and service plan were chosen for their inexpensive cost. A GSM antenna port accepts the cellular antenna used for network access. The GPS antenna port accepts a passive 50 ohm GPS antenna to provide location data. The module uses the UART lines to transmit this data and receive commands.

The power supply components are responsible for collecting, storing, and safely distributing power to the data components. Solar panels are responsible for the collection portion of the system. For a prototype and proof of concept implementation, the solar panels are larger than theoretically necessary to allocate headroom for power collection under non-ideal sun exposure. Two different types of panels are available for implementing the design,

a 12 volt nominal, 20 watt peak panel and a 6 volt nominal, 6 watt peak panel. Both will be tested to evaluate how real-world variables impact the design. The panels can be connected in parallel, series, or a combination of the two in order to produce the appropriate voltage and current to the system.

Solar panels on their own do not output consistent voltage. In direct sunlight, solar panels can generate far more voltage than their nominal rate, and output voltage sags as a cloud passes overhead. If the solar panels connected directly to the lead acid battery, the panels could overcharge the battery in peak sun hours and then drain the voltage from the battery at night. The panels also cannot directly power the data components as it would cause surges and starvations that damage those devices. These obstacles expose the necessity for a regulator, also known as a solar charge controller. The regulator adjusts the voltage output to the battery to prevent surges, stop flow if the battery is fully charged, and block voltage from discharging from the battery. The Raspberry Pi Zero and the FONA module are also connected to the regulator to collect their power so as to receive stable levels. The USB ports on the regulator deliver the standard 5V which recommended for both the Pi and FONA.

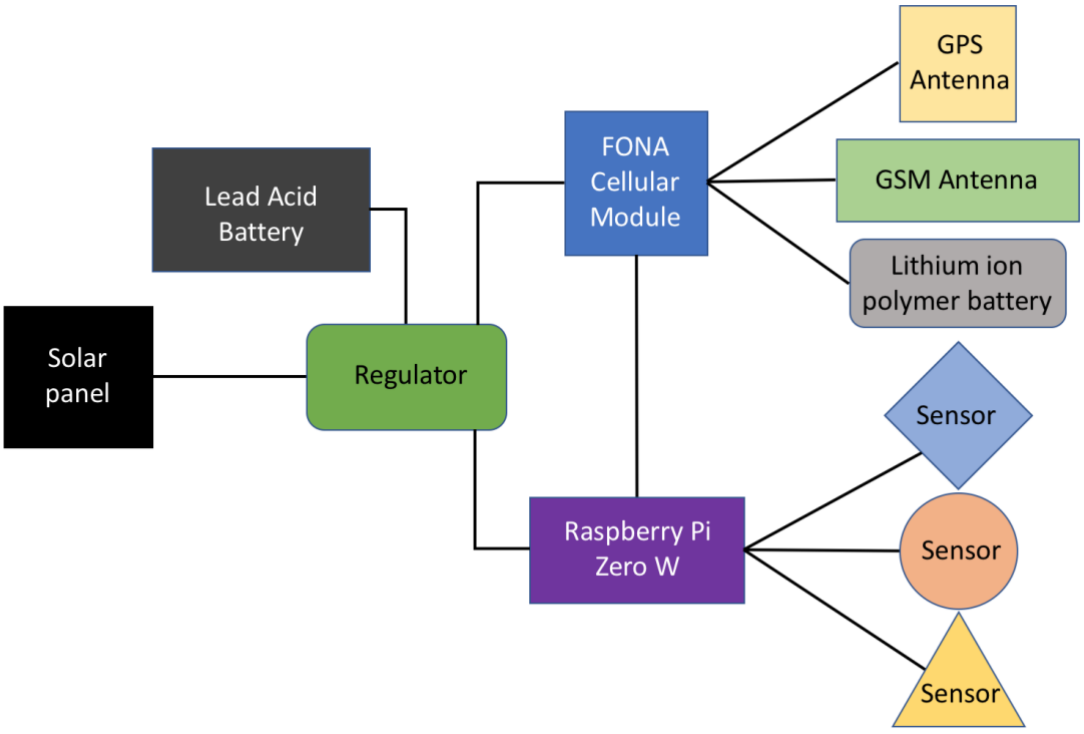

Figure 2: Node Hardware

Software

With the hardware set up to receive data day and night, it must have the supporting software to give commands to the peripherals, translate the inputs from the sensors into usable data, and convey that data to the server. The server expects TCP packets that contain the unique UUID for the node, a timestamp for when the readings were taken, and value for each of the sensors at that time. High level language scripts run by the Pi are employed to communicate through its data pins to the attached components. Many Adafruit sensors, including the BMP280 and SHT31-D, have libraries available from GitHub which can be imported to easily convert the data from the I^2C data line into the expected format for the particular sensor. This implementation utilizes Python libraries to interpret the incoming sensor data. Check accompanying documentation for units and other sensor specific information in order to correctly interpret the outputs.

