Aether Sensor Network

Group 41

Ian Wallace
Computer Engineering

Paul Wood
Computer Engineering

Parke Benjamin
Electrical Engineering

Randy Alvarez
Electrical Engineering

April 26, 2022

Contents

List of Tables

List of Figures

1 Executive Summary

The goal of our project was to design a sensor node that can detect and measure the quantities of various pollutants within its immediate surrounding and send this data to a server, where it can be displayed to the user over a web interface. The sensor data is transmitted via a LoRa transceiver on the sensor node over a LoRaWAN network, consisting of a gateway and an server. In addition, the electronics of the sensor node are encased in a 3D-printed shell that will both protect the electronics and provide a method of securely mounting the sensor node to various surfaces.

The sensor nodes are the primary design focus of our project. We designed them with cost in mind, so that it will be cost-effective to build multiple sensor nodes. Having multiple sensor nodes will provide greater coverage over a given area and will provide be more useful to the user. Similarly, another objective for the sensor node design was the overall size of the device to make it easy to setup. We used the U.S. Environmental Protection Agency's (EPA) Air Quality Index (AQI) as the basis for selecting the sensors in our design. The pollutants that go into calculating the AQI are: ozone (ø3), particulate matter (PM2.5 & PM10), nitrogen oxides (NO_x), sodium oxides (SO_x), and carbon monoxide (CO). However, as defined by the EPA NowCast AQI, the two most prominent pollutants are ozone and particulate matter, and are what we focused on in our design due to cost constraints.

As part of the LoRaWAN specification, a LoRaWAN gateway is required to route data from a LoRa transceiver to a server. The gateway will not be included as a part of the design aspect of our project, however, one is needed to have a functioning LoRa device. As such, we used an off-the-shelf The Things Stack indoor gateway for development and a Raspberry Pi with a RAK2445 LoRaWAN Pi Hat inside of a project box for outdoor testing.

The network server and web interface is a critical part of our design as it is how the user will acquire information about the AQI. We aimed to allow the user to view real-time and historical data feeds for the various pollutants and AQI values. To facilitate the back-end services of the Aether Node, we decided to use Supabase as our database and API, AWS to host our static website, and The Things Stack (TTS) as our LoRaWAN network and join server. Supabase is a managed PostgreSQL database with additional services that allow the database to be accessed through a REST API and facilitate real-time data-feed updates. The entire back-end logic was created inside the PostgreSQL database.

LoRaWAN packets are forwarded from the gateway to TTS which are then again further forwarded to Supabase through REST POST requests. Once processed by the back-end, the readings are sent to listening clients using websockets for real-time updates. The web interface allows users to select the data they would like to view on the interactive map along with historical values. Users can also view and modify devices they own, and view real-time and historical data.

2 Project Description

2.1 Motivation

Industrialization and climate-change go hand in hand in affecting air quality. Industrialization worsens climate change and air quality, and climate change further worsens air quality through natural disasters such as wildfires, acid rain, or sand storms. It is estimated that 7 million people die from air pollution globally a year [3]. Wildfires pose a significant risk for the general population, and especially those at risk for respiratory infections and people over the age of 65. There is also a disproportionate increase in air pollution for those in poorer communities where often times they are close to industrial areas [2]. By providing a low-cost way of measuring the overall air quality, people can be more educated about the air quality in their communities, and provide them with the data to back up their complaints to their local municipalities. For developing countries, they might be less focused on the environment due to cost constraints, but by providing a cheap option for measuring air pollution, they can potentially make more educated decisions about where to make improvements. A low cost air quality sensor might also be of interest for those that are health conscience, or are already at risk for respiratory disease, as they can make decisions based on the current air quality. People living in cities, or other highly industrialized areas, are also subject to increased air pollution, mostly from motor vehicles (what create smog and haze), and from factories. Even short term exposure can cause adverse health affects [5]. However, the full extent of long term exposure to these pollutants is inconclusive. What is conclusive is that air pollution is the cause for millions of deaths world wide each year.

Air quality sensors are very valuable for people living in areas that are high risk for wildfires, such as California. Smoke from wildfires can be deadly, if not cause serious health effects depending on the smoke concentration and exposure time. Based on the air quality, using either the AQI or the AirNow index, people might need to stay indoors. Another potential use case of an air quality sensor is the ability to detect wildfire by deploying them to wildfire-prone areas. By using LoRa, these sensors can provide a wide range of coverage centered around a LoRa gateway.

2.2 Goals and Objectives

- The Aether Node should be low cost.
- The Aether Node should be lightweight.
- The Aether Node should be able to read data from the sensors and transmit that data to a LoRa gateway.
- The Aether Node should be able to communicate over a large enough distance in order to make it feasible to cover large enough area with nodes.

- The Aether Node should require little to no maintenance.
- The Aether Node should be easy to install.

2.3 Requirements Specifications

In this section, we will describe in detail all of the requirements for our design. However, before going through the entire list, we will first explore the three demonstrable requirements. These three requirements are the ones that we demonstrated after our design was fully implemented. These three requirements can be found highlighted in blue in Table 23.

Demonstrable Requirement 1 The first of our demonstrable requirements is to have the Aether Node calculate the AQI within ±25% of the AQI value reported by the EPA. We have this as a requirement to ensure that the Aether Node is accurately reporting air quality.

Demonstrable Requirement 2 The second of our demonstrable requirements is to have an average power draw of less than 100 mW. This requirement was chosen because it is critical that the Aether Node is a low power device. By choosing the average power draw, we are considering the fact that the Aether Node will be in a sleep mode for the majority of the time, but when the SPS30 (the particulate matter sensor) is on or it is transmitting LoRa packets, then it will draw significantly more current.

Demonstrable Requirement 3 The third and final demonstrable requirement is to ensure that the Aether Node is able to automatically take a sensor reading from each sensor every fifteen minutes. This requirement is fundamental to the basic operation of the Aether Node. The Aether Node must be able to consistantly and periodic take sensor readings. These requirements are shown in Table 23.

In terms of the remaining requirements, the first of these is that the Aether Node battery shall be able to be fully charged through a USB connection at 500 mA in six hours. This requirement ensures that if an Aether Node needs to be taken down to be serviced, it can quickly recharge the battery while doing so. The fifth requirement for our design is that the solar panel of the Aether Node shall maintain the battery at 80% capacity or more during normal operation. The goal of this requirement is to ensure that the Aether Node is able to, at a minimum, not lose charge while connected to the solar. The sixth requirement is that the device shall have an average current draw of less than 5 mA while operating in sleep mode. The goal of this requirement is to ensure that the Aether Node is not drawing much current when in sleep mode, since this is the state that the Aether Node will operating in for the majority of the time. Finally, the seventh requirement is that the device shall have an average current draw of less than 250 mA during normal operation. This value was chosen to account for the possibility of the Aether Node taking readings for the high-current

draw SPS30 and also transmitting LoRa packets. Those are the two most power expensive operations performed by the Aether Node. These requirements are shown in Table 23.

Table 1: List of Requirements

No.	Requirement	Unit
1	The device shall calculate the AQI within a specified percentage of the EPA's reported value.	±25%
2	The device shall have an average power draw of less than the specified number of watts.	100 mW
3	The device shall be able to automatically take a sensor reading from each sensor at a specified time interval.	15 minutes
4	The battery shall be able to be fully charged through a USB connection at 500 mA in 6 hours.	6 hours
5	The solar panel shall maintain the battery at 80% capacity or more during normal operation.	80%
6	The device shall have an average current draw of less than 5 mA while operating in sleep mode.	5 mA
7	The device shall have an average current draw of less than 250 mA during normal operation.	250 mA

Constraint
We only had approximately six months to design and implement our project.
We self-funded our project and were therefore limited by money.
Due to the combination of both money and time, we had to limit the complexity of our project.

Table 2: The constraints for our project.

2.4 House of Quality

In Figure 1, we provide a comparison between our marketing parameters and our engineering requirements. The marketing parameters are qualitative in nature and are chosen based on the qualities that we imagine customers would be looking for in this product. Accurate means that the sensors produce a reading that reflects the correct concentration of gases in the air around the node. It could also refer to having enough nodes to cover an area to provide a high resolution of sensor data. Low maintenance and reliable are somewhat related parameters, however, low maintenance refers to the ability of the node to operate for a long enough period of time without needing

to replace various components and reliable refers to the ability of the node to survive the outdoor elements and continue to provide quality sensor data over time. Real time means that the node should be transmitting and reading data from its sensors as often as possible. Easy to install means that the nodes should be able to placed in the desired location with minimal effort. Ideally, the nodes would be able to be placed in wide variety of situations and locations with relative ease.

In contrast to the marketing parameters, the engineering requirements are quantitative in nature. Battery life is key since it determines how long the nodes can be deployed without having to have batteries replaced or recharged. Node-to-gateway distance refers to how far away a node can be away from a gateway and is a critical requirement because it directly impacts how large of and area you can cover with nodes. The number of nodes per gateway is also important because we would like to minimize the number of gateways required for any given number of nodes in the network. Sensor accuracy refers specifically to how far the true measurement of a given gas is from the value outputted by the sensor. It is given as a ±percentage. Finally, the nodes should of course be as small, light, and cost as little as possible.

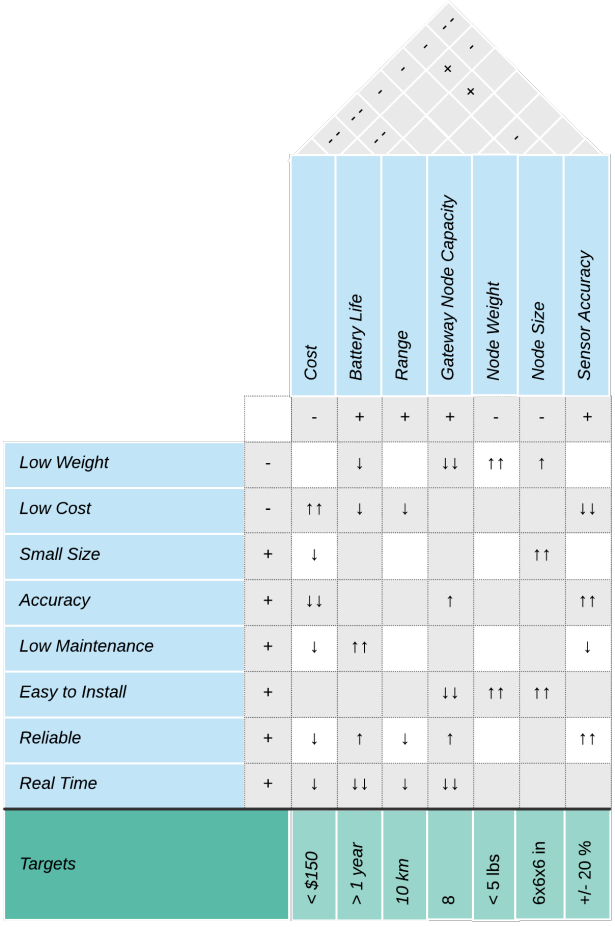


Figure 1: The house of quality diagram

Almost all of the engineering requirements negatively impact one or more of the other engineering requirements. Every requirement has a negative impact on the cost of the nodes and gateways. To increase the battery life, either more engineering work will be needed to optimize the system (and potentially increase the BOM) or simply increase the battery capacity. Both of these options increase the cost. Likewise, the same is true for the range of the nodes. The range of the nodes involves finding some way to increase the link-budget. As for the node's weight and size, these correlate to increased raw material costs and/or increased manufacturing costs, especially for the node size. Lastly, sensor accuracy affects the cost because of the engineering, material costs, and manufacturing needed to create more accurate sensors. Battery life also heavily negatively affects range, because of the need for active RF circuitry, namely amplification, when transmitting data. The size of the nodes helps battery life since it allows for a battery with a higher capacity. It also helps the range of the nodes by allowing for a larger antenna, which helps with increasing the receive sensitivity. Lastly, the weight and size of the nodes negatively impact each other since, because usually the larger the node is the more it is going to weigh.

2.5 Block Diagrams

In this section, we show the block diagrams for different parts of our design. Our block diagrams also indicate who is going to be responsible for each part of the design. The Aether Node is our primary design focus for this project. The block diagram for the Aether Node shows how the various sensors will interface with the microcontroller as well the system is powered via a battery and a solar panel. This block diagram can be found in Figure 2. In addition, a more detailed description of each block in the diagram can found in Table 3. We also include a diagram showing a high-level flow of the Aether Node firmware, this can be found in Figure 3.

2.5.1 Overall Block Diagram

The Aether Sensor Network consists of two main components: the device which collects air quality data, known as the Aether Node, and a web server that hosts and displays that data. The Aether Node uses multiple gas sensors to collect the data used in calculating air quality. This collected data is transmitted by the Aether Node over LoRaWAN, an upcoming technology that has been growing in popularity recently for use in internet of things (IoT) applications, to a server. The Aether Node is implemented on a four-layer PCB and housed in a 3D printed enclosure. The computation that is performed by the Aether Node is done by a Seeed LoRa-E5 LoRaWAN module. This module packages an ST STM32WLE5JC, which is an SoC combing an ARM M4 processor and a LoRa chip. Figure 2 shows how the LoRa-E5 module processes data collected from the three primary senors of the Aether Node, which include the two main gas sensors, a Renesas ZMOD4510 and a Bosch BME688, and a Sensirion SPS30 particulate matter sensor. The LoRa-E5 module transmit the collected sensor data to the second major component of the Aether Sensor Network, the web server.

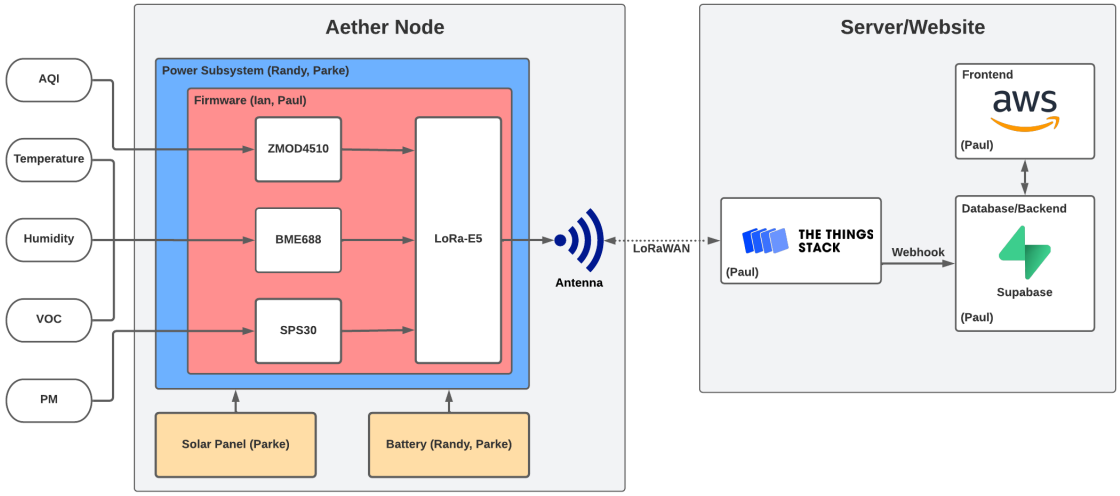


Figure 2: The overall block diagram for the Aether Sensor Network.

As seen in Figure 2, the web server used by Aether is hosted on Amazon Web Services (AWS) and uses Supabase for the database and API. Finally, the results are results are displayed on a web page using the popular JavaScript framework, React. Through this web page, users can see map with the data collected from all Aether Nodes that have been registered with the server overlaid on a Mapbox map. Users are able to add alerts for different events, such as when the AQI exceeds a user-defined threshold. These alerts will be sent to the user through email.

Block Name	Description
Battery	The battery provides power to the Aether Node through the power subsystem. It is charged by the solar panel.
Solar Panel	The solar panel charges the battery.
Power Subsystem	The power subsystem takes in and regulates power from the solar panel, charging the battery. The power subsystem outputs power onto 5 V and 3.3 V rails for the sensors and LoRa-E5.
LoRa-E5	The LoRa-E5 runs the device firmware. It controls the sensors to take readings and sends the collected data through it's LoRa radio.
ZMOD4510	The ZMOD4510 receives temperature and humidity data from the BME688 and combines that data with its own sensor readings to calculate the AQI. It recieves 3.3 V from the power subsystem.
BME688	The BME688 reads temperature, humidity, and VOC data. This is sent to the LoRa-E5. It receives 3.3 V from the power subsystem.
SPS30	The SPS30 reads particulate matter data. This is sent to the LoRa-E5. It receives 5 V from the power subsystem.
The Things Stack	The Things Stack forwards packets from the LoRaWAN gateway to the web server backend.
Supabase	Supabase is used for the website's database and API.
Amazon Web Services	AWS is used to host the website frontend.

Table 3: A description of each block in the overall block diagram.

2.5.2 Firmware Block Diagram

Additionally, we created a block diagram to describe, at a high level, how we plan for the embedded software running on our sensor node to function. We have split the software into three major sections: the application logic, the networking stack, and the device drivers. The application logic will manage node functions, such as when the device will sleep and wake, how the device will manage its power sources, and how the device will handle measuring sensor input data. The networking stack will sending the data that was collected from the sensors to the internet over LoRaWAN. Finally, the device drivers section will handle all of the raw signals coming in from all of the attached sensors and hardware modules, such as the physical LoRa module, the gas sensors, and the particulate matter sensor.

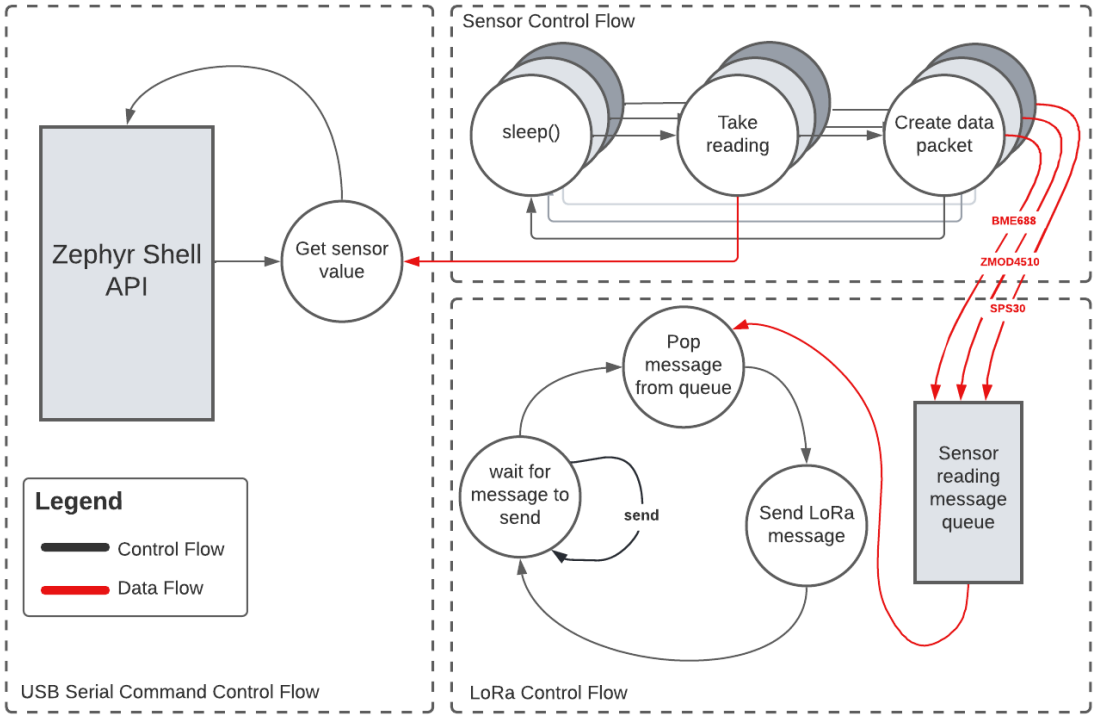


Figure 3: The block diagram for the Aether Node firmware.

Block Name	Description
Sensor Control Flow	The sensor control loop is responsible for collecting data from the three sensors used by the system (BME688, ZMOD4510, and SPS30).
USB Serial Command Control Flow	This is the control loop that handles accepting user commands from the UART shell.
LoRa Control Flow	This is the control loop that handles sending collected sensor readings to our server over LoRaWAN.

Table 4: A description of each block in the Aether Node firmware block diagram.

3 Background

In this section, we provide information pertaining to the background knowledge that we used to inform our design decisions. The information in this section played an important role throughout the design process, especially in the selection of the sensors for the nodes. We discuss information regarding the different types of air pollution and how government agencies like the EPA classify this information to create useful useful bulletins like the Air Quality Index (AQI).