The sensor data is transmitted to the server via TCP packets over the 2G cellular network using the Adafruit FONA 808 module. Like the sensors, the FONA module also has available Python libraries in order to connect the Pi to the FONA and the FONA to the internet via the 2G network. The FONA responds to a set of instructions called AT commands which are used to make cellular connection and send and receive data from remote and peripheral sources from the module. Once internet connection has been established, collected data can be sent to the server. Data flow from the sensors to the server follow a general path. First, the Raspberry Pi establishes internet connection using the FONA over the 2G network. Next, readings are recorded from the sensor peripherals using the Python libraries to render the data. Then, the UUID which identifies the node, a timestamp for the current time, and all of the readings separated by commas are written to the server using a socket. For most node to server interactions, this is the extent of the flow.

The exception is instances when the TCP packet should include GPS data from the FONA. On a high level, the same process is implemented: connect to the internet, save a value from the peripheral, write the value to the server. However, when the data must be retrieved from the FONA, this process must be interrupted. This is because the cellular connection and the transmission of GPA data from the FONA to the Pi both require the UART data lines and cannot be used for both the data collection and transmission simultaneously. Therefore, steps have to be added to the work flow. Before data collection, the regular serial connection must be suspended before using AT commands to request GPS data from the FONA. Then, once the coordinates are saved, the connection resumes in order to transmit the packet.

On inception of the node into the network, the node does not have a UUID by which to identify itself. The server expects a node's UUID in the TCP packet in order to know to which node this sensor data belongs. The node could create the its own UUID locally and assign itself the ID, but if two nodes assign themselves the same ID, the server would not be able to distinguish them. Therefore, on the node's first TCP transmission, the UUID field will be null. Thus, when the server receives sensor data and finds the UUID is null, the server can create a UUID, query the database to ensure it has not been used, and respond to the node with the new UUID. The node will save this response as its personal identifier for all future TCP packet transmissions.

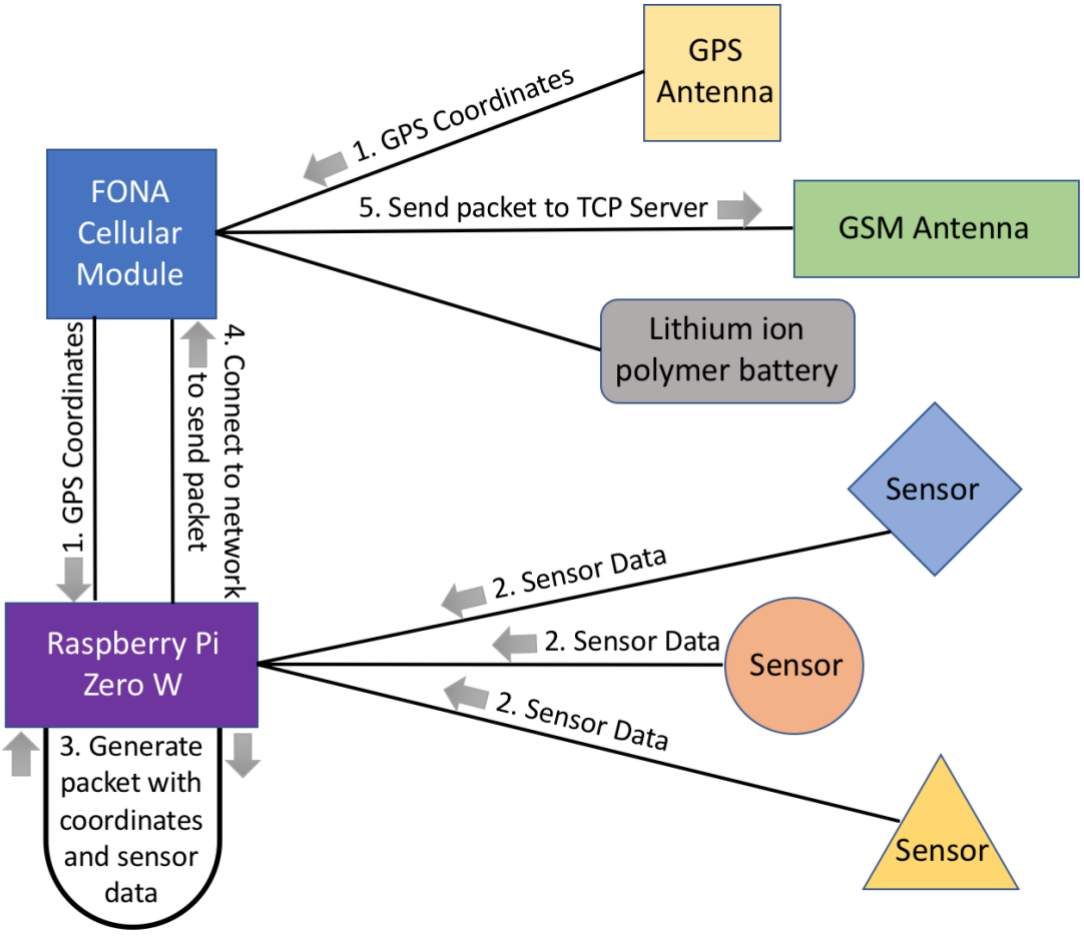

Figure 3: Node Data Flow

Case

Designing the exterior case which houses the node hardware components posed a distinct dilemma; how can the hardware be protected from weather and human interference while simultaneously exposing sensors to ambient air? The Raspberry Pi Zero, the FONA module with lipoly battery, GPS and cellular antennas, the regulator, and the lead acid battery must be shielded while the solar panels have direct exposure to sunlight. The air sensors must have a combination of exposure and protection as they cannot get wet from precipitation or condensation, but they must contact outdoor air. For proof of concept and keeping production costs low, two five-gallon buckets were purchased to nest within one another. Five-gallon buckets were chosen for their wide availability, small cost, and minimal weight for strength. Furthermore, the plastic used for buckets is thin enough to allow cellular and satellite connection from the FONA through the material.

The shielded hardware components will be places in one of the buckets covered with its lid and closed with an impermeable sealant. A port through the top of the bucket would allow the wires from the regulator to reach the solar panel outside the bucket. A port through the bottom of the bucket would allow the wires from the Pi to reach the air sensors below the sealed bucket. To protect these sensors, the sealed bucket will be partially placed in another bucket. A thick sealant placed around the inner sealed bucket will lock the outer bucket from the exterior climate. Then, the sides and bottom of the outer bucket with be punctured with many small holes below the overlap with the inner bucket. This should allow air flow in the space between the buckets for the sensors, yet still be securely contained. Ensuring there are enough holes in the bucket is critical for several reasons. The air in the space between the buckets should represents the ambient air accurately for temperature, humidity, and pollutant

levels. Additionally, any condensation that collects in between the buckets should be able to escape to prevent the sensors from accumulating moisture. Therefore, the size and number of holes in the bottom bucket must be optimized to compromise air flow with hardware protection. All the ports must be sealed with a watertight caulk in order to properly contain the hardware components from outdoor interference.

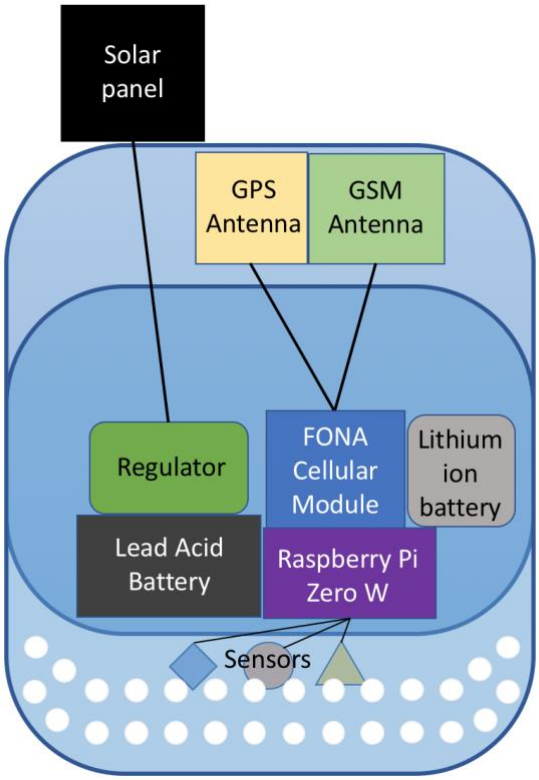

Figure 4: Node Case Diagram

3.4 Server

The construction of the server can be separated into three segments: the TCP server, the Cassandra database, and the Spring server. The TCP server is written in Python and is the access point between the nodes and the server. Its main responsibility is receiving sensor data and inserting it into the Cassandra database. The Cassandra database consists of several tables which hold the user, node, and sensor data. The Spring server is written in Java and

acts as a RESTful API that allows the web application to query the Cassandra database. The server is hosted a virtual machine on the University of Arkansas Nebula Cloud platform.

TCP server

The TCP server is a multithreaded Python server that runs on port 5005 of the EarthSense partition of the VM. It uses a socket connection to allow the streaming of TCP packets between the nodes and server. A loop continuously listens for a node to write to the socket address. If a node sends a packet, the server creates a thread to handle that packet. In the thread, the packet is decoded and parsed using a comma as a delimiter. The parsed data is save in an array. The packet is expected to contain the UUID which identifies the node, the timestamp to indicate when the readings were taken, and the value from each sensor. The server initiates a connection to the Cassandra database, then issues a command to insert the values in the data array into the sensor data table. The server completes its data processing by sending the node an acknowledgement of successful submission.

Cassandra database

The database schema consists of three tables: user, node, and sensor data. The user table contains an entry for each user registered from the web application. Users are people who registered on the web application primarily to control and modify their nodes in the network. Each entry in the user table has a user ID, permission ID, username, password, and email. The user ID is a UUID which uniquely identifies the user and is the primary key of the user table. The permission ID indicates whether a user is authenticated to modify a node. The

email is used to register the account. The username and password are used to log in to the user's profile.