3.1 Effects of Air Pollution

Pollution is the introduction of substances into the environment that are harmful to humans and other living organisms [5]. Pollutants can be solids, liquids, or gases. Over the years, human activities have had an adverse effect on the environment by polluting the air we breath, the water we drink, and food that we eat. Climate change is also increasingly creating more air pollution from wildfires and other chemical reactions involving greenhouse gases. Inhaling smoke, while potentially short term, can lead to various health effects. However, air pollutants are more likely to affect those with predispositions, lower economical statuses, and those living in urban areas [2]. Human created air pollution accounts for approximately nine million deaths per year [5]. Short term exposure to air pollutants is correlated to Chronic Obstructive Pulmonary Disease (a chronic inflammation lung disease), cough, shortness of breath, wheezing, asthma, other respiratory diseases, and increased hospitalization rates. Long term exposure is correlated to chronic asthma, pulmonary insufficiency (a condition where the pulmonary valve in your heart allows backflow), cardiovascular diseases, and cardiovascular mortality. Newborns and children may also develop potentially life-long cardiovascular, mental, and perinatal disorders.

There are four main sources of air pollution: major sources, indoor sources, mobile sources, and natural sources [5]. *Major sources* are sources of pollutants that a created from factories, power stations, refineries, and other industrial plants. *Indoor sources* are pollutants that arise from the use of cleaning chemicals, appliances, and other gas emissions that stay contained inside of industrial buildings. *Mobile sources* include anything relating to infrastructure, such as cars, trains, and airplanes. Lastly, *natural sources* arise from naturally occurring events, which may have to exacerbated due to climate change, such as wildfires or dust storms. Air pollution can also be absorbed into the ground soil, or cause acid rain.

3.1.1 Types of Air Pollution

The focus of this project is to be able to detect air pollutants defined by the U.S. Environmental Protection Agency (EPA) as being harmful and required to calculate the Air Quality Index (AQI) to help people identify dangerous air pollution levels. For communities and cities that suffer from poor air quality, having more data to help them make changes to their environment, such moving, applying pressure

to those regulating pollutant producers, or getting alerts of dangerous events/gas-accumulations, such as wildfires. The AQI is further discussed in a later section, but it defines a measurable, national standard to calculate the overall air quality based on five main pollutants: ground level ozone O_3, particulate matter (PM2.5 and PM10), carbon monoxide CO, sulfur dioxide SO_2, and nitrogen dioxide NO_2 [6]. Short-term exposure is defined as any exposure continuous over and up-to eight hours. While there is currently lots of data for short-term exposure to these pollutants, it is still inconclusive about the full extents of health risks due to long-term exposure.

Particulate Matter As defined by the EPA, is any tiny liquid or solid droplets that can be inhaled and cause serious health effects, 10 μm or less in. Coarse particles is defined as particles that are 10 μm or less. Fine particles are defined as particles being 2.5 μm or less. The half-lives of $PM_{2.5}$ and PM_{10} particles are relatively long, so they last longer in the atmosphere. This can cause particulate matter pollution to spread over long distances, such as smoke from some extreme wildfires in California reaching all the way on the east coast. Due to the small size of $PM_{2.5}$ particles, they can cause more serious health conditions [5]. Most particles in the atmosphere are created through chemical reactions with sulfur dioxide and nitrogen oxides, which are already pollutants created from power plants, factories, and combustible engines [23]. Particulate matter can be produced from dust, gas aerosols, and biologics (viruses, microorganisms and allergens) [5].

Ground Level Ozone According to the EPA, tropospheric (ground-level) ozone O_3 is a highly reactive gas, and is not naturally forming, unlike stratospheric ozone. Stratospheric ozone is created through the interaction of solar ultraviolet (UV) radiation with molecular oxygen (O_2). Tropospheric ozone, however, is formed through photochemical reactions between two classes of air pollutants: volatile organic compounds (VOCs) and nitrogen oxides (NO_x). Anyone that has ever been to a city on a hot day will noticed the smog and haze, which is the result of ozone formation. While some VOC and NO_x occur naturally, most tropospheric ozone is man-made. The main sources of VOCs are: chemical plants, gasoline pumps, oil-based paints (plastics, petroleum), or recycling facilities. And, as for NO_x, it is mainly found where ever there's combustion, such as in power plants, industrial furnaces, industrial boilers, and oil-based vehicles [22].

Carbon Monoxide Carbon monoxide CO is an odorless gas produced through incomplete combustion of fossil fuels. Carbon monoxide is mainly produced from motor vehicles, or anything that burns fossil fuels [5].

Sulfur Dioxide Sulfur dioxide SO_2 is an colorless gas with a "choking or suffocating odor". Sulfur dioxide is heavier than air, so on inhalation, it causes the person to severely cough [40]. SO_2 is mainly used to measure for all sulfur oxides SO_x. Oxides are a group of highly reactive gases. But, SO_2 can mainly be measured since other sulfur oxides occur in much smaller quantities [21]. It is used to manufacture many

different types of chemicals ranging from household detergents, building materials, such as flooring, tile, sinks, bathtubs, and drywall, paper pulping, metal, batteries, and food processing. Sulfur dioxide air pollution mainly comes from burning coal or oil at power plants or other industrial plants, or copper smelting [40]. High production of sulfur dioxides can increase the likelihood of SO_2 oxidizing into sulfur trioxide SO_3 particles. Sulfur oxides SO_x can react in the atmosphere to form poisonous particulate matter (particles). As previously mentioned, particulate matter, including SO_x, can cause haziness commonly found around cities and busy industrial zones.

Nitrogen Dioxide Nitrogen dioxide (NO_2) is a reddish brown gas or yellow-brown liquid. It is a highly reactive and poisonous gas [39]. Like SO_2, NO_2 is a part of group of highly reactive gases, and is used as an indicator for other forms of nitrogen oxides. Nitrogen dioxide emissions mainly come from the burning of fuels, such as from combustion engines and vehicles, power plants, or other industrial plants [20]. Like SO_x, NO_2 and NO_x can react to form particulate matter, which can also be inhaled. Nitrogen oxides are also what make up for smog and haze during warm seasons in cities and industrial areas. Nitrogen oxides can also react in the atmosphere with water and other vapors to form acid rain, which can pollute water supplies and ground soil [20].

3.1.2 Adverse Effects on Health

Particulate Matter (fine particles) can potentially enter the bloodstream after being inhaled. Long term exposure to $PM_{2.5}$ was found to be related to cardiovascular diseases and infant mortality. Particularly, people with pre-existing respiratory problems will be more susceptible to the worsened negative effects of $PM_{2.5}$, along with other pollutants. Short-term exposure to PM has shown to affect people's lungs and heart. Countless scientific studies have linked particulate matter pollution to the following: premature death to those with pre-existing lung or heart diseases, nonfatal heart attacks, irregular heartbeat, aggravated asthma, decreased lung function, airway and lung inflammation causing coughing or difficulty breathing [18]. However, there are studies correlating long-term, high exposure to various air pollutants to chronic, life-long cardiovascular, neurological, and psychological complications [5]. Wildfire smoke can cause long-term, chronic heart and lung disease. Wildfires can put anyone at risk that are in at-risk areas. Wildfire smoke has also been known to travel thousands of miles, affecting areas to varying degrees [24].

Ozone Tropospheric ozone can have many short-term adverse health effects. It is currently inconclusive whether or not ground-level ozone has any long term consequences for individuals. However, even short-term exposure has shown to slightly increases chances of mortality (0.5 to 1.04%) for the general population, but especially for those with pre-existing respiratory conditions, such as asthma, or those over the age of 65. Short-term symptoms include: lung and airway inflammation, coughing, throat irritation, pain, burning, or discomfort when breathing. Ozone is able to reach the lower respiratory tract since it is less water soluble, unlike sulfur dioxide

Table 5: Unhealthy Exposure Levels [25]

$PM_{2.5}$ $\mu g/m^3$(24h)	PM_{10} $\mu g/m^3$(24h)	CO ppm(8hr)	O_3 ppm(1hr)	SO_2 ppb(1hr)	NO_2 ppb(1hr)
55.5 - 150	255 - 354	12.5 - 15.4	0.165 - 0.204	186 - 304	361 - 649

and chlorine gas. Studies have shown that people exhibit approximately a 50% drop in forced evacuated 1-second volume (FEV1) (a measure for lung function) for exposures to 400 ppb over 2 hours, and similarly for a 80 ppb exposure for 5 hours. The result was severe coughing and an increased rate of asthma attacks [19].

Carbon Monoxide Carbon monoxide can, in high indoor levels, cause confusion, dizziness, unconsciousness, and death. For outdoor levels during short-term exposure, it can cause chest tightness [17].

Sulfur Dioxide and Nitrogen Dioxide As previously mentioned, SO_2, NO_2, and other oxides (SO_x and NO_x) are dangerous, poisonous compounds that come in the form of solid particles and gases. The EPA has strong control measures for SO_2 and NO_2 emission producers. Also, since SO_2 and NO_2 can oxidize, controlling SO_2 and NO_2 can also control its derivates, and in this case, the particulate and gaseous forms of oxide compounds. SO_2 and NO_2 have immediate, short-term health effects. It can cause choking, since they are both heavier than normal air [21]. However, after removing SO_2 from the environment, people recover from short-term symptoms in about 30 minutes. For people with asthma, this time is approximately 4 hours [5]. However, for NO_2, while it may cause slight pain and lung inflammation, longer exposures can lead to death many days after when they where exposed [39].

Table 6: The Different AQI Levels [25]

AQI Values	Levels of Health Concern	Colors
0 - 50	Good	Green
51 - 100	Moderate	Yellow
101 - 150	Unhealthy for Sensitive Groups	Orange
151 - 200	Unhealthy	Red
201 - 300	Very Unhealthy	Purple
301 - 500	Hazardous	Maroon

3.2 The Air Quality Index (AQI)

The air quality index (AQI) is used to indicate the current air quality within a standardized range. The AQI was established through the Clean Air Act in 1963. An Air Quality Index can be calculated from readings of any 7 measured pollutants

over a 24-hour time period since the science on air pollution exposure is based on 24-hour long exposures. The pollutants include PM2.5, PM10, SO_2, NO_x, NH3, CO and O_3. For PM2.5, PM10, SO_2, NO_x, and NH3 the average value over the last 24-hrs is used as long as there are a minimum of 16 hourly measurements. For CO and O_3 the maximum value in the last 8-hrs is used. Table 5 summarizes the measurement timescale for each pollutant and also details the concentration required to be classified as "unhealthy." At times, there will be no available readings present for many gases, in this situation AQI is calculated based on the gases present using sub-indexes for each gas. The final AQI measurement is calculated with the condition that at least one of PM2.5 or PM10 measurements are available and at least 3 out of 7 gas pollutant measurements available. The EPA sets standards for exposure levels for each of the gasses based on scientific research. The levels can be advised over time. The range of the index is from 0 to 500 with an AQI value of 100 being the national air quality standard for each of the pollutants. Various levels have been defined within 0 and 500 to indicate various health risks as shown in Table 6.

3.3 NowCast AQI

Since the AQI is based on a 24-hour time period, in order for the EPA and municipalities to report hourly air quality data, the EPA established the NowCast AQI to estimate 24-hour exposures so that people can reduce their exposure. The NowCast AQI can be used to measure the near real-time air quality due to the effects of natural or man-made disasters. The EPA established algorithms to measure the ozone and particulate matter NowCast AQI [28]. Both ozone and PM NowCast algorithms use linear regressions, however the ozone NowCast requires the use of the partial least squares linear regression as detailed in their ozone NowCast AQI white paper and GitHub page [27].

4 Related Standards and Design Constraints

In the following section we will discuss both standards that our design utilized and constraints that our design adhered to. Standards are important to make the design process and parts acquisition easier. Understanding and following the constraints throughout the design process allowed us to have a functional device that behaved as expected when tested on the subsystem and end-to-end level.

4.1 Related Standards

Standards are an important part of any engineering design. Standards are employed in order to help ensure interoperability between different components within a design or between different designs. In our design, we utilized a variety of different hardware and communication standards. Specifically, we will be discussing the different standards used to communicate between the microcontroller and different sensors on the node, such as I^2C, UART, and SPI. We will also describe the LoRa and LoRaWAN standards, which are used to send data from the nodes to the remote server.

4.1.1 I^2C

Inter-Integrated Circuit (I^2C) is a synchronous, multi-master, multi-slave, serial communication bus. The standard was designed in 1982 by Philips Semiconductor and was originally proprietary and used exclusively by Phillips. The first public specification for I^2C was published in 1992 and it is now used by almost all IC manufacturers. I^2C is a popular bus because of how simple it is to use. The protocol specifies the speed of only the upper bus and just two wires that are connected to V_{DD} with pull-up resistors are required in order to connect a nearly unlimited number of I^2C devices. The two wires used are the serial clock wire (SCL) and the serial data wire (SDA).

The addressing scheme for I^2C specifies the use of a 7-bit address. Each I^2C device connected to a bus is required to have a unique address. This can be achieved by have a fixed unique address or a set of I^2C addresses that can be selected from to ensure a unique address is used. It should be noted that the address bus does have eight bits total on the bus, however, only bits 7 through 1 are used for the address, while bit 0 is used for indicating the read/write status of the device. A 1 indicates that the master device will read from the slave device. Master devices do not require addresses, since they are the ones generating the clock and they address I^2C devices connected to the address bus.

The I^2C protocol defines a set of states that the SCL and SDA wires can be in. This defines what the overall state of the I^2C bus is in. In the idle state, the SCL and SDA wires are both high. Communication begins when the master device generates the start condition. The start condition is defined when SCL is high and SDA transitions from high to low. The master then sends the address of the slave

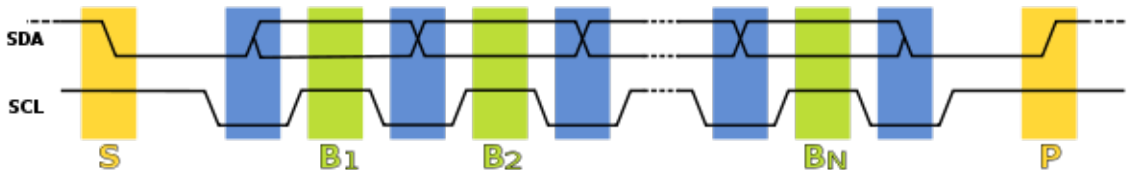


Figure 4: A signal graph showing an example of an I^2C transmission. Used with permission from [30].

device it wants to communicate with. The address itself is sent over the SDA line along with a series of high/low pulses on the SCL line, indicating when the SDA line should be read. If the read/write bit of the address is set to 0, the master will begin writing to the slave device, otherwise, if the read/write bit is 1, the next byte will be read from the slave device. Once all bytes have been read or written, the master device will generate the stop condition. The stop condition is defined when SCL is high and SDA SDA transitions from low to high. This puts the bus back into the idle state.

In our design, I^2C is utilized as the communication protocol between the gas sensors and the microcontroller. Using the I^2C standard will make it simpler and easier read data from the sensors, as opposed to trying to convert and analyze direct analog inputs from sensors that do not provide a digital interface.

4.1.2 SPI

Serial Peripheral Interface (SPI) is a synchronous serial communication protocol. SPI was created by Motorola in 1979. The protocol operates in full duplex mode, meaning devices connected over SPI can communicate simultaneously. The protocol uses a master/slave architecture. The master device can communicate with multiple slaves.

The signals that each device uses in a SPI setup is slightly different depending on if it is a master or slave device. The signal descriptions are as follows:

- SCLK: Serial Clock
- MOSI: Master Out Slave In
- MISO: Master In Slave Out
- CS: Chip Select (sometimes called Slave Select (SS))

The SCLK carries the clock signal and goes from the master to all slave devices. The MOSI signal carries data from the master to each slave. The MISO signal carries data from each slave to the master. The CS signal is used to determine which slave is currently communicating with the master. For the CS signal, there are a few different

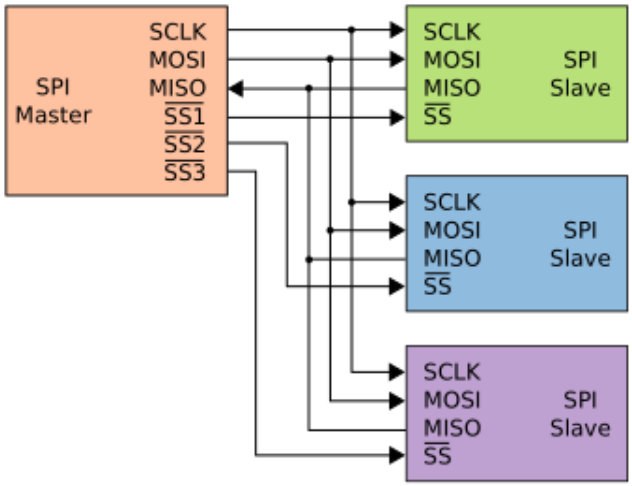


Figure 5: An example of the independent slave configuration of SPI. Used with permission from Colin Burnett, under the CC BY-SA 3.0 license. [12]

configurations that are possible. Those configurations are listed as follows:

- Independent Slave Configuration: This is the most common configuration used. There is one CS for every slave device connected. An example showing this configuration, is shown in Figure 5. This figure also shows how the signals are connected in SPI.

- Daisy Chain Configuration: In this configuration, there is a signal CS signal going from the master to each slave device. This configuration also somewhat changes how the MOSI and MISO signals of slave devices are used. The MISO signal goes from one slave to the MOSI pin of another slave, this is repeated until the last slave in the chain connects it's MISO pin back to the master's MISO pin.

The process of SPI communication occurs during clock cycles. SPI communication begins by the master configuring the clock and then selecting the slave device which is set to low on the CS line. Then, during each clock cycle, data transmission occurs over the MOSI and MISO lines between the master and slave. The master reads a word of data, usually 8-bits, over the MISO line and the master transmits a word of data over the MOSI line. This exchange of words can occur multiple times. Once the final word has been transmitted between the master and currently selected slave, on the next clock cycle the next slave is selected and the process repeats.

4.1.3 UART

Universal Asynchronous Receiver Transmitter (UART) is one of the most popular and widely-used communication protocols. UART is relatively simple protocol,

Start Bit	Data Frame	Parity Bits	Stop Bits
(1 bit)	(5 to 9 Data Bits)	(0 to 1 bit)	(1 to 2 bits)

Table 7: The structure of the UART data packet.

requiring only two wires to function. Each UART device utilizes two pins, an receiving (RX) pin and a transmitting (TX) pin. In a situation where two UART devices are connected with one another, the first device's RX pin will be connected to the second devices TX pin and the first device's TX pin will be connected to the other devices RX pin. Another requirement of UART is that the baud rate must be the same for two UART devices that are connected to each other.

The UART protocol defines that data is sent in a packet structure (Figure 7). A UART packet includes a start bit, the data frame, a parity bit, and one to two stop bits. In the idle state, the TX line remains high. Communication begins when the TX line transitions from high to low for one clock cycle. After the start bit, the data frame, which contains the actual data being sent, is transmitted. This is performed by setting the TX wire to high when a 1 is transmitted and to low when a 0 is transmitted. This wire is sampled by the receiving UART device at the predetermined baudrate. Due to the fact that the parity bit is optional, the data frame may be eight or nine bits long. After the last data bit is transmitted, the parity bit is transmitted. This bit used to tell if any data has been altered during the transmission process. The parity bit is calculated by summing the 1's and 0's of the data frame and then, depending on whether or not the sum is even or odd, the parity bit is set to 0 or 1 respectively. Finally, the stop bit is transmitted to indicate the end of the packet. This is done when the TX line transitions from low to high for one or two clock cycles.

4.1.4 USB

Universal Serial Bus (USB) is an industry standard the creates specifications for physical connectors and communication protocols for those connectors. The goal of the USB standard is to simplify and improve the interface between personal computers and peripheral devices. The different communication protocols that USB specifies include USB 1.0, USB 1.1, USB 2.0, USB 3.0, USB 3.1, USB 3.2, and USB4. This list of communication protocols dates back to the inception of USB in 1996. Today, only USB 3.0 and up are common on new devices. The different connectors include Type A, B, and C. The Type C connector is the most modern connector and supports the newest communication protocols, reaching a maximum data rate of 40 Gbps. However, many new devices still support the Type A connector.