There is one entry per node in the node table. Each node has a node ID, owner, name, latitude, longitude, city, state, country, and status. The node ID is a UUID which uniquely identifies each node and is the primary key of the node table. The owner is a UUID which uniquely identifies users and is a foreign key to the user table. The owner of a node is the only user who is authenticated to make modifications to node information. The name is a colloquial name for the node for simpler identification on the web application. Latitude and longitude are used to place markers on the web application map display, and the city, state, country, and status of the node are displayed on a marker's information window.

The sensor data table contains all of the readings from all nodes. This table is where the TCP server inserts data received from the nodes and is the reason why Cassandra was chosen for its scalability. As many nodes report regular readings over time, this table will get very large. Each entry in the table has a node ID, timestamp, and as many sensor reading entries as sensors on the node. The node ID is the UUID corresponding to the node from which the readings came and is foreign key to the node ID in the node table. The timestamp indicates when the readings were taken. Each of the sensor readings are in decimal value. The node ID and timestamp uniquely identify an entry in the sensor data table.

Spring server

The Spring server is a Java implementation of a RESTful API that the web application uses to access the Cassandra database. The Spring framework uses controllers, models, and repositories in order to perform HTTP requests on the database. Repositories are interfaces

which extend the Cassandra Repository for each database table and contain methods which query the database. Models define the schema for each table to store and modify tuples from the database. Controllers define the endpoints by which the web application links to execute the HTTP requests from the web application. A separate controller handles the requests for each table in the database, so there are three REST controllers on the Spring server. Methods in the controller utilize models to map tuples to a known format in order to use repository methods.

3.5 Web Application

The web application is Angular based to enhance the standard HTML-CSS-JavaScript design. Its purpose is to provide a public view of the node and sensor data that is easy-to-use and delivers useful interpretations of the data. The application consists of a home page, data page, map page, node management page, login and sign up page. The web application opens to a home page which outlines the basic design and purpose of the project. The data page and the map page are both platforms by which to exhibit the data. The login and sign up pages authenticate users. The node management page allows users to modify their account information and node information for the nodes they own.

The data page is meant to display the sensor data numerically. Users can inspect the value of a reading from any sensor from any node and see the change of gas concentrations over time. The trends of different sensors can be compared to each other to distinguish correlation. The map page allows users to see where nodes are located and basic information about the node. This page can also use a heatmap style map to show relative concentration of a pollutant in an area. The web application uses HTTP Client to communicate to the Cassandra

database with the RESTful API on the server. Requests to the API return the data to populate the data and map page displays.

4.0 Results

Based on the updated design, one node was created as a proof of concept for the hardware and to begin the data path flow for the server and web application. This node was not optimized for power as many of the major components were available for free or at a reduced cost for this initial system design and implementation. The solar panel is a 12V nominal voltage, 16.8V peak voltage panel which charges the 12V 8Ah lead acid battery. These power the Raspberry Pi Zero W and FONA module each with their respective attachments. While the original design called for the employment of the Sensly HAT for air quality testing, an ordering and shipping mishap hindered its arrival and addition to the system. Therefore, several smaller Adafruit sensors were added to test and demonstrate the capability of the system to take data readings from sensors, compile into a data packet, and send to the TCP server. A cron job is employed to automate the data retrieval and transmission procedure on a regular schedule. GPS coordinates can also be read from the GPS antenna attached to the FONA and sent to the TCP server. During various points of operation, the voltage and current draw has been measured to compare against earlier approximations in the design process. Voltage levels remained steady around the 5.00 to 5.05 range. When idling with all peripherals attached, the Pi only pulled from 80 to 90 mA. During higher strain when collecting data from sensors and sending packets, current draw was approximately 180 to 190 mA. With additional testing, this could mean an extension of time that the lead acid battery could supply the system during non-ideal sun hours as this current draw is lower than earlier estimations.

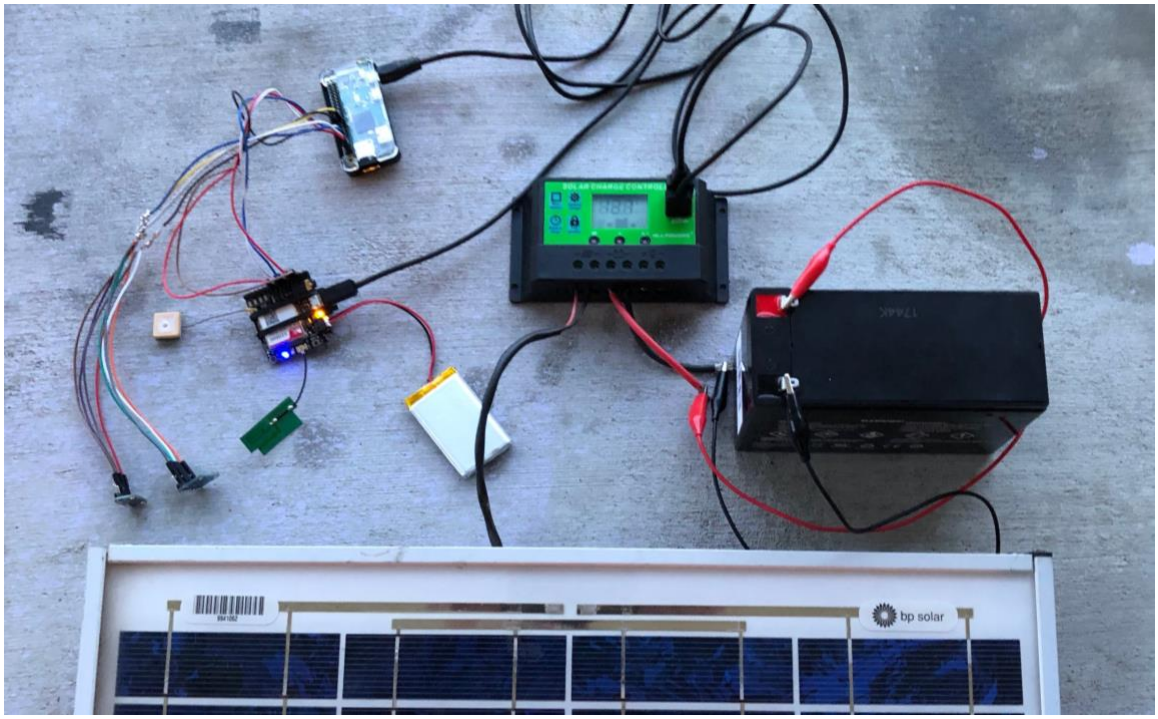

Figure 5: Node implementation

The central server is equipped with a TCP server, a Cassandra database, and a Spring server as per the design rework from the previous summer. The TCP server written in Python uses a socket to listen for a data packet from the node and insert the data into the sensor data table in the Cassandra database. Currently, the packet must fit a rigid, specified format in order for the TCP server to parse the packet and complete the insert. The TCP server uses the ThreadedServer Python class so that each new client gets its own thread and does not block the main thread by creating a new socket per client.

The Spring server written in Java implements a REST architecture style to service HTTP standard requests from the web application. The three REST controllers define the Spring server endpoints for the HTTP requests and use the Cassandra repository functions to perform requests on the database. The user controller has five endpoints: a GET to return all users, a POST to return a user from a username and password, a POST to create and return a

user from a username, password, and email, a PUT to modify existing information about a user from a user object, and a DELETE to delete a user from a user ID. The node controller also has five endpoints: a GET to return all nodes, a GET to return a single node from its node ID, a POST to create a node from a node object, a PUT to modify a node from a node object, and a DELETE to delete a node from a node ID. The sensor data controller has two endpoints: a GET to return all sensor data entries and a GET to return sensor data entries from a particular node given a node ID. The controllers use mapping annotations to define the endpoint for the HTTP request from the web application.

The web application produces data and map pages that read the node and sensor data tables from the Cassandra database and use the tuples to generate graphs and maps. The data page can generate a line graph over time for many nodes with filters for sensor type. The map page can produce a node location map which shows position of the node and an information window containing the most recent readings from that node. A user can also log in and log out, but full node management is still in progress.