4.1.5 LoRa

LoRa (Long Range) is a proprietary, wireless modulation technique derived from chirp spread spectrum (CSS) technology. LoRa is was developed and patented by Semtech. LoRa only covers the physical layer. An additional protocol is necessary in

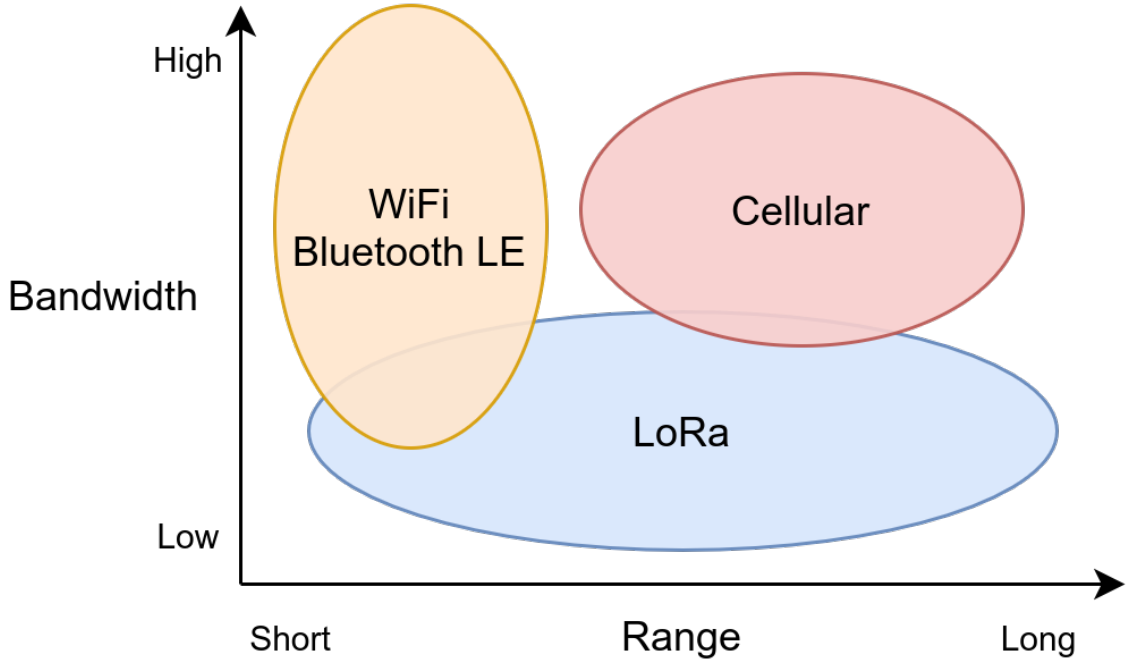


Figure 6: A range-bandwidth comparison between WiFi and Bluetooth LE, cellular, and LoRa.

order to specify the upper layers required for a functioning communication system. LoRaWAN is the one of the most widely used protocols to fulfill this purpose and will be discussed later in this section.

One of the reasons for choosing LoRa over other communication standards, such as WiFi or cellular, is due to where it stands in relation to bandwidth, range, and power. Technologies such as Bluetooth and WiFi are considered high bandwidth, but low range. Cellular technology is high range and can be high bandwidth, but a major drawback is the high operating power required. In contrast, LoRa is low power, high range, and can operate in the license-free frequency band in the United States, as well as in other countries. Figure 6 shows a comparison between these different types of technology.

In order to explain how LoRa functions, we must first give a brief overview of Chirp Spread Spectrum (CSS). CSS is a communication technique that utilizes its entire allocated bandwidth to broadcast a signal. It uses wide-band linear frequency modulated chirp pulses to encode information. A chirp is simply a signal whose frequency changes over time according to specific, time-varying function [1]. In the case of LoRa, the frequency band that is used is the license-free sub-GHz radio frequency that varies by region. For example, in the United States the license-free radio frequency band is 920 MHz to 928 MHz. The modulation technique that LoRa uses is derived from CSS. It functions by representing each bit of data in a payload as

multiple chirps. LoRa defines a term known as spreading factor (SF) to refer to the number of symbols sent per bit of information. Since multiple chirps are used to represent a single bit of data, spreading factor is also the relationship between the nominal symbol rate and the nominal chirp rate. The creates a discrete modulation where the $M = 2^{SF}$ possible waveforms at the output of the modulator are chirp modulated signals over the frequency interval $(f_0 - B/2, f_0 + B/2)$, where B is the bandwidth. There are M initial frequencies [4].

LoRa supports the ability to trade fixed sensitivity within a fixed channel bandwidth for data rate. It does this by modifying the SF parameter. A higher SF has the benefit of giving the receiver more chances to sample the signal power, providing better sensitivity. However, the drawback to a higher SF value is the greater transmission time and thus higher energy consumption, which is clearly a negative for the low-power applications that LoRa devices are typically used in. Understandably, a lower SF value results in more chirps per second, meaning worse sensitivity for the receiver. However, the benefit is the lower power consumption [46].

Another parameter of LoRa is the transmission power. Depending on the region the specific LoRa module is targeted for, the supported range of transmission power ranges from 2 dBm to 14 dBm in certain areas in Europe and Asia to as high as 20 dBm in some parts of Europe and all of North America. This means LoRa can support a transmission distance of up to 10 miles in ideal conditions e.g. line of sight.

4.1.6 LoRaWAN

LoRaWAN (Long Range Wide Area Network) is a protocol that defines the network portion of a LoRa stack. There are many different protocols that exist to fulfill the same role as LoRaWAN, however, LoRaWAN is public and is widely supported by many different hardware vendors. In fact, the LoRa Alliance was created in 2015 to support the development of LoRaWAN and ensure the standardization of all products and technologies utilizing LoRaWAN. The LoRa Alliance is composed of many different technology companies and organization, including high-profile members such as IBM and Cisco.

LoRaWAN is often referred to as using a "star of stars" network technology. What this means is that LoRaWAN consists of a collection of gateways which communicate with a central i.e. a classic star network topology. On top of this, each gateway communicates with a collection of end-devices, often referred to as nodes as they are in our design, creating a star sub-network topology. Figure 7 shows the layout LoRaWAN's network topology. LoRaWAN specifies that the gateways communicate with the server using IP packets. This means that the gateways are simply acting as a bridge between the RF signals coming from the end-devices and the IP packets being transmitted to the central server. In the context of the OSI model, LoRaWAN operates in between the physical LoRa standard and the application layer [7].

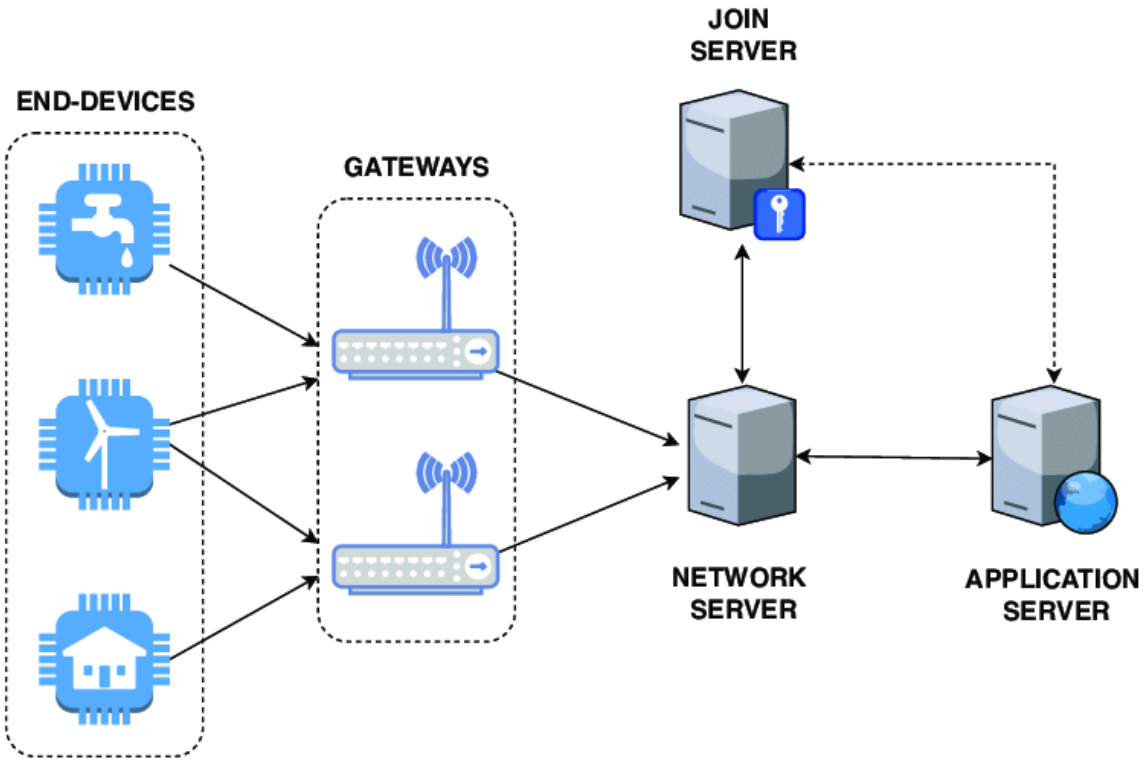


Figure 7: The LoRaWAN network topology. Used with permission under the Creative Commons Attribution 4.0 International license. [37]

The LoRaWAN specification specifies three possible different end-device types, which are summarized in Table 8 and described in more detail below:

- **Class A** devices are described as lowest power, bidirectional end-devices. For an end-device to compliant with the LoRaWAN specification, it must be able to function as a Class A device. Class A devices always initiate communication with the central server and can do this asynchronously. Every time a Class A device begins an uplink transmission, it must be followed by two downlink transmissions. This allows the server to send follow-up communication, hence the bidirectional description of the class. The LoRaWAN specification states that the central server must buffer downlink communication until the next uplink transmission from the end-device. Once an end-device has finished sending and receiving transmissions, it can enter a low-power state that it can define.

- **Class B** devices are described bidirectional end-devices with deterministic downlink latency. Class B devices still initiate communication with the central server, however, they initiate this communication during scheduled, periodic intervals. This means that the central server can send a downlink transmission predictably. This means that the end-devices will consume additional power. This transmission latency can be configured up to a maximum of 128 seconds.

Class	Latency	Power Consumption	Transmission Pattern
Class A	Highest	Lowest	End-device initiated (asynchronous)
Class B	Middle	Middle	End-device initiated (synchronous)
Class C	Lowest	Highest	Continuous

Table 8: A summary of the three different LoRaWAN end-device classes.

- **Class C** devices are described as bidirectional end-devices with the lowest possible downlink latency. This lowest possible downlink latency is achieved because the end-device is simply ready to receive at all times. The only time the end-device is not ready to receive is when it is transmitting data back to the central server. Due to the large power consumption, the ideal situation for Class C device would be for it to be connected to external power. For battery operated applications, it is possible to have a device temporarily switch between Class A and Class C.

LoRaWAN also specifies a parameter known as data rate. The data rate parameter controls the relationship between communication range and message duration. In addition, due to how spread-spectrum technology functions, end-devices communicating with different data rates do not interfere with one another. This effectively increases the capacity of each gateway by essentially creating multiple separate channels which are delineated by the data rate parameter. Taking advantage of this information, LoRaWAN specifies the creation of a scheme known as Adaptive Data Rate (ADR). The ADR scheme is defined by having the central server manage the data rate parameter and transmission power for each end-device. This can allow the central server to optimize battery life for end-devices and optimize the capacity of the network. The possible values for the data rate range from 0.3 Kbps to 50 Kbps.

The LoRaWAN specification also takes into account security. The specification details two layers of security. This is done via the inclusion of a 128-bit AES Network Session Key, which is shared between an end-device and the network server, and a 128-bit AES Application Session Key, which is shared between the end-device and network server at the application level. This results in the benefit of having authentication and verification of packets traveling from end-devices to the central server and end-to-end packet encryption from the end-device to the end application server. These features make it possible to have shared networks without the owner of the network being able to view packets.

4.2 Design Constraints

In this section, we discuss all of the realistic design constraints that exist for our project. We will also describe the impact these constraints had on our design. The constraints that are discussed include economic, time, manufacturability, sustainability, environmental, health, safety, ethical, health, social, and political constraints.

4.2.1 Economic Constraints

Cost will be one of the greatest constraints on our design. One of the main ways that this constraint manifests is in the selection of sensors in our design. Since our goal is to measure the concentration of various gases and particulates in the air, the accuracy and sensitivity of the sensors we choose to incorporate in our design is critical. During our research of different types of sensors, we found that sensors can range from being designed for low-cost, hobbyist applications all the way to expensive, high-end sensors that are designed for safety-critical, industrial applications.

For our design, we have chosen to design a system that can compute the air quality with respect to the EPA AQI. More specifically, we focused our sensor selection to pollutants defined in the EPA NowCast AQI as they have found through numerous studies that ozone and particulate matter are the main pollutants that can generalize the overall AQI. Therefore, we only need to select ozone and particulate sensors that are sensitive enough to differentiate between the different index categories e.g. from moderate to unhealthy. Any increased sensitivity and accuracy that would come from purchasing more expensive sensors would be unnecessary. This means we should strive to meet this minimum bound for each sensor in our design in order to effectively optimize the cost for our design.

Cost also plays a role in the selection of a microcontroller for our design. The microcontroller will likely be the second most expensive component in our design. The microcontroller is a key component in the design and at the minimum we are looking for one that is packaged with a LoRa module. Some also come packaged with Bluetooth and WiFi transceivers.

4.2.2 Time Constraints

Time will play a major role in the development of our project. There will be many deadlines that we will have to meet as we move forward and the complexity of our design will need to be adjusted accordingly. If we decide to add too many features or complexities to our design we may run out of time. While certain design ideas may be better for the functionality of our project, if they take too long to implement then they are not viable and must be discarded in favor of ideas that can be completed within our time frame.

4.2.3 Manufacturability and Sustainability Constraints

Manufacturability refers to constraints related to how a design can be produced. If manufacturability is not considered during the design process, the end product may not be possible to build. One of these constraints would be that we should design our circuits to use off the self components. Another manufacturability constraint would be that the PCB we design must be within a certain range of dimensions, as PCB manufacturers will not create a board that is too small or too large.

Sustainability refers to the ability of a design to be long-lasting and to resist degrading over time. Sustainability constraints are important to our design due to the sensor nodes being placed outdoors. We want these sensor nodes to be long lasting, therefore the electronic components of the node should ideally be enclosed in weatherproof housing. However, it is not a strict requirement for our project since we are not focusing on any mechanical aspects. In addition, the enclosure and PCB design should follow the sensor's datasheet for optimizing sensor placement.

4.2.4 Environmental, Health, and Safety Constraints

Environmental constraints are constraints that focus on how the design impacts the earth and the usage of the earth's resources. Our design utilizes a rechargeable battery rather than disposable ones, such as AA. In addition, our design will contain solar panels that will be used to recharge the battery, increasing its longevity, decreasing maintenance, and decreasing its energy footprint as opposed to charging it through a wall adapter. We also must consider the material choice for the enclosure of our design. As such, our 3D model is printed with a recyclable plastic.

Health and safety are constraints that are necessary to keep the design from potentially causing harm to an individual. In our design, one of the main potential safety hazards is electromagnetic radiation. The LoRa IC we used is certified by the Federal Communications Commission (FCC) in areas related to reducing the of the user to electromagnetic radiation. All electrical contact points and wires shall be properly insulated in order to prevent any injury from electric shock. The housing for the node will be able to be securely mounted to the desired surface to prevent it from falling and potentially injuring someone.

4.2.5 Ethical, Social, and Political Constraints

In terms of political constraints, our design shall follow all FCC guidelines when it comes to transmitting radio signals and should be fully certified. The LoRa IC we chose for our design shall be able to transmit within the 900 MHz to 928 MHz frequency band, which is the frequency band designated by the FCC to permit unlicensed transmission. We are not able to determine any ethical or social constraints that apply to our design.

5 Design Research

The goal of this section is to explore the various potential components used in each area of the design including the main compute module, networking module, gas sensors, and power components. Research was conducted to evaluate current work in air quality sensors, as well as major hardware components require for the system. Due the ongoing supply chain crisis, shipping any electronic components from China is taking much longer. Therefore, we must consider choosing components that are currently stocked in American warehouses to avoid unacceptably long shipping delays. In our research, we looked at what worked and didn't for existing low-cost air quality sensors. We also considered various trade-offs in our sensing design to keep costs low and measurements as accurate as possible with what is available on the market.

5.1 Analysis of Similar Existing Products

In this section, we will provide a brief survey similar existing AQI monitoring/pollution monitoring nodes. We used what we learned from exploring these existing solutions in order to provide some guidance as to what we should strive towards in terms of design requirements. By analyzing these existing devices, we can better determine where our project should be in terms of factors, such as cost, battery life, and feature set.

5.1.1 MClimate AQI Sensor and Notifier LoRaWAN

The MClimate AQI Sensor and Notifier LoRaWAN is an indoor AQI sensor. The device is available on MClimate's website for $146. It is described as being handle 10+ years of battery life. It also features a sensor from Bosch that is able to detect VOCs, temperature, humidity, and barometric pressure [38]. The specific sensor model is not listed on the datasheet, however, it seems to be similar to the one we are planning on incorporating into our design. With regards to the specifics of the LoRa transceiver, the device transmits at 14 dB and has a link budget of 130 dB. The device communicates data over LoRaWAN to a server owned by the company. This data is then able to be viewed via a mobile app. In addition, the device features an audible alarm and visual LED indicator to warn users if the AQI drops below a determined threshold. The listed physical dimensions are 80 x 80 x 19 mm. The listed weight is 68 grams.

The device is similar to our proposed design in some ways, but there are some key differences. A major advantage of our design is the inclusion of a particulate matter sensor. While the MClimate AQI Sensor is marketed as indoor air quality sensor and particulate matter is not issue in that environment, the addition of a particulate matter sensor allows our design to function outdoors as well. Another benefit of our design is the increased transmit power that the LoRa module we selected supports.

5.1.2 Davis Instruments AirLink

The Davis Instruments AirLink is an air quality monitoring device that supports measuring indoor and outdoor air quality. It features a particulate matter sensor that can perform measurements of PM1.0, PM2.5, and PM10. Additionally, the device measures temperature, humidity, dew point, heat index, and wet bulb. The device also supports sending data to a mobile app and website. The device does not use LoRaWAN and instead uses WiFi. The device does not support the use of a battery, meaning it must be connected to a power source with a 5 V DC-AC adapter. The AirLink supports both indoor and outdoor operation. To support this, the AirLink includes a weatherproof cover that can be placed over the device to protect it from the outdoor elements. The listed physical dimensions without the weather cover are 2 x 3.5 x 1 in and the dimensions with the cover are 4 x 4.5 x 1.5 in. The weight of the device without the cover is 3.7 oz and the weight of the device with the weather cover is 6.5 oz [32].

Compared to our proposed design, this device has a somewhat similar feature set. Both our design and the AirLink support particulate matter measurement, as well as all of the other various weather measurements. However, a there exists a major difference between the portability and flexibility of the AirLink and our proposed design. Due to the fact that the AirLink only supports WiFi and must be connected directly to a wall outlet makes this device have a very small range. Taking advantage of the increased communication range provided by using LoRaWAN allows our design to be placed almost anywhere. In addition, having a battery makes our design able to be placed in more locations and also makes it easier to install.

5.2 LoRa Networking and Compute Module

Most LoRa modules that have integrated RF solutions already have integrated ARM cores of some kind. This makes development easier, lowers power consumption, and allows for a cheaper BOM. Many different types of LoRa modules were considered. The modules that were most sought after were ones with integrated U.FL or SMA connectors in order to ease PCB development. However, modules that did not have these connectors were still considered.

The main considerations in choosing a LoRa module were the following:

- Power consumption
- Transmit power
- Receive sensitivity
- Software stack quality

- Documentation availability and quality
- Digital I/O
- Analog peripherals
- PCB design complexity
- Cost

Choosing a module with high enough power consumption and receive sensitivity is important for increasing the range of the device. The transmit power, receive sensitivity, signal gains, and signal losses affect the link budget.

5.2.1 The Seeed LoRa-E5

The Seeed LoRa-E5 utilizes a STM32WLE5JC microcontroller, which is the first to integrate both a microcontroller and a +22 dBm LoRa radio into one silicon wafer. The Seeed LoRa-E5 simplifies development with the STM32WLE5JC by embedding the clocks and main RF front-end circuitry into a single package, obtaining up to +20.8 dBm transmit (TX) output and -136 dBm receive sensitivity, and takes care of startup and shutdown power sequencing. All in a 12 mm by 12 mm package. This microcontroller provides the best power consumption of 360 nA while in standby, 58μA while in low power sleep current (LPSleep), and 125μA while in low power run current (LPRun, $<$ 2 MHz) with a max LoRa transmit current of 110 mA at 22 dBm. STM also provides a hardware abstraction layer (HAL) for the device, and middleware, such as FreeRTOS and a FAT filesystem driver, and a LoRaWAN software stack all through their STM32Cube software package. The STM32WL utilizes an Arm Cortex M4 32-bit RISC core with 256 KB flash memory and 64 KB of SRAM. It also has a 12-bit ADC, which will be needed for the sensing sub-system. I/O connectivity is provided through also has 3 UART controllers, 1 SPI controller, 1 I^2C controller, programming pins, and 6 to 8 GPIO, depending on the UART configuration.