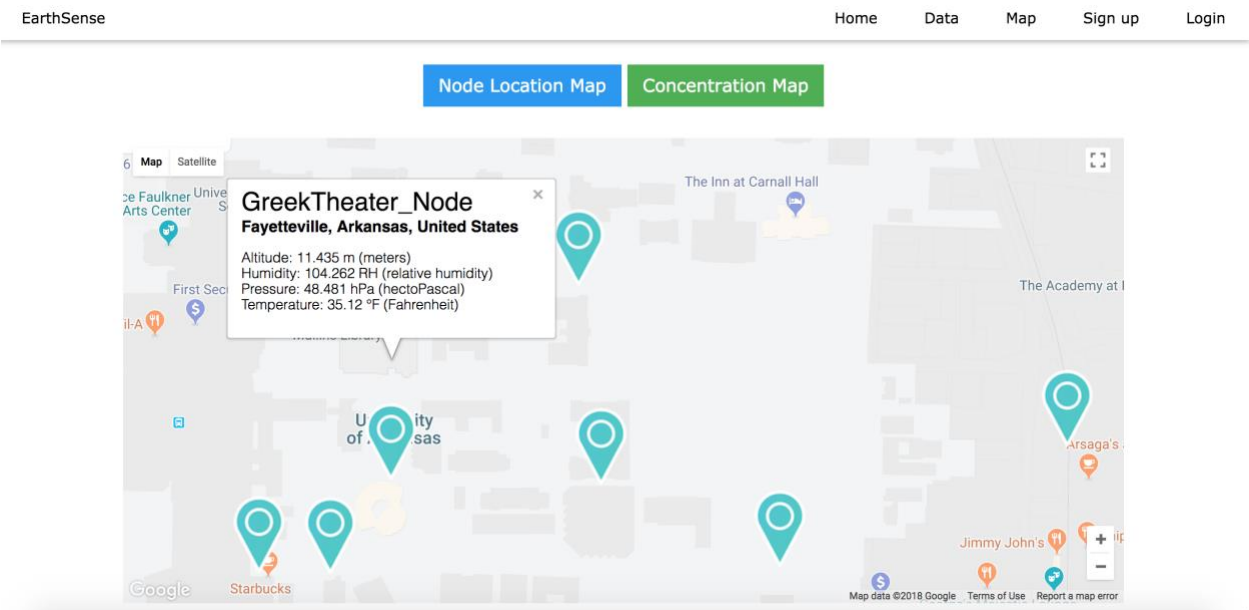

Figure 6: Node location map example

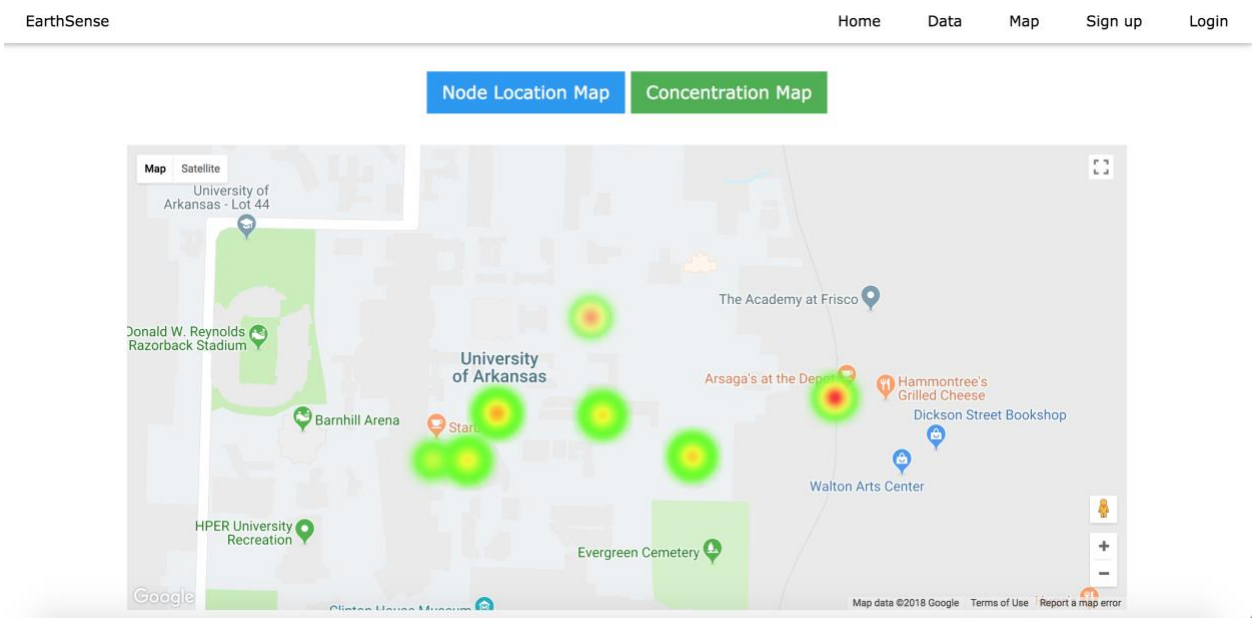

Figure 7: Concentration map example

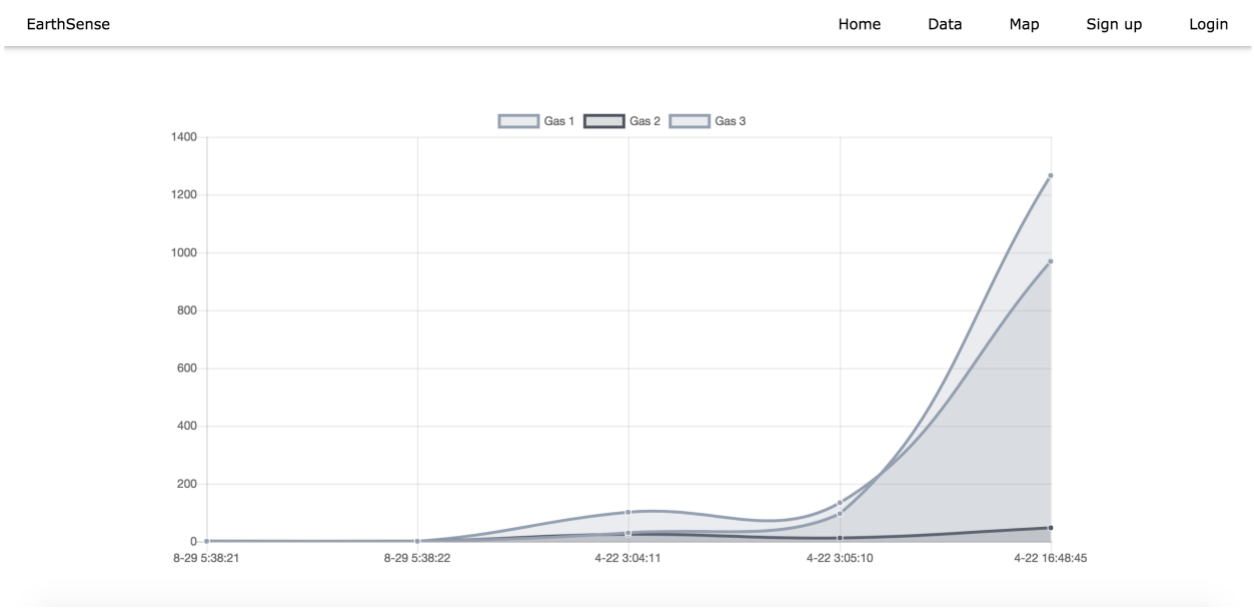

Figure 8: Data page example

5.0 Future Work

5.1 Implementation Improvements

While the current implementation of the EarthSense system meets the basic goals of the project, there are many improvements that can be made moving forward. First, the node hardware needs more extensive long-term system testing to better understand and optimize power consumption. The current draw and average number of peak sun hours over time should be measured in order to minimize cost and size of the components. The full system from node to web application would benefit from additional security measures. Data encryption from node to server and server to web server should be added. Data stored in the database should also be encrypted. A major hindrance to the current server implementation is the location of host behind the university firewall on the Nebula Cloud platform. Currently, in order to retrieve data from the database, the web server must be hosted behind the firewall on the cloud as well, and the user must either be connected to University of Arkansas Wi-Fi or on a University of Arkansas VPN to access both the server and web server. To bring this service public, this must change. The system was built for functionality for the majority of the lifetime of the node but lacks an automatic procedure for node inception to the nodal network. When a node is instantiated into the network, the user with authority over that node must be set manually as well as the city, state, and country of which the node is located. The node must also be given its unique ID by which to identify itself for future sensor readings. This could be solved in the future by using authentication codes given to node owners to authenticate a new node in the server for the owner's specific ID.

5.2 Design Improvements

EarthSense is a good foundation for implementing a dense, low-cost distributed network of air quality sensors, but there are some improvements that could be made to this design to better achieve the goals set out for this project. One such improvement could be to use a communication method like LoRa to transmit sensor data from the nodes to the server. LoRa is a type of Low Power Wide Area Network (LPWAN) specification known for very low power and long-distance data transfer using radio and is a common inclusion in IoT devices[28]. LoRa meets the objective of allowing remote location for nodes as a single LoRa base station can grant coverage to entire cities [28]. Additionally, LoRa modules have extremely low power consumption. Some modules, like ones from the manufacturer Telit, have batteries that can last for years [29]. Using LoRa could also lower cost per node by eliminating the cost of the lipoly battery, SIM card, and subscription to a 2G network provider. Adafruit's LoRa modules with radio antennas are a comparable price to the cellular module used currently. The tradeoff would be maximum packet size and speed of transfer as LoRa data packets must be small and are relatively slow.