However, the module provides no integrated U.FL connector, which increasing the complexity of the PCB design and increases the number of components in the BOM. When compared to other modules, this poses as a big drawback to using this module along with the fact that it provides no USB or Bluetooth connectivity out of the box. Even then, the module simplifies the RF traces needed outside the module, and the reference design (development board) uses a short, straight trace to a SMA connector with a 0Ω resistor to allow the user to switch to a U.FL connector. Another positive aspect is the fact that it is FCC and CE certified [56][57].

5.2.2 The Midatronics Windy

The Midatronics Windy also utilizes a STM32WLE series microcontroller (specifically the STM32WLE5J816), but otherwise is almost identical to the Seeed LoRa-E5.

The only real difference is the Midatronics Windy is a fully integrated surface mount device module that also includes an U.FL connector on the package. This drastically decreases the PCB complexity and requirements, but is double the cost of the Seeed LoRa-E5. The analysis of the various trade-offs between the PCB requirements (material, trace width, trace spacing, etc) and the LoRa module chosen is discussed in a later section. Just like the Seeed LoRa-E5, it is FCC and CE certified. However, not much else is publicly documented, but the rest can be assumed since it uses almost an identical microcontroller to the Seeed LoRa-E5 [42][41][56].

5.2.3 The Microchip WLR089U0

The Microchip WLR089U0, like the Midatronics Windy, is a fully integrated surface mount device module with an U.FL connector. The module utilizes an Arm Cortex M0+ processor with 256 KB of flash memory and 40 KB of SRAM. It has a 18.6 dBm transmit power, and a receive sensitivity of -136 dBm. It has ultra-low power sleep modes that offer low current consumption of as low as 790 nA, 12.6 mA while receiving, and 114.6 mA while transmitting at 18.5 dBm. There is not much documentation about the various power modes of the device. It does offer a 12-bit ADC with 7 channels which allows for easier sampling of multiple inputs, albeit at a lower rate, with only one physical ADC. Since an air quality sensor doesn't need that many readings per second (when active and taking measurements), the lower sampling rate doesn't pose as an issue. The module also has I^2C, SPI, UART, and USB I/O peripherals. Having a USB I/O digital interface built into the module saves money and design complexity, and makes up for the module's lower performance [60][59].

5.2.4 RAK 4600

The RAK 4600 LPWAN module is based off the Nordic nRF52832 MCU and the Semtech SX1276 Lora transceiver. The Nordic MCU is a general purpose, multi-protocol MCU capable of Bluetooth 5.2 with a +4 dBm TX output and Near Field Communication (NFC). The addition of peripherals would increase the usability and accessibility of the air quality sensor by allowing the user to connect and configure the sensor wirelessly through their phone or computer. The Semtech SX1276 empowers the RAK 4600 to have a LoRa TX output of +20 dBm while using 125 mA, and -148 dBm RX sensitivity while using a max of 14 mA. However, only either BLE or LoRa can be used, not both at the same time. The Nordic MCU supports ultra low-power sleep states than can achieve down to 1.2μA of draw current. However, RAK states that that the RAK 4600 module can only get down to 11.5μA at 2.0V input and 12.5μA at 3.3V.

The downsides of the RAK 4600 mostly come down to three things:

1. Its sleep power consumption

2. Its complete lack of I/O, especially analog I/O.

3. Its lack of a LoRaWAN software stack.

The RAK module does not allocate any pins for analog I/O even though the Nordic MCU has a plethora of digital and analog peripherals connected to its pins. The RAK module surprisingly mostly has no connect (NC) pins. Not only that, but it only has one set of UART pins and one set of I^2 pins. While the addition of Bluetooth would be a great addition, it would greatly complicate the sensor subsystem by not providing any analog I/O whatsoever. While the lack of a LoRaWAN stack is definitely not a deal breaker (and would be a great learning experience), it is definitely something to take into account, especially since it seems to be the standard across most LoRa modules researched [67].

5.2.5 The Onethinx Core

The Onethinx is another module that supports both LoRa and Bluetooth in a small 25mm by 20mm package. The module includes embedded wire trace LoRa and Bluetooth antennas, greatly simplifying PCB design. The wire LoRa antenna still provides +20 dBm out TX output power, but is lacking compared to the other modules with its -94 dBm RX sensitivity. It is powered by the Cypress PSoC 6x series MCUs, which houses an ARM M0+ core, M4 core, and a Bluetooth transceiver, along with a separate chip, the Semtech SX126x, for the LoRa transceiver. The Cypress PSoC focuses on power efficiency and security. The Cypress PSoC implements the ARM Platform Security Architecture (PSA) which provides three levels of hardware-based resource isolation. Since it has two cores, the user code can execute separately from the LoRaWAN stack so that the user code can never interfere with the LoRaWAN communication link.

On top of that, the LoRaWAN stack implements a chain of trust, which essentially is a hash chain formed by hashing the decrypted code segments at various parts of the stack, and comparing the hash with the public key. However, the software stack is not publicly available, and there is not much official or online support (StackOverflow, etc), so if issues were to arise, it would be much more difficult to fix, especially if deadlines are quickly approaching. Another factor to consider with the Onethinx Core module is: the availability. The module is not freely purchasable through major online electronics distributors (Digikey, Mouser, Arrow, etc). It is only purchasable through email correspondence, which has proven to be fruitless, even after 5 business days [44][49][14].

5.2.6 LoRa Module Comparison

In the following discussion, the LoRa modules are compared. Of the listed considerations at the beginning of the section, energy efficiency, documentation quality, and cost were the main deciding factors. The reason for prioritizing those three characteristics were two-fold: time constraints and cost constraints. Choosing the part with ample amounts of documentation significantly affects development speed, and consequently allowing for rapid prototyping and last-minute changes.

As shown in Table 9, the modules are summarized in regard to their computing capabilities and cost. Most of the modules have comparable costs for the modules, but the development kits for each of them drastically vary. The Seeed LoRa-E5 is the most cost effective for both the individual modules and development boards. Compared to the Midatronics Windy, it is essentially the exact same system with the only difference being that the Midatronics Windy has an embedded U.FL connector on the package for double the price. The ARM Cortex-M4 (and Cortex-M0+ for that matter) is more than enough to meet the computing requirements of the air quality sensor nodes since all they have to do is:

1. Periodically wake up.

2. Send data to a gateway.

It is more of an added benefit that the Onethinx Core has two cores. However, having two cores, and requiring both of them for its specialized, secure LoRaWAN stack, increases the power usage of the device. Another thing that is a nice-to-have (and uses more power), but not necessarily a need-to-have, are the Bluetooth peripherals on the Onethinx Core and the RAK4600. However, having Bluetooth would create a better user experience, but mainly only for initially configuring the device.

Table 9: LoRa Module General Comparison

Module	Core(s)	Core Clock (MHz)	Cost	Dev Board Cost
Seeed LoRa-E5	Cortex-M4	48	$8	$18
Midatronics Windy	Cortex-M4	48	$20	$65
Microchip WLR089U0	Cortex-M0+	48	$15	$114
RAK 4600	Cortex-M4F	64	$10	$18
Onethinx Core[1]	Cortex M0+ & M4	100/150	$20	$80

When researching and comparing the module's electrical characteristics, we mostly focused on the power efficiency so that we can meet our goal of having the battery life can last as least one year. In order to meet our range requirements, we looked for a LoRa tranceiver with a 150dB direct line-of-sight link budget. The link budget can be calculated with the equation below, but it is provided in most, if not all, cases.

$$P_{RX} = P_{TX} + G - L \tag{1}$$

[1]Not readily available at a distributor or official website. Must email company.

where

P_{RX} : received power (dBm)
P_{TX} : transmitted power (dBm)
G : gains (dB)
L : losses (dB)

The full link budget equation that accounts for all sources of gain and loss is shown below:

$$P_{RX} = P_{TX} + G_{TX} - L_{TX} - L_{FS} - L_M + G_{RX} - L_{RX} \tag{2}$$

where

P_{RX} : received power (dBm)
P_{TX} : transmitter output power (dBm)
G_{TX} : transmitter antenna gain (dBi)
L_{TX} : transmitter losses (coax, connectors, etc) (dB)
L_{FS} : path loss, usually free-space loss (dB)
L_M : miscellaneous losses (fading margin, body loss, etc)(dB)
G_{RX} : receiver antenna gain (dBi)
L_{RX} : receiver losses (coax, connectors, etc) (dBm)

Table 10 tabulates the researched module's electrical Characteristics. "TX" and "RX" refer to the transmit power and receive sensitivity, respectively. The sleep current is defined as the lowest possible sleep state where the core(s) can still be woken up by a timer at 3.3V. The run current is the current required to operate all cores on the MCU. Even though all the modules researched can operate at lower voltages, only 3.3V is considered in the table. I_{tx} and I_{rx} refer to the current required to transmit and receive using the LoRa radio, respectively. The RF characteristics are the maximum possible values, and the conditions, such as spreading factor (SF) and bandwidth (BW), may vary between modules. Lastly, V_{in} is the voltage range accepted by the module where the module's cores are still operable. By using a lower voltage, the device can operate at a lower power, at the expense of reduced RF performance, or the inoperability of a peripheral on the MCU. In some cases, the run current is either the current in TX or RX modes (LoRa transceiver can't be turned off), indicated by "RX|TX".

Table 10: LoRa Module Electrical Characteristics Comparison

Module	TX (dBm)	RX (dBm)	$I_{sleep}(\mu A)$	I_{run}(mA)	I_{tx} (mA)	I_{rx} (mA)	V_{in} (V)
Seeed LoRa-E5	+22.0	-136	0.360	3.45	107	4.82	1.8 - 3.3
Midatronics Windy	+22.0	-148	0.360	3.45	120	12	1.8 - 3.3
Microchip WLR089U0	+18.6	-136	1.61	RX|TX	115	12.6	1.8 - 3.3
RAK 4600	+20.0	-148	11.2	RX|TX	125	17.0	2.0 - 3.3
Onethinx Core	+20.0	-137	1.66	3.67	118	4.20	1.8 - 3.3

Table 11 shows the I/O capabilities of the modules researched. An integrated LoRa antenna connector would be ideal, but if the performance of the module is good, then it doesn't matter as much. Also, it was a requirement that the module have at least one UART, SPI, and I^2C port. Note that while the MCUs and transceivers used may have a set of I/O capabilities, they may not be exposed to use on the module. Also, the I/O capabilities tabulated are not mutually exclusive. Meaning, that if one of the UART ports are enabled, then there may be less GPIOs available, or even SPI, for example, if they are mux-ed on the same port. However, in most cases, using UART, SPI, or I^2C only removes 2-4 GPIO pins. For the Microchip module, it has 4 general I/O ports that can configured to be either of the aforementioned communication protocols.

Table 11: LoRa Module I/O Comparison

Module	LoRa antenna	BLE?	USB?	ADCs	SPI	I^2C	UART	GPIO
Seeed LoRa-E5	External	×	×	12-bit, 1ch	1	1	3	10
Midatronics Windy	U.FL	×	×	12-bit, 1ch	1	1	1	23
Microchip WLR089U0	U.FL	×	✓	12-bit, 7ch	4	4	4	27
RAK 4600	U.FL	✓	×	None	0	1	1	10
Onethinx Core	PCB trace	✓	×	12-bit, 1ch	1	1	1	8

5.2.7 The Chosen LoRa Module: The Seeed LoRa-E5

The chosen LoRa module is the LoRa-E5, as highlighted in the tables above. Feature and cost wise, the Onethinx and the RAK4600 is a clear winner. However, the RAK4600's power efficiency, specifically its sleep current, is lacking when compared to the Seeed LoRa-E5 and Midatronics Windy. Since the sensor node is going to be spending most of its time sleeping, having the lowest possible sleep/standby current is going to be one of the most important factors affecting the battery life. Not only that, but the documentation is spotty and inconsistent. Lastly, the RAK4600 does not come with a LoRaWAN stack, unlike every other module. Writing a LoRaWAN stack from scratch wouldn't be a huge hurdle, but since writing a LoRaWAN stack isn't the main goal of our project and we would rather use already well-tested code, we opted to not use the RAK4600. The Onethinx then would be the next contender, but the development boards are costly, and the individual modules are not readily available. In order to purchase them, we would have to email them and negotiate a price assuming they would sell in super low quantities (<10). We have reached out on many occasions, with no response even after two full weeks. Due to not having any confidence that we can easily source the Onethinx Core module, we opted out of using it.

Next, we considered the Microchip WLR089U0. This module has the most flexibility when it comes to its I/O. It is unmatched in this area when compared to the modules tabulated above. Almost everything is great about it, like the integrated U.FL connector and embedded USB peripheral hardware. However, there is a limitation when it comes to power management: the LoRa module inside of it can only

be configured in receive or transmit mode and cannot be turned off. This, and also the relatively high sleep current (that still has an operational clock), is what turned us away this module.

The Seeed LoRa-E5 and the Midatronics Windy are essentially the same exact module with the only difference being that the Midatronics Windy has an integrated U.FL connector. It uses almost the exact same LoRa transceiver/MCU IC as the Seeed LoRa-E5, but is double the price for the modules, and triple the price for the development boards. Both of these devices have the best sleep currents and transceiver performance. They are able to go into a standby mode with a running RTC that is still able to wake up the CPU - all while operating at 360 nA. One downside of using the Seeed LoRa-E5 is that it does not have an integrated antenna connector. However, all of the RF front-end is embedded inside the module, and all that is needed outside of it is a very short trace connecting to a U.FL or SMA connector. Another added benefit is that Seeed and STM worked closely together to create the module, so it can be assumed that the module will perform as intended.

5.3 Lithium-ion Solar Charge Controller

Lithium-ion solar charge controllers are current and voltage regulators that are used in stand-alone solar photo voltaic systems. The solar panel takes in power from PV arrays and the charger delivers optimal power to the electrical load. Due to the overall desired goals of our project it has been decided that our design will incorporate a Li-Io solar charge controller IC to regulate the charging of the battery in the system. The heart of our solar power design behind the power system is the stand-alone Li-Ion battery charge management controller. This chip takes in a certain voltage range and can be manipulated to output the desired voltage without drawing too much current from the solar panel.

The power considerations that must be taken into account are the voltage of the micro-controller to be powered, our various sensors, as well as the solar power output. The selected micro-controller and two of our three sensors has an input voltage range of 1.8V to 3.6V with peak performance coming from an input of 3.3V. The selected solar panel has a peak performance output of 6.12V. Finally our PM sensor has an input of 5v which creates the needs of a converter capable of transforming 3.3v to 5v. This means that the desired charge controller for our system must be able to take in a range of voltages around 4V-6V and output a voltage of 3.3v. The charge controller is be responsible for delivering power to the system load and only draw from the Li-Io battery when more power is needed. By prioritizing available power input over the Li-Io battery, the charge controller can efficiently power and charge the system while maximize battery life. The other major factor considered in choosing a battery charge controller was its ability to have dual-input power modes. This is because the power design is centered around not only the ability to power and charge the system through a solar panel but also charge and power the system through a USB input. [53]

5.3.1 BQ24168 Charge Controller

The first of two Li-Io chargers considered was Texas Instrument's BQ2416xx 2.5A, dual-input, single-cell switched-mode Li-Ion battery charger with power path management and an I^2C interface. The needs within our circuit design call for several things. First it must take in voltage from a solar panel at various levels and convert them into a stable DC voltage. Second it must take in power from a USB source to supplement the needs of the load and battery in cases where a solar panel won't be in use, for instance, during initial charging, if the device will not have sustainable output, or if the panel is damaged/broken.

The final major concern is that the IC be equipped with safety protocols that protect and manage battery output and system power management. The BQ2416xx IC solves the issue of needing a dual input charger. It has pins dedicated to taking in both USB and our solar power source taking in up to 6.5V on the USB side which exceeds the projected 5V input of the USB. It can also take in 10.5V of an alternative source which fits with the range of the 6V projected input of our selected solar panel. This device also has in place protections used to monitor battery charging current and decreases its input current when the load is requiring more operational current. It has parameters set in place so if the battery were to be defective or absent the system would be able to solely power the load itself so long as it's being given power by one of the sources. It can also seamlessly allow the battery to become the source if needed to power the load.

Using I^2C it can be programmed and uploaded with charge perimeters allowing the team to be able adjust its operation and exercise its complete control over the device. Below is a figure showing the default timer flowchart used in the case of no parameters uploaded over I^2C. In regular intervals independently the device checks the levels of the battery to see if its charge is completed in order to prevent overcharge. So overall this chip provides the user the capability to take in power from the USB and solar panel seamlessly, outputs power to the load and battery independently while it can power the load and offers protection to the lithium-ion battery at a price range of $5-10. Another feature of the BQ24168xx is the watchdog timer that monitors the safety of the charging system. Once a read/write operation is performed on the I^2C interface, a 30 second timer is initiated. The 30 second timer is reset by the host using the I^2C interface. This can be done by writing a 1 to the reset bit (TMRRST) in the control register. The TMRRST bit is then automatically set to "0" when the 30 second timer expires. This process is reiterated until the battery is fully charged or the safety timer expires. The flow chart can be found in Figure 8.

5.3.2 MCP73871 Charge Controller

The second of the two Li-Io charge controllers researched was the MCP73871 stand-alone system load sharing and Li-Ion battery charge management controller. The MCP73871 chip is equipped with both AC-DC wall adapter and USB port power source selections, satisfying the design requirement for having two main sources avail-

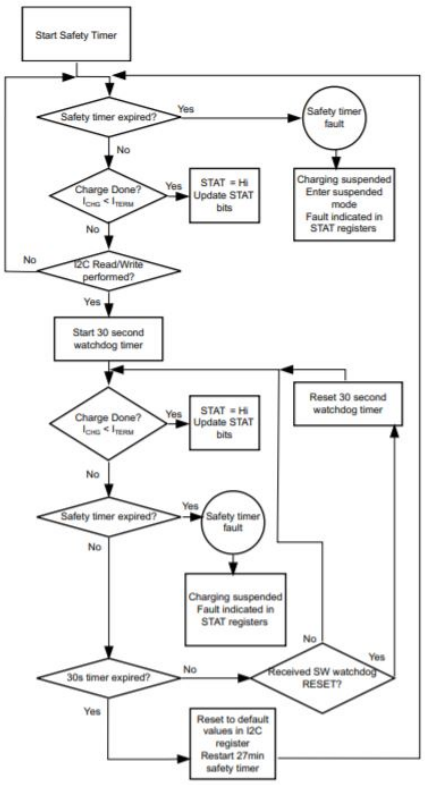


Figure 8: Watchdog Timer Flowchart for BQ24168

able to power and charge the system. One of the most attractive features of the MCP73871 charge controller was its autonomous system load sharing feature. This allows the Aether node to be fully operational while simultaneously charging the Li-Io battery. Additionally, the charger automatically uses available input power to meet the system load demands, directly taking the input voltage to the output pin. In the situation where input power, whether it be solar or USB, does not meet the load demands, the charger draws the remaining current demand from the battery up to 1.8A. A Voltage Proportional Charge Controller or VPCC pin on the IC is used to prioritize the system load demands over the the Li-Io battery charge. This means the MCP73871 is constantly powering the system while the battery is being charged and discharged. A key improvement to the design of this solar/USB charger is a pass transistor inside the chip connected to the load output. This transistor prevents the loss of efficiency from charging and discharging the Li-Io battery.In 9 the interior design of MCP73871's load sharing is shown in more detail.

The MCP73871 comes with a constant voltage regulation feature that is fixed with four available options depending on the charge requirements of the Li-Io battery: 4.10V, 4.20V, 4.35V or 4.40V. In conjunction with a thermistor, the charge controller is also able to regulate charge current depending on the temperature of the battery to ensure reliability and optimization of the system. In addition to these features, this solar/USB charge controller comes equipped with three battery status pin indi-

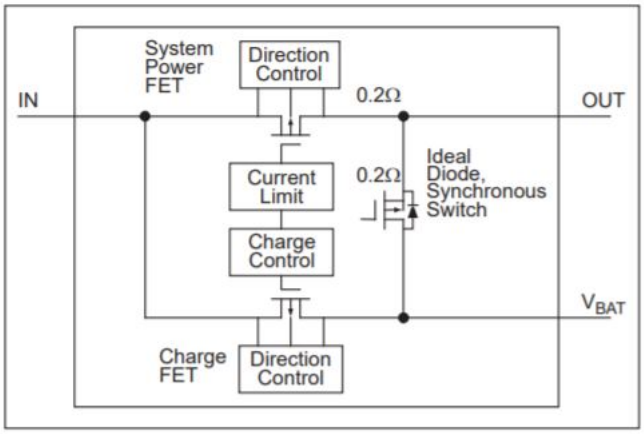


Figure 9: Load Sharing Design of the MCP73871.

cating a low battery state, charging done state, and a power on state. LED's can be connected to these pins to indicate to the user that status of the system and they can simultaneously be configured to a micro-controller to relay the information back to the server.The charger is also equipped with a resistor programmable charge mode ranging from 50mA to 1A. Overall, the MCP73871 charge controller is an extremely versatile IC with the built in capabilities meeting the demands of the power system design for the Aether nodal network.

5.3.3 Charge Controller Chosen for Power System

Microchip's MCP73871 charge controller was ultimately chosen as the best option for the solar/USB powered Aether node sensor network system. In the end, the decision for which solar/USB charge controller would be used in the power system ultimately came down to the two charge controllers discussed above. Both met the requirements of having dual-input power lines for operating the system with both a solar panel or a USB input. Additionally, both chargers came with built in circuit protection features and timers to make sure the Li-Io battery was not charged under dangerous conditions. The BQ2416xx and MCP73871 charge controllers both implemented a load sharing design where the system can be powered directly from the input voltage while simultaneously charging the battery. Although these charge controllers were similar in price and other features, there were some differences which led to choosing the MCP73871 charge controller over Texas Instrument's BQ2416xx IC.

The first difference that leaned the decision towards our chosen charge controller was the operating input voltage range. MCP73871 required a minimum of 0.3 V to operate with a maximum input of 6 V. On the other hand, the BQ2416xx charge controller needed a minimum of 4.2V input to properly operate with a max of 6 V. Since the input voltage line would be directly connected to the solar panel output, the voltage output could not be guaranteed to always provide 4.2 V to the charge controller. By using the MCP73871 chip, it gave a certain degree of flexibility to the

Table 12: Comparison between two candidate solar panel charge controller

Part Name	MCP73871	BQ2416xx
Price ($)	3	3
IN Operating Voltage Range (V)	$0.3 - 6$	$4.2 - 18$
I_{SS} Max Charge Range (mA)	$2.5 - 3.75$	$0 - 2.5$
Leakage Current (μA)	1	5
Fast Charge Rate (mA)	$100 - 500$	$550 - 2500$
Charge Voltage Range (V)	4.10/4.2/4.35/4.4	$3.5 - 4.44$

required output of the solar panel. If the solar panel's input voltage was not enough to meet load demand the chip would still be operational and draw current from the Li-Io battery. However, if the solar panel was not outputting a minimum of 4.2V, the chip would not properly operate and the system may only be powered through USB input. Another factor contributing to the decision was the charge controllers charge rate range. The MCP73871 charge controller can output a low fast charge rate of 100mA and can be controlled through resistors. On the other hand, the BQ2416xx charge controller had a fast charge rate ranging from 550 mA to 2500 mA, given more than what is needed and not being able to provide a lower fast charge rate for low power mode system operation. One feature Microchip's IC had over Texas Instruments was the battery status lines that would play a crucial role in monitoring battery status remotely and on the field. From the table comparing both choices, it can also be seen that the leakage current for the BQ2416xx chip was slightly higher than that of the MCP73871. Microchip's MCP73871 was determined to be the best choice for a solar/USB charge controller given its added features and flexibility along with the ease of integration to the system. We provide a comparison between these two components in Table 12.

5.4 Batteries

In this section, we provide an overview of the different types of batteries that we considered for use in our design. These types include lead acid batteries and lithium batteries. We also cover the different characteristics that we considered during the process of choosing a battery type.

5.4.1 Lead Acid Batteries

Lead batteries consist of two electrodes, a negative electrode consisting of lead and a positive electrode consisting of a lead oxide. The two electrodes are submerged in an electrolyte solution comprised of a mixture of water and sulfuric acid. Typically, an electrically insulating material that is chemically permeable is added to ensure the two electrodes do not come into direct contact. The configuration of a lead acid battery can be seen in Figure 10, while the chemical equation of the reaction being

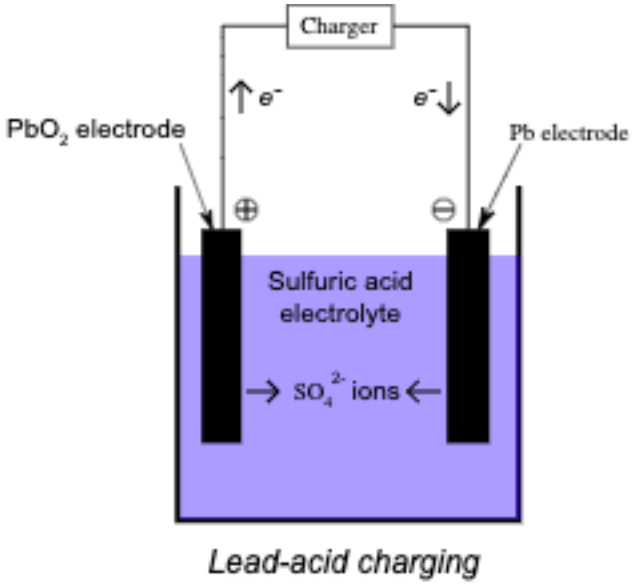


Figure 10: The configuration of a lead acid battery. Used with permission under the CC BY 3.0 license [33].

implemented can be found in Equation 3.

$$PbO_2 + Pb + 2H_2SO_4 \underset{charge}{\overset{discharge}{\Longleftrightarrow}} 2PbSO_4 + H_2O \tag{3}$$

5.4.2 Lithium Batteries

Lithium batteries differ in a few substantial ways from their lead acid counterparts. The basic component of a lithium battery is a positively charged cathode typically made of a lithium oxide metal that will give off lithium ions. A negatively charged anode that while charging, will store lithium ions from the cathode and while discharging will allow the lithium ions flow through the electrolyte and the electrons through the electrical path back to the cathode. The Electrolyte is usually a material that is highly ionic conductive, this will permit lithium ions to pass through while the electrons will be kept in the anode. The last material is called the separator and it usually consists of polyethylene or polypropylene. The separators' job is to not inhibit the other functions of the battery but also keep the anode and cathode from coming into physical contact. The configuration of a lithium-ion battery can be found in Figure 11 [13].

5.4.3 Battery Characteristics

To more thoroughly understand which battery is the right choice for a project one must look at how these methods of constructing a battery will affect the overall characteristics of the battery. The characteristics that impact the usefulness of a

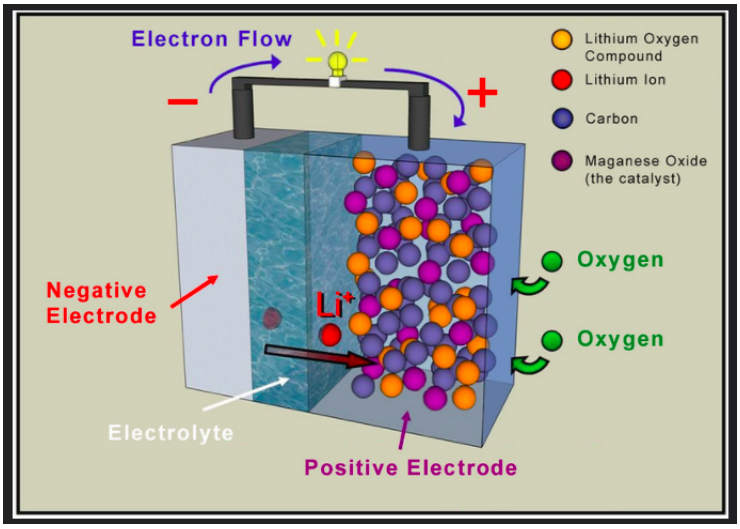


Figure 11: The configuration of a lithium ion battery. Used with permission under the CC BY 3.0 license [34].

battery are as follows: discharge curve, energy density, temperature dependence, service life, charge/discharge cycle, and cost. Once we have amassed information around each characteristic we can then compare Lead acid to lithium batteries more thoroughly. From there if there is not an overall better choice in each and every category we will decide based on which characteristic is most important to complete the objectives of the project. For instance if it is determined that lithium would put us out of our needed price range but is only marginally better than lead acid in other categories then our overall decision will be impacted. Much like any well thought out purchase, we must look at each characteristic to gain a better understanding of how our project is being designed and where money is being allocated to.

Discharge Curve A discharge curve is a graph that plots Voltage against percentage of the capacity discharged. Essentially the Output voltage of a battery will lower as the battery is used up. Given this effect the most desirable curve would be flat (keeping voltage consistent at every level of discharge), so let us look at the average discharge curve of our lead acid and lithium-ion batteries. Figure 12 shows example discharge curves one might find in a lithium ion and lead acid batteries. The figure has been provided by Power Tech, an energy storage system manufacturer. This figure shows first hand how steadily lead acid batteries drop in voltage as the depth of discharge increases. Meanwhile lithium is kept at a steady for the most part until the depth of discharge reaches upwards of 80 percent. [10]

Energy Density Energy density is the amount of energy you can get out of a battery per unit volume of weight required. Clearly the higher the density the better for a battery. The characteristic known as specific energy density is closely related. However, it considers the Discharge curve of the battery and how it would affect the voltage and current during the battery discharge cycle and is solely measured

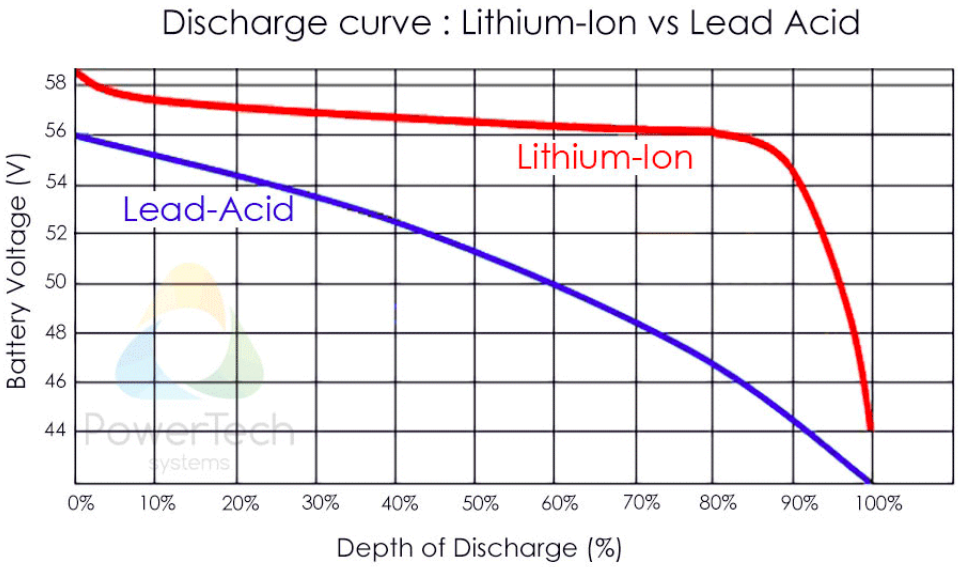


Figure 12: A comparison of a lithium ion battery discharge curve and a lead acid battery discharge curve. Used with permission under the CC BY-NC 3.0 license [36].

against weight. An example of the range of energy density and specific energy density can be found in Figure 13 and is provided by the National Aeronautics and Space Administration. This figure shows that lithium far surpasses lead acid in energy density. When it comes to density by volume and density by weight lithium is roughly three times superior to lead acid. [16]

Temperature Dependence Battery performance and temperature are highly correlated. Rates of reaction within battery cells themselves are temperature dependent as well as the internal resistance. Low temperatures give higher resistances in a battery, could freeze the electrolyte giving a lower voltage, and cause a steeper discharge curve. At higher temperatures chemicals may decompose or cause enough energy to become available to activate unintended reactions reducing the capacity. The temperature ranges for charging and discharging the viewed battery types are shower in Table 13. This information came from Battery University(TM) an online free website teaching hands on battery information. This tables shows that the temperature ranges during charge and discharge are relatively similar with the only difference coming in charging advisories. As one can see lead acid is suited better below freezing during charging mean while lithium performs better at higher temperatures at the cost of its life span.

Service Life As each recharge cycle of a rechargeable battery takes place its active components are slowly depleted, lowering its capacity. The industry service life of rechargeable batteries is defined as when the battery's capacity is 80% of its intended use. The typical service life of rechargeable batteries could be anywhere from 500-1200 cycles. Figure 14 shows how repeated cycles influence the types of batteries

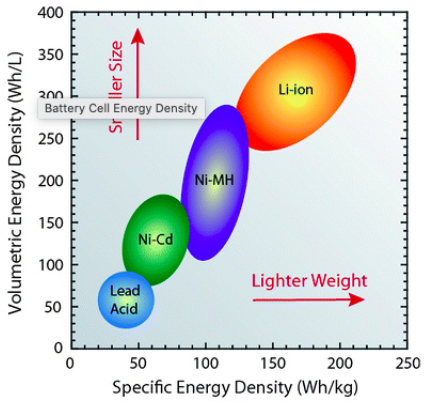


Figure 13: A comparison between volume energy density and specific energy density for various battery types. Used with permission under the Creative Commons Attribution 4.0 International license [48].

Table 13: Temperature Dependence

Battery Type	Charge Temperature	Discharge Temperature	Charge Advisory
Lead Acid	-20 C to 50 C	-20 C to 50 C	Charge at 0.3 C or less below freezing Lower V-threshold by 3 mV/C when hot
Li-ion	0 C to 45 C	-20 C to 60 C	No charge recommended below freezing. Good charge/discharge performance at higher temperature but shorter life.

were researching with the lead acid curve on the left and lithium ion on the right. As seen within the figure lithium batteries could go through 2-3 times the number of cycles as lithium batteries with a similar depth of discharge available. Meaning the the Lithium Ion batteries are capable of going through hundreds more charge and discharge cycles before reaching the same industry standard state that is end of life. [35]

Charge Curve The charge curve shows the process of charging different batteries. This would include information on the voltage needed to charge, current used to charge, charging times, and capacity of battery. Figure 16 shows the charge curve for an example lead acid batteries provided by power sonic a battery manufacturer, and figure 15 shows the charge curve for an example lithium-ion battery from Battery University(TM) an online free website teaching hands on battery information. While the charging curves will differ from battery to battery meaning the examples are not completely accurate as to what can be expected the overall structure of the curve will match those with similar chemistry.So for instance while our battery will be smaller since all the proportions we will be using are equally small we can expect a similar outcome. What we can see reliably however are the charging time for lithium batteries are far lower than their lead acid counter parts.

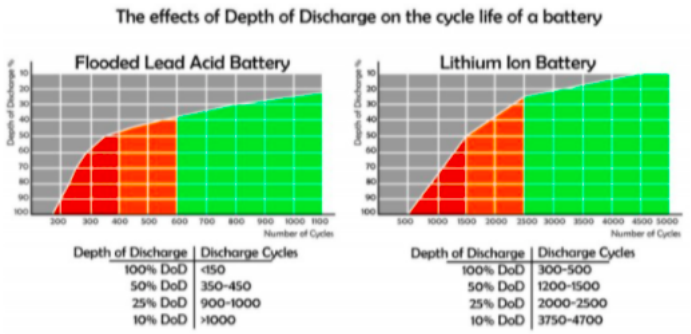


Figure 14: A comparison between the service life of lead acid and lithium ion batteries. Used with permission from [15].

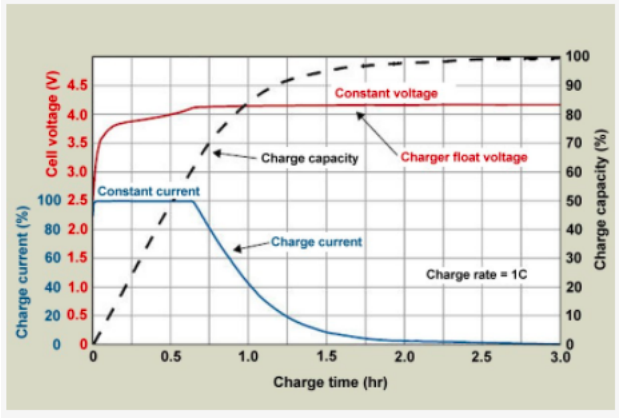


Figure 15: The charge curve for a lithium ion battery. Used with permission from [26].

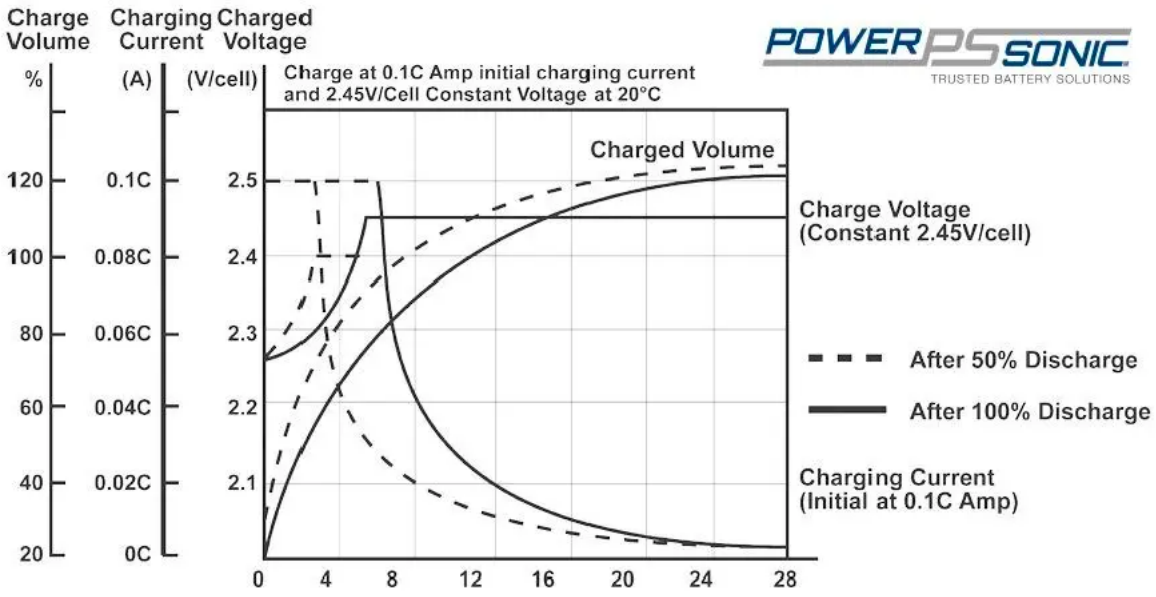


Figure 16: The charge curve for a lead acid battery. Used with permission from [29].

Table 14: Initial cost per capacity and cost per life cycle comparison between lead acid and lithium ion batteries.

	Flooded Lead Acid	Lithium-Ion
Initial Cost per Capacity (USD/kWh)	131	530
Cost per Life Cycle (USD/kWh)	0.17	0.19

Cost The cost category is self-explanatory. When it comes down to the devices components certain materials can be acquired for a much lower cost than others. For example, according to the US government in 2018 lithium costs $13,000 per metric ton, while on the other hand during the same year lead hit an all-time high with $2,600 per metric ton. While there are other factors such as manufacturing cost based on the age of the industry and scale the most hefty cost comes from materials which we can see first hand. Table (b) is a handy chart to estimate the cost one could expect to pay for the batteries we have researched. Information within the table comes from "Off Grid Solar: a handbook for Photovoltaics with Lead-Acid and Lithium-Ion batteries" by Joseph P. O'Conner. It is complete with both upfront costs and projected costs per life cycle. As can be seen both the upfront cost and per life cycle cost of lead acid is lower. This is because even though lithium ion will last many more life cycles the cost accrued from using it is also much higher.

Throughout this section, we have researched and explored the different characteristics that impact how a specific type of battery functions. Here we will provide a brief summary of the comparison of the battery characteristics between lithium ion batteries and lead acid batteries. The overall discharge curve of lithium batteries is much flatter than lead acid, indicating that they give a constant voltage as they discharge. In terms of materials, lithium ion batteries have a much higher energy density than lead acid batteries meaning that the batteries will be smaller, weigh less, and have a higher capacity. In terms of temperature charge, lead acid batteries have a higher temperature range and vice versa for discharge and lithium ion batteries. The charging advisory for lithium ion batteries is higher and not supposed to occur below freezing, which can be an issue in colder climates. We can see the depth of discharge has a much steeper curve for lead acid batteries. This tells us that lithium ion batteries can go through more charge cycles before becoming reaching the 80% capacity threshold. The only large difference in the charge curve between lead acid batteries and lithium ion batteries is that lithium ion batteries have a much shorter charge time. Finally, the comparison between initial cost per capacity and cost per life cycle between both battery types can be observed in Table 14. The operational cost per cycle is slightly higher and the upfront cost is much higher for lithium ion batteries. Due to these factors we have decided to use a lithium ion battery in our design.