Another possible design improvement would be to incorporate both the current cellular network nodes and LoRa nodes together to increase the sensor density. At a lower price point, LoRa nodes could be created and implemented in mass to collect sensor data. Then, multiple LoRa nodes in one area could send their data to a single cellular network node which could aggregate the data and send larger packets to the server. Since LoRa nodes would likely be cheaper to produce and take less power to operate, they could be used as the bulk of the network. Then, fewer of the more expensive cellular nodes could be made as hubs for the LoRa nodes.

In order to bring positive attention to the node, the physical appearance of the node could be improved. While the case serves its purpose, it is not an aesthetically pleasing design nor is it inconspicuous. If these nodes are places around cities, parks, and homes, most people would rather them go unnoticed or add some visual value. Reducing the size of the node as much as possible would add to their subtlety. If the design goal was reformulated to encourage user interaction, some hardware additions could be considered. LED indicators on the node in a green, yellow, and red configuration could generally indicate the quality of the air sensed by that node at the time. A more radical implementation of this objective would be to add interactive utilities to the node. If the node was collecting excess solar power, phone chargers or other desirable conveniences could be added. This would bring awareness to the node, and, in turn, bring awareness to the harms of air pollution and the objective of EarthSense.

5.3 Applications of System

The function of projects like EarthSense influence more than just the computing technology community. Gathering, storing, and presenting air quality data is a necessary step for governments, industries, and research companies. Governments create and evaluate air quality standards based on the data that air monitoring systems collect. Testing the air around industrial and manufacturing facilities determines whether current and additional activity is acceptable for quality standards [30]. Researchers utilize this data to understand the contributing factors to worsening air pollution levels and the effects that current and future emissions have on human health and the environment. Understanding and publicizing this information is critical for decisions in policy making for governments and businesses as well

as influencing the consumption habits of the individual. Solutions to eliminate pollution at its present rate and reverse the existing damage cannot be developed without the collection of the data.

6.0 Personal Contributions and Acknowledgements

This project could not have been executed to this degree without my team of computer science and computer engineering students at the University of Arkansas. Without the aid from Russell Lowry, Brok Stafford, and Keaten Stokke, EarthSense would not have been the complete project it became. My personal contributions towards the final product were heavily based in the research and optimization of the node hardware and server and the design and implementation of the home and map pages on the web application. I researched the required materials for the construction of the node and how to optimize the high-level design for the most efficient system. For the node software, I contributed to the scripts of the automated cron job on the Raspberry Pi. I designed, in part, the implementation of the Spring server as well as created and modified some endpoints for the REST API. I also established the framework for the web application, created the services for retrieval of node and sensor data from the server for use on the web application, and designed the home and map pages.