Table 15: Differences between Lead Acid and Lithium Ion Batteries

Characteristic	Findings
Discharge Curve	The overall discharge curve of lithium was much flatter showing that they give a constant voltage as they discharge.
Energy Density	In terms of materials lithium has a much higher energy density than lead meaning that the batteries will be smaller, weigh less, and have a higher capacity.
Temperature Dependence	When it's come to temperature charge lead acid has a higher temperature range and vice versa for discharge and lithium. Charging advisory for lithium is higher and not supposed to occur below freezing which can be a factor in colder climates.
Service Life	As we can see the depth of discharge has a much steeper curve for lead acid. This tells us that lithium can go through more charge cycles before becoming reaching the 80% capacity threshold.
Charge Curve	The only large difference in the charge curve is lithium's much shorter charge time.
Cost	The cost of lead acid and lithium are clear and square from the table. The operational cost per cycle is slightly higher and the upfront cost is much higher.

5.4.4 Summary

Throughout this section we have noticed many large and small differences between our possible selections. These have been complied and are presented as such in Table 15. You can plainly see when compared lithium ion out performs lead acid in terms of quality however in terms of cost lead acid clearly leads. As a result our project must choose if lithium acids' quality exceeds the increase in price tag and it has been decided that it has. As a result of sing a lithium ion battery our project will be better equipped in colder climates, last longer, and give a more constant output during use. While the increase in cost in notable we believe this decision will turn out to be an asset when it comes to concerns of long term power supply for our nodes.

Decision The chosen battery is the LP-103454 an image of such can be found in 17. This is a 3 cell battery with a voltage of 3.7V and 2AH. This means we can expect 7.4WH of capacity allowing our device to run for days without need for external power. This battery will be a major asset when it comes to concerns of keeping our system supplied with power in more extreme conditions over longer periods of time. A link to the data sheet is as follows: https://www.soselectronic.com/products/eemb/lp103454-1-157393

5.5 Solar Panel

In this section, we will discuss the different types of solar panels that are being considered for use in our design. Discovered in the early 19th century solar panels utilize the photo voltaic effect. This phenomenon was observed when certain materials

Figure 17: Our selection of a lithium ion battery. Used with permission under the Creative Commons Attribution 4.0 International license [45].

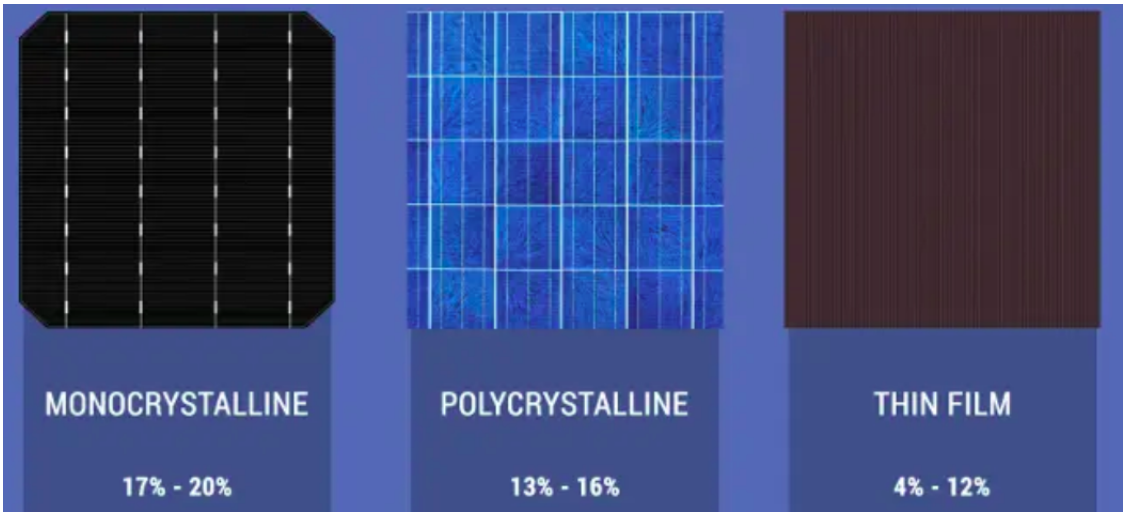


Figure 18: The three types of solar panels. Used with permission from [55].

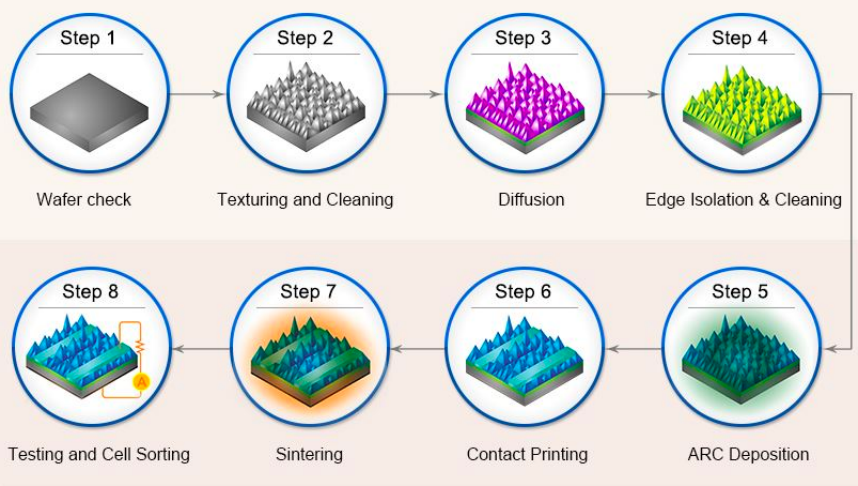


Figure 19: Solar panel wafer manufacturing levels. Used with permission from [66].

were produced and electric current when exposed to light. The way this phenomenon is utilized by using two semiconducting materials. One layer must have depleted electrons and when exposed to sunlight some photons are absorbed by the semiconductor which excites the electrons causing them to jump from one semiconductor layer to another. This jump produces a small electric current which when happens in mass makes a usable power source. The most common material to use in solar panels is silicon which is cut and polished into wafers.Figure 19 shows the different steps taken into account to manufacture these wafers into the solar cells that are so recognizable today. Some of these wafers are doped to make an electrical imbalance further helping the process. Finally, electrically conductive strips are attached to the cells to absorb the generated current.

Additionally we should discuss the method at which we measure the efficiency of solar panels. Equation 4 shows the mathematical equation used for measuring solar panel efficiency. In this equation you can see a break down of some of the characteristics of some of the solar panels. For instance the incident radiation flux is the proportion at which power can be radiated through an area. For this think about how mono-crystalline allows for more power to flow through because of the single cell silicon structure the wafers consist of. Second one should look at the area of the collector. This describes the incident face of the solar panel, that being the surface area that collects power from the sunlight. This on our design is limited due to the size of the node in question but should the other 2 factors decrease proportionally thee will be no effect on efficiency. Lastly and most importantly we have the maximum power output in watts. This is the amount of power one can expect the solar panel to generate.

$$\eta_{\max} = \frac{P_{\max}}{E \cdot A_C} \times 100\% \tag{4}$$

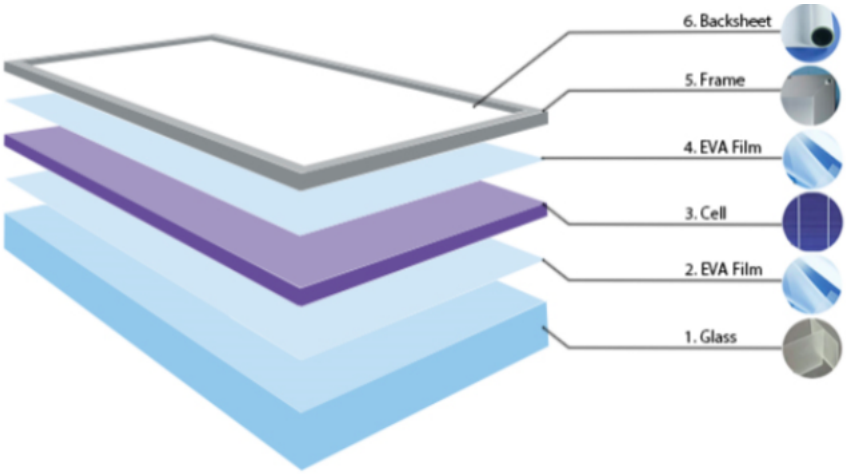


Figure 20: An example of a monocrystalline solar panel. Used with permission from [11].

$$P_{\max} = \text{maximum power output (W)}$$
$$E = \text{incident radiation flux } (\frac{W}{m^2})$$
$$A_C = \text{area of collector } (m^2)$$

5.5.1 Monocrystalline Solar Panels

Like the name suggests monocrystalline solar panel cells are made from a single crystal of silicon. Having a single crystal gives electrons more room to flow providing greater efficiency providing more power to the load which is the largest advantage of this type of panel. This, however, is more difficult to create, making it the more expensive option. This structure also provides more durability enabling the panel to last for years longer than its alternatives. An example of the features one can expect from a monocrystalline panel can be seen in Figure 20. Here one can see the EVA films on either side of the silicon wafers. The wafers itself made up of a single bar of silicon giving the solar panel its name monocrystalline. This figure was provided by Alternative Energy Solutions, as alternative energy information source.

5.5.2 Polycrystalline Solar Panels

Unlike monocrystalline panels these solar panel cells are made from many crystals of silicon. This offers much less freedom of movement for electrons which gives a much higher efficiency. This is also simpler to build, meaning that costs are lower, which could be a factor in our decision. This structure also provides not as much durability as its monocrystalline counterpart however it does last longer than its thin film competition. An example of the features one can expect from a polycrystalline panel can be seen in Figure 21. Here one can see the EVA films on either side of

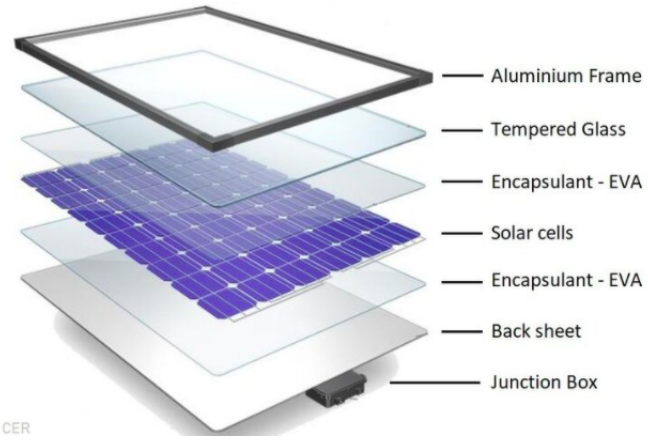


Figure 21: Polycrystalline solar panel. Used with permission under the CC BY-NC 3.0 license [54].

the silicon wafers. The wafers itself made up of silicon fragments melted together to produce individual wafers giving the solar panel its name polycrystalline. This figure was provided by Clean Energy Reviews an independent energy source review site. [43]

5.5.3 Thin Film Solar Panels

Unlike the previous two types of solar panels, thin film solar panels are mostly not made of silicon. They are made of a mixture of materials mostly cadmium telluride, which is comprised into a thin sheet between two transparent conductive layers that help capture sunlight. Thin film solar panels usually have the lowest efficiency and durability of all solar panel types however they make up for it in flexibility and where it can be comfortably placed . The main advantage of this type of solar panel is their low weight, profile, and its ability to adhere to whatever required surface it may need to. Figure 22 shows an example of thin film solar panels. This figure was provided by Sun Power Source, a solar energy review, news, and project work site. [61]

The selection of solar panels comes down to less factors than the battery. Here the differences between the first two types monocrystalline and polycrystalline are very straight forward. Monocrystalline will provide more efficiency and durable but polycrystalline will be more affordable. However when thin film solar panel are introduced we must consider different factors such as placement of the solar panel. Show we provide the solar panel with a jagged or uneven surface to be placed on then extra precautions would be needed to have one of the first two panels making thin film more convenient and reliable. How ever due to the nature of our project and its intended use we can comfortably provide our solar panel with a reliable and safe solar panel placement leading once again if the increase in efficiency is worth the increase in cost, the increase in cost being roughly 20 percent on average given the typical cost. Seeing as on average monocrystalline is between 6 and 53 percent more efficient with

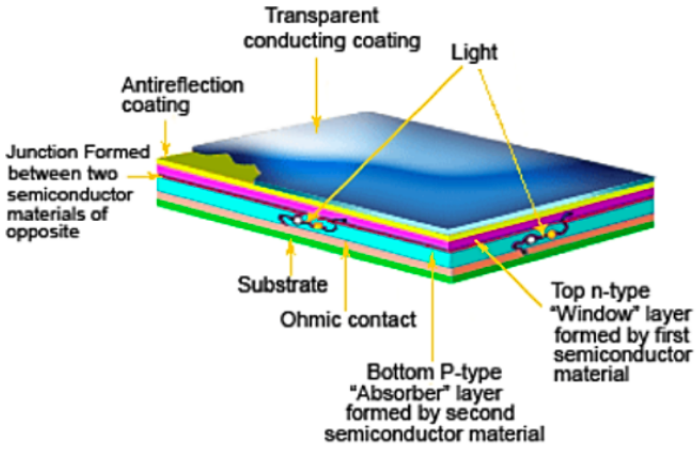


Figure 22: An example of a thin film solar panel. Used with permission from [62].

Table 16: Differences between solar panel sub-types

Solar Panel Type	Efficiency (%)	Price Per Watt (USD/W)	Projected Life Span (years)
Monocrystalline	17 – 20	1.00 – 1.50	25 – 35
Polycrystalline	13 – 16	0.70 – 1.00	23 – 27
Thin Film	4 – 12	1.00 – 1.50	14 – 17

the average being 20 percent more efficient than polycrystalline our team has decided that the increase in cost is justified. As a result the solar panel we have decided upon is the P105, a monocrystalline 5.5W 6V solar panel. This should provide our system with a reliable and powerful source of power capable of keeping the system independently online. Seeing as the capacity of the battery itself is 22.2WH and the daily power consumption has been calculated as 0.045WH we have implemented a system where solar power could be lacking or non-existent for weeks and the panel could fully power the system and charge its battery to full over the course of a single day in prime conditions.

5.5.4 Solar Panel Chosen for Nodes

The decision of which solar panel to use ultimately only came down to very few factors and was trivial. Output voltage was the only main requirement that needed to be met. The charge controllers for Li-Io batteries operate best with 6V solar panels. Both of the Voltaic solar panels considered were 6V panels and were similar in dimensions. The final decision came down to cost and wattage. Voltaic's P105 was chosen for this design due to its lower cost and higher wattage output. When compared to the other option, this solar panel was higher in wattage by 2W and $10 cheaper, contributing to a lower overall cost. The maximum voltage output was 6.2V versus 6.7V, also making it a safer choice to use on the battery charge controller used in the design. The comparison between these two components can be found in Table

Table 17: Comparison between two candidate solar panels

Part Name	Voltaic Systems P105	Voltaic Systems 3.5W 6V
Type	Monocrystalline	Monocrystalline
Dimensions	8.740" L x 5.394" W x 0.157" H	8.3"L x 4.4"W x 0.2"H
Price ($)	35	45
Max Power (W)	5.75	3.69
I_p/I_{sc} (mA)	940/990	550/590
V_p/V_{oc} (V)	6.12/7.13	6.7/7.7

17.

5.6 USB Power

USB charging has become a wide spread standard for powering and charging all different kinds of systems today. Which is why it represents a great asset in our design. With how common the hardware for this method is, it would be an oversight to not incorporate it. There are a total of 6 USB specifications, USB 1.0, 2.0, 3.0, 3.1, 3.2, and USB4. To charge and power the system, USB-C will be used in the design, which incorporates USB4 technology. In our design we wish to plan for a scenario in which a field crew could come and charge the battery quickly and keep basic nodal functions operational without the need to wait for a solar panel. This functionality could also come in handy if certain equipment were to malfunction discontinuing the capabilities of the node to recharge itself.

This design incorporates an IC that receives power from the solar panel and a USB charging connection. The circuit would take power from the solar panel and feed it to the load and battery, unless the input selection pin were instructed to take power from the USB connection and use that to power the load and charge the battery. This way the battery could take in power from either only the solar panel or USB connection. The USB-C interface will also be used as a communication link to the Lora-E5 micro controller. Since the micro controller cannot communicate directly via USB, a USB to UART communication IC will be needed so that USB can be used as a means of communicating with the Lora-E5. [63]

5.7 Air Pollution Monitoring Sensors

Over the last century world-wide energy consumption has increased at a rapid rate due to industrialization, economic and population growth. As a result, the world has also seen an evident spike in the levels of polluting gases emitted in the air which can have detrimental effects to the environment and humans. The burning of fossil fuels for energy consumption leads to the emissions of Carbon Dioxide (CO2) and other harmful gases such as Carbon Monoxide (CO), Nitrogen Oxides (NO_x), Sulfur

Dioxide (SO2), and Particulate Matters (PM2.5 and PM10) all of which are harmful toxins. Air pollution monitoring systems have been developed to track these emissions and ensure air quality and enforce environmental standards.These systems are based on sophisticated equipment that measure air quality and can detect gases and matters using a variety of methods. In this section and the following subsections, the different types of sensors that are available and the methods they use to track pollutants are discussed. The sensors that were considered and chosen for the Aether air pollution sensor network are also discussed here. These sensors were chosen based on several factors including cost, accuracy, integration, and power consumption. Table 23 shows the comparison of these type of sensors.

5.7.1 Environmental Gas Sensor

There are mainly 5 different types of gas sensors which use different methods to detect gases, each with their own advantages and disadvantages. The most widely used gas sensors today are electrochemical sensors, catalytic sensors, solid-state semiconductor sensors, non-dispersive infrared (NDIR) and photo-ionization detector (PID) sensors. Due to the constant interaction of different gasses in the air, individual gas sensors have to be calibrated more than other kinds of sensors. Sensors will need to be exposed to a pre-determined pollutant with a specified concentration so the difference between gas concentration and sensor reading is minimized. Although portable and low cost, these sensors cannot achieve the level of accuracy as other industrial grade measuring systems but can however be accurate enough to detect dangerous levels of toxic and polluting gases used to calculate AQI levels. Four gases are the primary contributors to poor air quality and are the ones mainly used in calculating the AQI index, CO, NO2, O3, and SO2. The type of gas sensors best fitted to detect these gases are listed below.

- O_3: Is best detected by electrochemical and solid-state sensors
- SO_2: Only detectable with electrochemical and solid-state sensors as NDIR sensors aren't sensitive enough and catalytic sensors would be burned with the toxins
- NO_x: Is best detected by electrochemical and solid-state sensors; readings easily interfered by the presence of O3
- CO: Detected by electrochemical and solid-state sensors

In the research conducted for finding the most suitable sensors for Aether, it was determined that electrochemical sensors were the most reasonable in price and power consumption while still providing reliable and accurate readings for measuring air quality in outdoor and indoor uses. The operating mechanisms of each type of gas sensor is discussed below.

Solid-state Gas Sensors The working principle behind solid-state gas sensors was discovered around the same time when research was being conducted on p-n junction devices. Solid-state sensors typically have one or more transition metal oxide with a heating element which regulates the temperature of the sensor. The metal oxide causes the gas particles to dissipate into charged particles resulting in the transfer of electrons. There is a circuit which regulates the heating element to operate at the sensor's specific temperature range where gas can be optimally detected. Electrodes integrated into the metal oxide cause a change in conductivity due to the molecular interaction of gas particles. In turn, this change in conductivity is measured as a signal relating to the level of gas concentration.

Solid-state sensors have appealing characteristics as they provide a high degree of versatility and longevity. Due to the operating mechanisms behind these types of sensors, with the right filtering in place to block out certain gases, solid state devices can be made to detect a over 150 different type of gases. Additionally, another major advantage of solid-state sensors is the lifespan, with an average life expectancy of over 10 years it makes it an ideal candidate for long-term low maintenance use.

Electrochemical Gas Sensors The operating mechanisms behind electrochemical gas sensing are electrochemical reactions, specifically oxidation and reduction reactions. With working electrodes in place, the reaction of gas molecules with ambient oxygen particles generates an output current proportional to the level of gas concentration present. The output current is then converted to a parts per million (PPM) unit. The type of electrodes and membranes used within the circuit are determined by the type of gas the sensor is trying to detect.

Electrochemical sensors have the lowest power consumption of any other kind of gas sensor making it an ideal candidate for use in the Aether sensor network system. The response times for these sensors are usually under 50 seconds and the typical life span ranges anywhere from 2 to 3 years. When compared to solid-state sensors, there is a trade off between cost and accuracy. Although not nearly as accurate as industrial grade solid-state devices, it still generates an accurate enough reading to calculate ambient air quality and gas concentrations at a tenth of the price, with most electrochemical sensors ranging any where from a few dollars to 50 dollars.

Catalytic Gas Sensors Catalytic-type gas sensors consist of a detector element that burns in the presence of combustible gases causing a rise in temperature and thus a rise in resistance. The change in resistance is proportional to the level of combustible gas present. Combustible gases include methane, propane, and hydrogen. Since the gases fall outside of the gases contributing to poor air quality, specific catalytic sensors were not considered for the development of this product design.

Sensor Type	Gases Detected	Linearity	Cross Sensitivity	Power Consumption	Maintenance	Response Type	Life Expectancy
Electro-chemical	Electro-chemically active gases such as Oxygen and other toxic gases (About 20)	Linear at room temperature	Avoided using chemical filtering	Lowest of all	Low	< 50s	2-3 years
Catalytic	Combustable gases	Linear at 400° - 600° C	-	Very High	Sensitivity decreases with time due to burn out	< 15s	< 3 years
Solid-State	~ 150 different gases	Linear at operating temperatures	Minimized with filtering	Large	Low	20s - 90s	> 10 years
Non-Dispersive Infrared	Hydrocarbon gases and carbon dioxide	Non-linear	-	Low	Lowest	< 20 s	3 - 5 years
Photo-ionization	Volatile organic compounds (VOCs)	Linear	Measures any VOCs	Medium	Filter required to keep lense free of debris	< 3s (fastest)	~ 5000 lit hours

Figure 23: A comparison of the different features of gas sensors.

5.7.2 Particulate Matter Sensor

Particulate matters are small particles dispensed in the air defined by its diameter, usually 10 micrometers or 2.5 micrometers (PM10 and PM2.5). The smaller the particle, the easier it is for it to pierce through the human respiratory and cardiovascular system, making it one of the most dangerous toxins to humans. There are several different ways of detecting particulate matter in the atmosphere. All PM sensors can be categorized under two methods. The first being direct measurement providing continuous readings at predetermined intervals and the other is a laboratory technique using gravimetric analysis. In the laboratory method, a sensor is equipped with a filter that collect ambient particles in the air and is then weighed in a lab and the concentration is calculated. This method is much more time consuming and not feasible for real-time air monitoring systems. The most common methods used in PM detection for air quality measurements are discussed below.

Optical Analysis One method of detecting particulate matter is optical analysis. PM sensors use the principle of laser scattering to track the particles present in the air. A light beam is projected towards a chamber while a fan brings in ambient air into the same chamber. The particles present in the air deflect and scatter the projected light beam and a photodiode then measures how much light passes through the chamber and how much is scattered. A microprocessor embedded within the sensor then converts this measurement into ambient particle concentration. The benefit of these type sensors are that they're lightweight, battery-operated and small. Optical analysis particulate matter sensors can be further broken down into 3 optical principles:

- Light Obscuring: The basic operating principle behind this kind of optical analysis for particulate matter is infrared detection. An infrared emitter such as an LED or light bulb emits light on one side of the chamber while on the other end of the chamber there is an infrared detector. When particulate matter passes through the chamber it obscures some of the light emitted, based on the brightness detected by the sensor, the amount of particles in the air can be detected.

Although this method of detection is fast and affordable, it is the least accurate of all the existing methods and cannot distinguish particle diameter (i.e. 2.5 or 10). Air quality can be detected in a binary nature with good or poor, however an exact concentration cannot be given.

- Direct Imaging: A much less common form of detecting particulate matter is through direct imaging with the use of a high-resolution magnifying camera. As particles pass through the chamber the camera records the ambient particles and a computer software analyzes the images for size and count. This is not a method often used in indoor and outdoor pollution monitoring systems as power consumption is much greater when compared to other methods.

- Light Scatting: In this kind of optical analysis, a laser beam is used as the primary light source. When the fan brings ambient particles into the chamber that come into contact with the laser beam, the light is scattered, and a photodetector is able to detect the light scatting. The particle count is proportional to the intensity of the scatting light. This is then converted into particle concentration. Laser diffraction for detecting particulate matter is perhaps the best option for this design. In addition to low power consumption, it provides fast and accurate readings as long as the sensor is properly calibrated to distinguish between the diffraction of different particles.

Beta Attenuation Analysis Another way of detecting particulate matter in the air is through beta attenuation mass monitoring. A sample of the ambient air is heated and dried to extract any moisture in the air and then ran through the system and collected by a filter. The filter is then exposed to rays of beta radiation which particulate matters then absorb. The attenuation of the beta rays is then measured determine particle concentration. If properly calibrated and external factors are controlled beta attenuation is an extremely accurate method of detecting particle mass and concentration. However, these machines are large and expensive and only used in laboratory settings and buildings such as the U.S embassy in Beijing.

5.7.3 VOC and VSC Sensor

Volatile Organic Compounds (VOC) and Volatile Sulfur Compounds (VSC) are discharged human-made air pollutants. As defined by the EPA, VOCs are characterized by their high vapor pressure and low water solubility. These emitted gases are often the result of human made chemicals such as paint, benzene, ethylene glycol, petroleum fuels and other factory produced chemicals and aerosols. Studies have shown concentrations of VOCs and VSCs are typically up to 5 times higher in indoor environments than outdoor environments. VOCs and VSCs have a significant role in the contribution to ozone and particulates in the atmosphere and are considered dangerous pollutants to humans. There are several different types of sensors in use today that can measure VOC and VSC concentrations and they vary depending on the type of gas being measured and the industry in which they are used. Some of the

common methods used in detecting these type of air pollutants are discussed below.

Photoionization Detector (PID) The basic operating principle behind this type of sensor is the use of ultraviolet light to decompose volatile organic compounds into charged ions. Once the particles are broken down a detector then measure the charge of the sampled ionized gas in order to determine concentration. With an average response time of under 3 seconds, this form of detecting VOCs and VSCs is among the fastest. However, power consumption for these devices falls somewhere in the middle when compared to other methods and filtering is required to keep the system free of debris.

Non-dispersive Infrared Sensing (NDIR) With these type of sensors, infrared radiation emitted causes resonance of the sampled gas molecules. Similar to the principle of light scatting, these sensors are equipped with an emitter, a detector, an optical filter, and an electronic module for signal processing. The infrared light passes through a chamber and to a detector, gas particles absorb some of the light and the difference from light emitted to light detected is converted into a gas concentration. NDIR sensors have a higher life-expectancy than electrochemical sensors and are used to detect hydrocarbon pollutants as well as carbon dioxide.

Flame Ionization Detector (FID) This method is widely used in the automotive industry for detecting hydrocarbon emissions as required by government and industry standards. The basic operating principle behind FIDs is ion detection formed from the combustion of organic compounds in a hydrogen flame. A positive and negative electrode are set in place and generate a potential difference, positively charged ions are attracted to the negative plate and when they come in contact with it they produce an electrical current. This current is proportional to the rate of ionization which can then be related to the concentration of the hydrocarbon. FID sensors are large in size with high power consumption. Because of their operating nature, they also cannot be integrated with other type of gas sensor, hence making it non-ideal for use in the Aether sensing network.

5.7.4 Sensors Used In Aether

After sufficient research was done on the different type of sensors that can be used to monitor air quality, the most suitable sensors for the Aether sensing network were selected. These sensors were chosen based on specific characteristics including:

- Size
- Power Consumption
- Integratebility
- Cost

- Gas Sensitivity

There will be 3 main sensors used in this design, a particulate matter (PM2.5) sensor, an ozone/nitrogen dioxide sensor, and a multi-gas VOC/VSC sensor capable of detecting several gases through machine learning (ML) algorithms as well as measuring ambient temperature and humidity. These sensors are discussed in further detail below

Sensirion SPS30 – Particulate Matter Sensor The Sensirion SPS30 sensor is a particulate matter sensor designed for use in air quality monitoring system applications. It's an optical PM sensor using the principle of laser scattering to detect particulate matter concentrations of varying diameters including PM1.0, PM2.5, PM4 and PM10. With an expected lifetime of over 8 years, the SPS30 sensor provides reliable consistent use without the need for cleaning or maintenance, making it an ideal choice for a stationary indoor/outdoor air quality monitoring system such as Aether.

The SPS30 operates off a laser-based scattering principle alongside sophisticated algorithms to provide accurate measurements for different kinds of particulate matters in the atmosphere. It has a mass concentration range of 0 to 1,000 $\mu g/m^3$ with a +/- 10% accuracy. EPA standards list acceptable 24-hr average PM concentrations can reach up to 35 μg/m3 making the sensor range more than capable of detecting dangerous levels of particulate matter. As previously stated, the SPS30 can detect particulate matters of varying diameters including PM1.0, PM2.5, PM4 and PM10. It's lowest limit of detection for PM is 0.3 micrometers with a minimum sampling interval of 1s when on continuous mode and an operating temperature ranging from -10 to 60^{o} C. The sensor dimensions of 40.6 x 40.6 x 12.2 mm3 provide a small ultra-compact package that allow for easy integration onto the Aether sensing network, contributing to a smaller more compact overall design. It's compatible with both UART and I^2C interfaces and requires a constant voltage supply of 4.5-5.5 V. The design outline and table of specification obtained from Sensirion's SPS30 datasheet can be seen below.

Renesas ZMOD4510 – NO_2/O_3 Gas Sensor The Renesas ZMOD4510 is a solid-state gas sensor that utilizes a heating element on a MEMS structure and a MOx resistor to detect gases. An IC regulates the temperature of the heating element and measures the resistance across the MOx resistor which is a function of the accumulation of ions resulting in the interaction of targeted gas molecules and the metal oxides. This change in conductivity measured is then converted to gas concentration in the ambient air. The device can be used in several different indoor and outdoor applications but is ideal for measuring outdoor air quality. It's equipped with 2 different sensor outputs based on the selected operating method, a non-selective ozone and nitrogen dioxide reading or a selective ozone-only reading using the chips ultra-low power mode.

The ZMOD4510 is a 12-pin LGA assembly with dimensions of 3.0 $\times$ 3.0 $\times$ 0.7

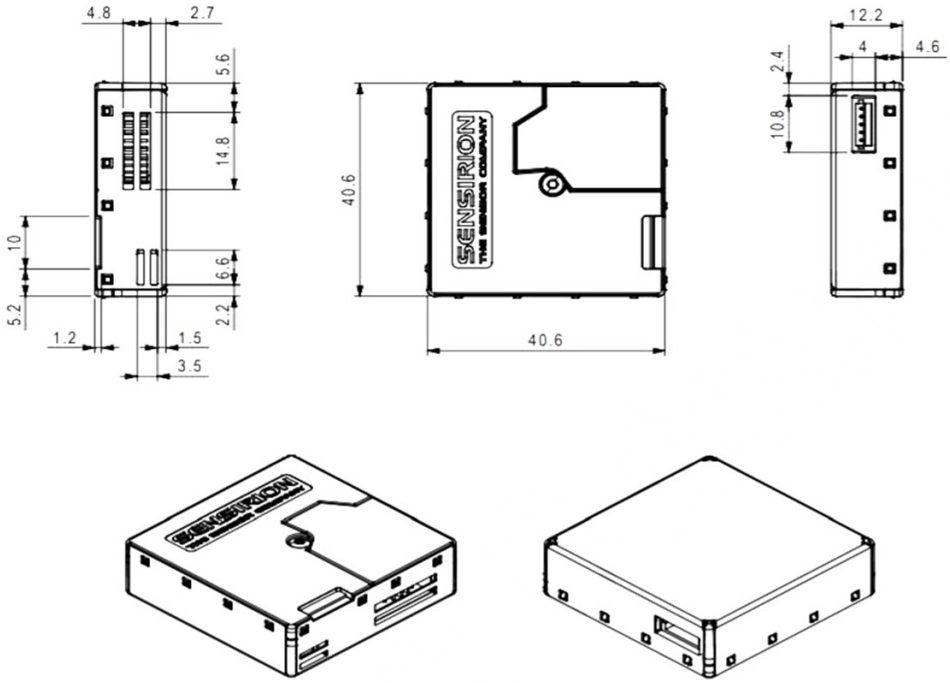


Figure 24: Sensirion SPS30 Layout[50]

mm. It uses an I^2C communication link supporting up to 400kHz with an average low power consumption of 0.2 mW making it an ideal choice for use in this design. The sensor's operating temperature ranges from -40°C to +65°C and requires a supply voltage of 1.7V – 3.6V. In its first operation mode, although not able to detect gas concentrations of other gases the ZMOD4510 sensor module is capable of directly calculating the AQI based on EPA standards. It does this by using embedded artificial intelligence (AI) and machine learning (ML) algorithms to detect several gases in the atmosphere. Additionally, its second mode of operation is a selective O3 measurement using a low power mode. This provides flexibility to the overall design of the system, as it can be used to directly calculate AQI or just have the sensor output an ozone concentration reading and using that along with the other sensors in Aether to calculate the AQI. The different measurement capabilities and characteristics of the ZMOD4510 sensor based on its selected operation method are shown in the figure below.

Bosch BME688 Gas Sensor – VOCs and VSCs The Bosch BME688 detects Volatile Organic Compounds (VOCs) and Volatile Sulfur Compounds (VSCs) and through the use of sophisticated AI embedded in the system, machine learning algorithms can be implemented to detect a variety of other gases and compounds including carbon monoxide (CO2) and hydrogen. Additionally, through the use the BME AI studio provided by Bosch, the sensor can be customized and trained to detect different smells such as methane or smoke making the overall design of the Aether much more flexible and capable of detecting early on set symptoms of methane leaks, fires,

Symbol	Parameter	Conditions	Minimum	Typical	Maximum	Unit[a]
AQI	Air Quality Index	OAQ 1st Gen: Rating according to EPA for ozone and nitrogen dioxide	0		500	
		OAQ 2nd Gen: Rating according to EPA for ozone	0		500	
	Measurement Range OAQ 1st Gen	Ozone (non-selective)	20		500	ppb
		Nitrogen dioxide (non-selective)	20		500	ppb
	Measurement Range OAQ 2nd Gen	Ozone (selective)	20		500	ppb
		NO_2 cross sensitivity at 200ppb		25		AQI Level
RH	Humidity Range	Non-condensing	5		90	% RH
T	Temperature Range	Typical outdoor environment	-20		50	°C
		Extended range	-40		65	°C

Figure 25: Renesas Module Characteristics [47]

and other smells hazardous to humans. BME688 also provides consistent and stable temperature and humidity readings eliminating the need to integrate another sensor onto the design and thus saving time and space. Since VOC and VSC concentrations are always higher in indoor environments, it gives the Aether sensing network the ability to be an effective indoor air quality IoT sensing system while also contributing to its ability to measure the AQI in outdoor environments.

The BME688 is enclosed in an 8-pin LGA assembly with dimensions of 3.0 x 3.0 x 0.93 mm3 making it an extremely compact design. This allows Aether to add several robust sensing features while keeping the overall design as compact and portable as possible. It uses an I^2C or SPI communication link with an operating supply voltage of 1.2 V – 3.6 V. Its temperature detecting ranges from -40°C to +85°C and can detect ambient humidity up to 100% . The gas sensors and humidity sensor response times are both under 11 s contributing to fast and accurate measurements. To achieve the sensors full operating potential, the Bosch Sensor Environmental Cluster (BSEC) and BME6xy API will need to be used to train the sensor to detect the required gases and smells. BME688 is equipped with 5 different operating modes.

- Gas Scan Mode: Using the BME AI Studio the sensor can be trained to scan and detect selective gases for AQI and indoor air quality applications with a 10.8s refresh scan rate.

- Ultra-Low Power (ULP): This mode is designed for use for battery-powered operation over longer periods of time. It operates the sensor with a < 1 mA consumption and a 300 s refresh rate.

- Quick Ultra-Low Power (q-ULP): This mode detects temperature, pressure and humidity with a 3s refresh rate and a not much higher current consumption.

- Low-Power (LP): Designed specifically for an interactive application tracking air quality, this mode has a 3s refresh rate with a < 1 mA current consumption.

- Continuous (CONT): With an update rate of 1 Hz, this mode is intended for use in short-term uses where extremely fast events occur and fast responses are needed.

The overall design of this product allows it to be used in battery-powered IoT applications where size and power consumption are of paramount importance. At a price of $20 its low cost allows the design to have much more capability and flexibility in its measurements while keeping the total product cost relatively low.

5.7.5 Other Sensors Considered

This section discusses different sensors that were considered for this project but ultimately not used. Initially, the thought process behind having an air pollution detection network was to use an individual gas sensor for each different gas detected in order to calculate the AQI Index. As more research was done, it became obvious that using a separate gas sensor for each pollutant would unnecessarily complicate integration onto the system while increasing power consumption, cost, and overall product dimensions. Some of the sensors considered for the Aether sensing network are discussed below.

Spec Sensors - Multigas The company Spec Sensors produces a variety of high performance, ultra-low power consumption electrochemical gas sensors. As discussed in the sections above, electrochemical based gas sensors typically have a lifespan of approximately 1-3 years, significantly less than other sensors researched, making the overall design not ideal for long-term low maintenance use. Additionally, these sensors are also cross-sensitive to other gases and because a variety of different gases will be measured in this design, the likelihood of inaccurate readings would be increased. Electrochemical sensors base their measurements off electrochemical reactions which are temperature dependent. Although the built-in sensor design compensates for temperature changes, rapid changes in temperature and humidity may cause all the sensors in the system to temporarily destabilize and affect product performance. At a cost of $20 per gas sensor, using Spec Sensors for monitoring air quality in our design would have also increased overall product cost while not providing much flexibility in design capabilities and making circuit integration more difficult.

Sensirion AG – Temperature and Humidity Initially for this design, a separate sensor intended for specifically monitoring ambient temperature and humidity was going to be used. However, after conducting more research on available products, the Bosch BME688 would not only provide readings for VOCs and VSCs but could also provide stable and consistent temperature and humidity measurements. Thus, eliminating the need to integrate another sensor onto the PCB design thus further limiting the cost and size of the overall product.

After considering different sensors and the different operating principles they used to detect gases, it was determined the best course of action would be to use

sensors that have low power consumption while also having the ability to detect more than one type of gas. This would allow for the system to use less sensors and minimize the size of the main PCB. The Bosch BME688 uses built in machine learning software and is capable of being trained to detect VOC/VSCs, temperature and humidity, and a variety of different gases. Although not able to provide direct measurements for other gases, it still gives the system the ability to detect harmful air pollutants. The Renesas ZMOD4510 sensor used in the design can measure both nitrogen dioxide and ozone which are two of the most prevalent air pollutants used in calculating the AQI Index.

5.8 LoRaWAN Gateway

In a LoRaWAN network, the gateway exists to receive data packets from LoRa end-devices and send those packets to an network server. It is a critical component for our sensor node to function. For our project, the sensor node is the main design focus and where the majority of our time will be spent.

While it is possible that we could function without our own gateway, we did not want to have to rely on public networks to be able to test our sensor node design. We couldn't count on there being reliable service, if there was any at all to begin with. We determined that having our own gateway would also make testing and debugging our design easier. Since we opted to use The Things Stack as our LoRaWAN network server, we decided to buy The Things Stack Indoor Gateway for development and the RAK2245 Pi Hat for long-range, outdoor use and testing. The RAK2245 Raspberry Pi Hat is an adapter board that provides the Raspberry Pi with the ability to function as a LoRaWAN gateway. It has both a GPS and LoRa antenna connection through two U.FL RF connectors. There were a variety of factors that lead to us choosing to go with the Raspberry Pi/RAK2245 solution for our gateway. One of the primary drivers were cost. A member of our team already owned a Raspberry Pi and has experience working with one. This helped reduce cost for the project. Since the gateway is not part of the engineering design focus of our project, we determined that this method for creating a gateway would help reduce the time spent here and allow us to spend more time on the sensor node and application layer software.

5.9 The Frontend Software Stack

In IoT embedded systems, you usually have three options for the front-end stack: bare-metal, a real-time operating system (RTOS), or full operating system, like Linux. In this context, "front-end" refers to the edge computing devices, and "back-end" refers to the cloud services. For our project, Linux isn't an option due to power constraints, which in turn affects processing power. Also, most low-power microcontrollers, including some that include ARM Cortex M0 and M4 cores, don't have a memory management unit (MMU), which is required by the Linux kernel. So, we are really left with two options: bare-metal or an RTOS. Before discussing either choice, we first will discuss the software requirements for the front-end.