7.0 References

[1] Barrett, Kim E., and William Francis. Ganong. *Ganong's Review of Medical Physiology*. 24th ed., McGraw-Hill, 2013.

[2] "How Much Oxygen Does a Person Consume in a Day? | Air Quality." Sharecare, Discovery Health, www.sharecare.com/health/air-quality/oxygen-person-consume-a-day.

[3] "FINA FACILITIES RULES." Fédération Internationale De Natation, 28 Oct. 2015.

[4] "Air Chemistry Laboratory." ADEQ, Arkansas Department of Environmental Quality, 2018, www.adeq.state.ar.us/techsvs/air_chem_lab/.

[5] Ghorani-Azam, Adel, Bamdad Riahi-Zanjani, and Mahdi Balali-Mood. "Effects of Air Pollution on Human Health and Practical Measures for Prevention in Iran." Journal of Research in Medical Sciences : The Official Journal of Isfahan University of Medical Sciences 21 (2016): 65. PMC. Web. 27 Apr. 2018.

[6] World Health Organization. (2016). Ambient air pollution: a global assessment of exposure and burden of disease. World Health Organization. http://www.who.int/iris/handle/10665/250141

[7] "Health & Environmental Effects of Air Pollution ." Commonwealth of Massachusetts, Executive Office of Energy & Environmental Affairs, Department of Environmental Protection, 2018.

[8] "Effects of Acid Rain." EPA, Environmental Protection Agency, 1 June 2017, www.epa.gov/acidrain/effects-acid-rain.

[9] Choi, Sukwon et al. "Micro Sensor Node for Air Pollutant Monitoring: Hardware and Software Issues." Sensors (Basel, Switzerland) 9.10 (2009): 7970–7987. PMC. Web. 27 Apr. 2018.

[10] "About AQMesh – AQMesh." AQMesh, www.aqmesh.com/about-aqmesh/.

[11] Polidori, Andrea. "Air Quality Sensor Performance Evaluation Center." 27 Mar. 2015.

[12] "Environmental Monitoring Systems and Instruments for Outdoor Air Quality ." Aeroqual, Aeroqual Limited, 2018, www.aeroqual.com/outdoor-air-quality.

[13] "Aeroqual Series 500 Monitor (S-500) with Sensor Head." Gas Sensors Aeroqual Series 500 Monitor with Sensor Head - Gas-Sensing.com, www.gas-sensing.com/aeroqual-series-500.html.

[14] "Air Quality Monitoring." PurpleAir, 2017, www.purpleair.com/.

[15] "Frequently Asked Questions." MQTT RSS, 2014, mqtt.org/faq.

[16] "Eclipse Mosquitto." Eclipse Mosquitto, 8 Jan. 2018, mosquitto.org/.

[17] "Manage Massive Amounts of Data, Fast, without Losing Sleep." Apache Cassandra, 2016, cassandra.apache.org/.

[18] Ionicframework. "Build Amazing Native Apps and Progressive Web Apps with Ionic Framework and Angular." Ionic Framework, 2018, ionicframework.com/framework.

[19] "Sensly." Altitude Technology, Altitude Tech LTD & IoTBit LTD, 2016, altitude.tech/sensly/#prettyPhoto.

[20] "SIM800 Series_TCPIP_Application." Shanghai SIMCom Wireless Solutions Ltd., 10 Dec. 2013.

[21] "Angular Docs." Angular Docs, 2018, angular.io/.

[22] Masters, Jeff. "PurpleAir's $250 Air Pollution Monitor Gives Government Equipment Run for Money." Weather Underground, 16 July 2017, www.wunderground.com/cat6/purple-airs-250-air-pollution-monitor-gives-government-equipment-run-money.

[23] Adafruit Industries. "Raspberry Pi." Adafruit, www.adafruit.com/category/105.

[24] “Ting Rates - Only Pay for What You Use. Use Less? Pay Less.” Ting, 2018, ting.com/rates.
[25] “BU-303: Confusion with Voltages.” Battery Voltage Information – Battery University, 9 May 2017, batteryuniversity.com/learn/article/confusion_with_voltages.
[26] “BU-403: Charging Lead Acid.” Charging Information For Lead Acid Batteries – Battery University, 4 Apr. 2017, batteryuniversity.com/learn/article/charging_the_lead_acid_battery.
[27] “Raspberry Pi Resources.” Raspberry Pi Projects, www.raspberry-projects.com/pi/pi-hardware/raspberry-pi-zero/raspberry-pi-zero-hardware-general-specifications.
[28] “About LoRaWAN™.” LoRa Alliance™, 2018, www.lora-alliance.org/what-is-lora.
[29] “Telit_RE866_Datasheet.” Telit, 2017.
[30] Fort Air Partnership, and Government of Alberta. “How Air Quality Monitoring Data Is Used.” Fort Air Partnership, 2016.
[31] “How Much Power Does Pi Zero W Use?” RasPi.TV, 1 Mar. 2017, raspi.tv/2017/how-much-power-does-pi-zero-w-use.