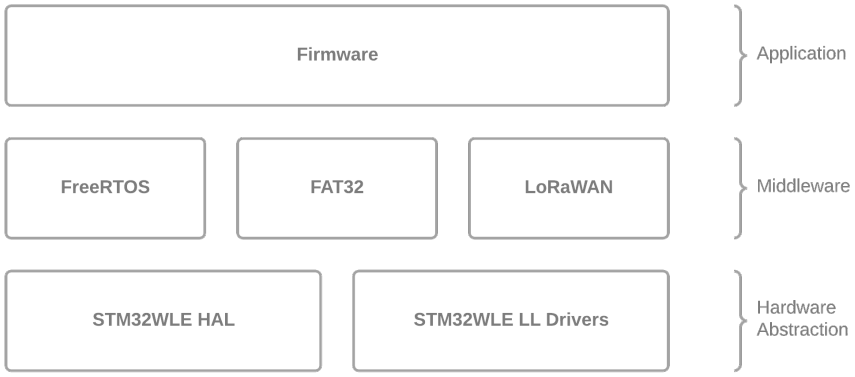


Figure 26: End Device Server Stack

At its core, all the sensor nodes need to do is periodically take readings and transmit them back over the LoRaWAN protocol to a gateway. Once it is received by the gateway, the results are sent to our back-end service for additional processing. The nodes should be able to receive commands over LoRaWAN to adjust how often sensor readings should ideally take place, perform firmware updates, or update the device's configuration. The device should be aware of whether it is plugged into USB power so that it can change its power savings mode. It should make decisions based on these characteristics, such as how often to take sensor readings, and therefore, how long it should sleep for (and the sleep mode).

On the LoRa-E5, the device is equipped with a ARM Cortex-M4 microcontroller capable of sleep, deep-sleep, and standby low-power modes, each have the ability to be woken up from the real-time clock (RTC) or specific I/O pins. Any choice in choosing front-end software stack components should not hinder the core's ability to move into either of these modes. Ideally, the standby low-power mode is preferred since it offers the best power efficiency - down to approximately 360 nA. If the device is plugged in over USB, the device doesn't need to go into sleep anymore since power is being provided through USB. Also, if the device were asleep, then it wouldn't be able to respond to the user through the command-line interface (CLI). Thus, if plugged-in, the device should never go to sleep as it'll be powered by USB anyway. As just mentioned, the device should provide a CLI interface for configuration or view real-time sensor measurements. The software requirements are shown below.

The device should be able to:

- Enter the standby low-power mode
- Periodically take sensor readings as the frequency specified by the user, if possible
- Be woken up when connected to a computer over USB

- Provide a CLI when connected over USB that allows for the device's configuration and real-time output of the sensor's data

5.9.1 Bare Metal Architecture

A bare metal software architecture refers to programming the microcontroller directly by modifying its registers or with its hardware abstraction layer (HAL). With bare metal programming, there is no operating system involved. A HAL provides facilities for controlling the microcontroller and accessing its peripherals independent of the specific model, but only for the same vendor. The other method of bare metal programming is through using the MCU's header files that contain register definitions (addresses), and manually reading from and writing to them.

The architecture of a bare metal program typically includes an initialization phase followed by an infinite loop. Inside the infinite loop, registers might be polled and tasks can be done. The infinite loop can essentially be a synchronous event loop. Asynchronous execution can only happen through interrupt service routines (ISRs), or if the microcontroller has more than one core. ISR handlers must be manually written for bare metal programming to allow for asynchronous, non-blocking events to occur. ISRs are typically written for direct memory access (DMA) controllers, analog to digital converters (ADCs), I/O, such as I^2C or pin interrupts, and much more. When an ISR occurs, usually some kind of global variable is update to inform the event loop that something has occurred. Bare metal programming is typically done when the microcontroller doesn't have the processing power, or other peripherals, to run an operating system. A bare metal program allows for the most amount of determinism, which is important for for real-time systems. RTOSes try provide some higher level abstractions while maintaining determinism. Another reason why bare metal programming might be chosen over using an operating system is that bare metal programming can provide the best performance for both compute and memory usage.

Figure 27 depicts an example of the control flow a bare metal program typically takes. On boot, the main function is executed. This is followed by one or many initialization functions that configure the system for the specific application. This usually involves configuring the clock speed, ADC timings, DMA initialization, I/O pin direction and functionality, etc. Once the initialization is complete, the main loop executes infinitely until either the microcontroller hits an unhandled trap (error/exception) or is powered off. Each task, task 1 through n, is completed synchronously. The task might poll a bit in a register to see if, for example, an I/O transaction is complete, or it could be performing a computation. The synchronous execution can only be interrupted through an interrupt, assuming there is a enabled and linked ISR. In the figure, task 3 is asynchronously interrupted twice. First, by a single-level interrupt. And, second, by an interrupt that was then interrupted by an interrupt with a higher priority than the one currently executing. In between the first and second interrupt, task 3 was resumed, executed some more, and then paused again for the

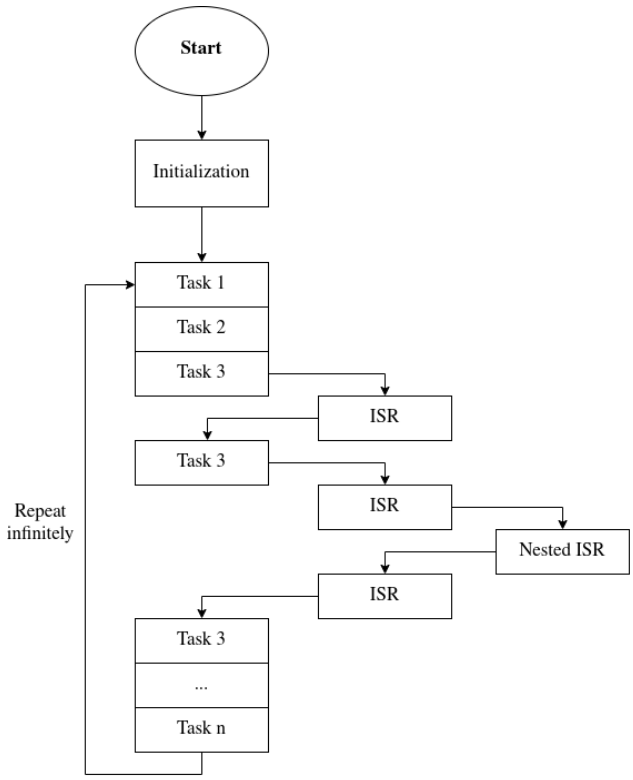


Figure 27: Typical Bare Metal Embedded System Software Architecture

second round of interrupts. When the CPU is interrupted, the state of the CPU is saved onto the stack. When the ISR completes, the state is restored and what was previously executing can as if nothing had changed. However, this can cause problems during critical sections of code. Usually, ISRs can be disabled and deferred during the sections, or the code can be rethought. Finally, after the second round of interrupts, task 3 completes and so does the rest of the loop. The execution will repeat starting at task 1, but the interrupts don't necessarily have to execute (as shown) at the same as the previous loop. Interrupts can happened whenever.

5.9.2 FreeRTOS Architecture

FreeRTOS is an specific type of operating system that focuses on determinism and predictability, rather than responsiveness, like in Linux, Mac OS X, or Windows. It supports preemption and co-operative scheduling, as well as priority inheritance to prevent priority inversion. FreeRTOS is small. It only requires 5 kB to 10 kB of RAM for operation. FreeRTOS, like many other RTOSes focuses mainly on task scheduling and communication. In fact, the main source files provided for FreeRTOS are:

- `tasks.c` - for creating and managing task's priority and state
- `queue.c` - for task to task communication - also includes mutex and semaphors implemented with queues

- `list.c` - a generic linked-list implementation
- `timers.c` - timer implementation
- `port.c` - MCU specific code for porting FreeRTOS
- `FreeRTOSConfig.h` - FreeRTOS configuration

All of the data structures provided by FreeRTOS integrate with tasks and will automatically block/un-block them, such as waiting for a timer or other event. Code written for FreeRTOS can also be transferred to other microcontrollers that are not necessarily from the same vendor. This is because of the layered software abstraction architecture. At the lowest level, is bare metal programming with a HAL. Above that, is MCU specific code to implement low-level FreeRTOS code, called a port (`port.c` as shown in the list above). On top of the platform specific FreeRTOS port code is the platform independent FreeRTOS code. This is the code that is responsibly for the main logic of FreeRTOS, such as the kernel, queue and list data structures, and other middleware. The user/programmer code is written above the platform independent FreeRTOS code. This has many ramifications, such as what was already mentioned back code portability. Since the FreeRTOS main interface is platform independent, middleware can be more easily written in a standardized way, such as providing support for FAT file systems, USB communication, LoRa/LoRaWAN, cryptography, and much more.

At its core, FreeRTOS is all about scheduling tasks and communicating between them. Most of the code is oriented around configuring and initializing tasks. FreeRTOS is deterministic because of its scheduling algorithm. Generally, the task with the highest priority should be running. A task with a priority of 0 has the lowest priority, like the idle task, and a task with priority `configMAX_PRIORITY - 1` has the highest priority. The `configMAX_PRIORITY` definition is a user defined parameter inside of `FreeRTOSConfig.h` that defines how many priorities there are for scheduling. The more priorities there are, the more memory the kernel will need since each priority needs a list allocated for each task state. Priorities are manually assigned by the programmer, so with knowledge of the scheduling algorithm (which can be configured), the programmer can reason about potential issues much easier.

The scheduler can be configured to be preemptive or non-preemptive. With non-preemptive scheduling, tasks must either block, suspend, or complete before another task can be executed. Once a task is put into a not-running state, the task with the highest priority is selected. If two tasks have the same priority, they are selected using a round robin algorithm. Non-preemptive scheduling is good if tasks are commonly executing critical sections of code, or using shared resources. However, this can lead to some tasks being starved and are not able to make progress even though they have a higher priority than the task currently running since it never blocks or suspends. Each task state is represented as an array of lists where the index into the array represents the task's priority. The number of priority levels is defined by the programmer in

`FreeRTOSConfig.h`. For example, when a task with priority 2 moves from the blocked to ready state, the scheduler will find it in the blocked array at index 2. It'll then search through the list that's at index 2 until it finds the task, and it'll move it to the list in the ready array at index 2. Each time the scheduler executes, it'll iterate through each list inside the ready array. Each task has a task control block (TCB) to manage the task's state, like where its stack is located (start and end addresses), its priority, its name, and pointers to list item objects so that the list the task is in can be found quickly.

Figure 28 shows the different states a task can be in along with state transitions. The two main states are: running and not-running. Not-running is a super state, which means it has sub-states. If a task is not running, then it can be either suspended, blocked, or ready. A task that is suspended is not available to the scheduler, and therefore cannot make progress unless manually resumed by another task. Tasks can be put in the suspended list from any other state, and can be done through the `vTaskSuspend()` function. When `vTaskResume()` is called, a task is moved from the suspended state (list) to the ready state (list). A blocked task is similar to a suspended task except that a blocked tasks is available to the scheduler and can make progress once it is unblocked. A task can be unblocked from many different kinds of events, such as from a timer expiring, a specific interrupt triggering an event, the availability of a semaphore, or the unlocking of a mutex. Before any task can be executed, it must first be moved into the ready state. Whenever the scheduler runs, either after each tick or when a running process stops running, it looks in the ready list and looks for the highest task to run.

A preemptive scheduler interrupts tasks whether they want to be or not. A preemptive scheduler interrupts tasks at a pre-defined interval, set in `FreeRTOSConfig.h`, called a "tick". Preemption can also occur when a task with a higher priority becomes ready for execution. At every tick, the scheduler will set the tasks with the highest priority as the active task, choosing tasks with an identical priority using a round-robin algorithm. Whenever there are no tasks in the ready list, the idle tasks is executed. The idle task has the lowest priority level, which is 0. The idle tasks also allows the programmer to hook into it to execute a handler each time the idle task is entered. The idle tasks performs maintenance, such as cleaning up freed memory freed during the task execution. When a task frees memory, that memory isn't actually removed from the heap until the idle task is ran in order to keep user tasks deterministic.

Figure 29 shows an example of the execution of three tasks each with different priorities. The vertical axis represents the task's priority, while the horizontal axis represents time and how long tasks executed. The kernel tasks and interrupts always have the highest priority, and are depicted at the top in red. The kernel runs the scheduling algorithm each time a task goes from running to not running, or on each tick assuming it is configured to use preemption and tick-mode. In the figure, arrows are only added when the scheduler/kernel runs and swaps the currently running task with another. Each time there's a red bar, the kernel is running instead of a previously

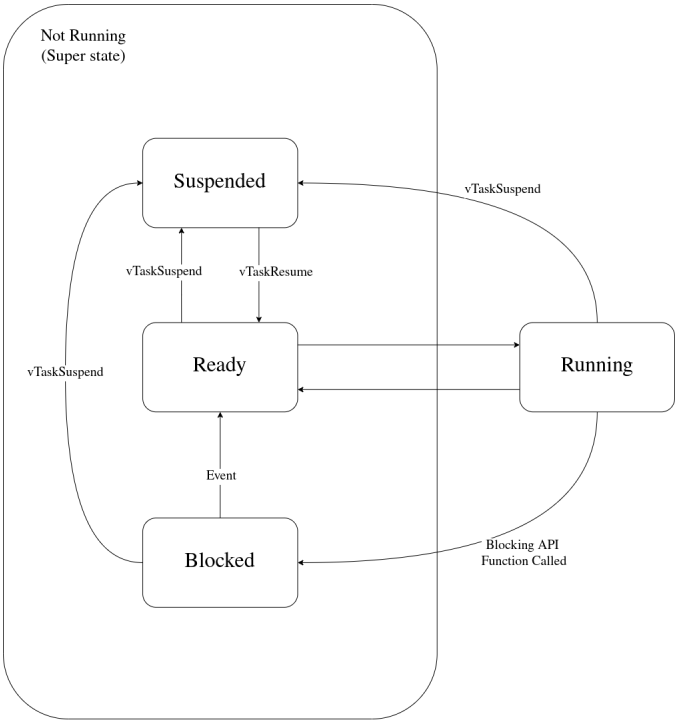


Figure 28: The FreeRTOS Task State Machine

running task, but arrows are not included so the diagram doesn't look too busy. Thin, horizontal bars between tasks on the same priority represents that task moving into the ready list, and is based on the task color.

Initially, Task A is running, but it preempted by Task D when it becomes unblocked due to an asynchronous event at time t_1, such as a timer or a transaction is received over I^2C. At time t_2, Task D is blocked, so the next available task to run with the highest priority is selected, Task B. Since Task B and Task C share the same priority level, on the second tick (time t_3), Task C is ran since the scheduler tries to be as fair as possible for tasks sharing priority. Similarly, at time t_4, Task C is ran instead of Task B since Task B and Task C are the only tasks that share a priority of 2. At time t_5, Task B is blocked, so Task A is ran since has the highest priority level of all ready tasks. Next, at time t_6, Task A is blocked or suspended, and since there are no ready tasks, the idle task is ran. Finally, at time t_7, Task D is moved into the ready state, and the scheduler starts executing it.

However, preemptive scheduling can cause synchronization issues for accessing shared resources. Consider the following example: two tasks, Task A and Task B, have the same priority and want to transmit strings over same UART channel, the shared resource. Task A wants to send `"abcdefghi"` and task B wants to send `"123456"`. However, the example would still work if task B had a higher priority than task A, task B is blocked or suspended initially while task A is running. At tick, t_0, task A is running and starts sending its message. At tick, t_1, since task A and task B

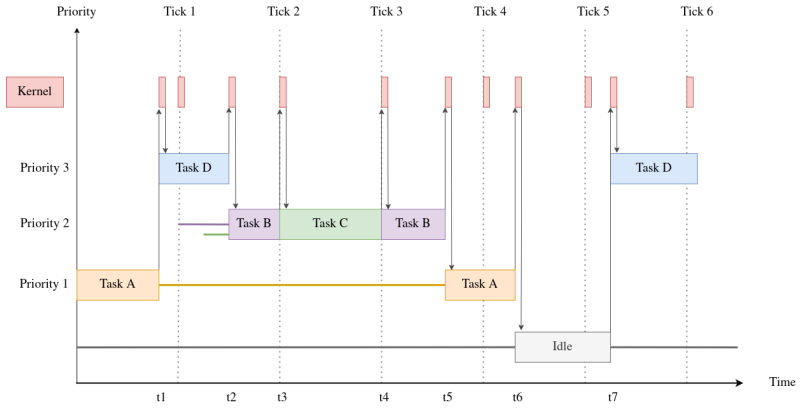


Figure 29: FreeRTOS Preemptive Execution

have an identical priority, task B is swapped with task A. But, at t_1, task A was only able to send `"abcd"`. Now, consider just for the example, that task B is able to send the entirety of its message, `"123456"`. At this point in time, `"abcd123456"` has been transmitted over UART. Finally, at tick t_3, task A is able to complete sending its message by sending the remainder, `"efghi"`. The end result is that `"abcd123456efghi"` was transmitted over UART. The message was corrupted.

Luckily, FreeRTOS, like other operating systems, provide utilities for synchronizing asynchronous operations: mutexes and semaphors. While similar, they are used for vastly different scenarios. Mutexes act as keys for locked doors. Only one person, or process/task, can hold onto the key at the same time. In other words, they are binary. Mutexes are used for synchronizing access to shared resources. For each resource, akin to having multiple locked doors, requires an unique mutex, or key, to gain permission to access it. Semaphors, while similar in implementation to mutexes as they can be explained as non-binary mutexes, their purpose is vastly different. Semaphors are meant for signalling events. The key difference here is that with semaphors, there is a producer and consumer, and they don't need to be the same task/process. Whereas with mutexes, only the task that holds the mutex can release it.

A problem with using semaphors as mutexes is that a semaphore doesn't portray which resource has been taken. Say there are two UART channels and two tasks. If a semaphore were used to allocate one of the UART channels, then the other task wouldn't know which UART channel is available. In order to portray that, state variables would need to be introduced, but this can cause more errors. It's better, for any kind of resource allocation and synchronization, to use unique mutexes for each. All sources used in this subsection comes from the FreeRTOS reference manual [51], the FreeRTOS tutorial guide [9], and a breakdown of the FreeRTOS internal architecture [58].

An issue that arises in RTOSes, and operating systems in general, when using

synchronization mechanisms is priority inversion. Priority inversion is a situation where a lower priority task effectively has higher priority than another task that was originally defined with a higher priority. It is more clear through the use of an example. Consider three tasks, A, B, and C, with priorities 1, 2, and 3, respectively. Tasks A and C share a resource that uses a mutex to synchronization. Consider, that initially task A is executing before C, and grabs onto the mutex. Then, some time later, task C is unblocked and begins executing and wants to use the shared resource, so it tries to grab onto the mutex. The mutex already belongs to task A, so now task C must wait and block. After C starts to block, task B begins to execute. If task A was previously executing, which it would've since it took the mutex for *a reason*, task B would have preempted it. Now, as long as B is executing and A holds the mutex, then C cannot make progress even though it technically should be since it has higher priority. The only thing keeping it from executing is the execution speed of task A and how much processing it does while holding onto the mutex. Thus, effectively, the priorities between task B and C have been swapped.

The solution to priority inversion is priority inheritance. Priority inheritance is where two tasks share a resource guarded by a mutex, and the task with the highest priority will share its priority with the lower priority task so that the higher priority task can make progress as soon as possible. The higher priority task might be performing safety critical or just critical sections of code, so it's important that they preempt lower priority tasks or let them finish as soon as possible if they hold mutexes. FreeRTOS implements priority inheritance by adding a pointer field, `pxMutexHolder`, which holds a pointer to the task that currently owns the mutex. If a task with a higher priority is waiting on the mutex, then it'll check `pxMutexHolder`, and if `pxMutexHolder` points to a task that has a lower priority, then the FreeRTOS scheduler/kernel will let it inherit the other task's priority.

Lastly, one of FreeRTOS's features that would be of interest for this project is its tick-less operation mode. As previously mentioned, tick-less modes allow the tick interrupt to be disabled to allow the device to go into a deep sleep. This is needed for the MCU to go into deep sleep because the tick interrupt would wake up the system, and any currently running tasks that depend on timers would be disrupted. Tick-less mode can be enabling tick-less idle in the FreeRTOS configuration file. The tick-less idle mode will automatically be entered when the following conditions are met:

1. The idle task is the only task able to run because all other tasks are in the *blocked* or *suspended* state

2. The idle tasks is expected to be executed for n or more complete ticks, where n is a user configured value called `configEXPECTED_IDLE_TIME_BEFORE_SLEEP`

5.9.3 Zephyr RTOS

Zephyr is an open-source, real-time operating system being maintained by Nordic Semiconductor. Zephyr includes all of the features one would expect to find in RTOS,

```
const struct device *dev_bme;
dev_bme = DEVICE_DT_GET(DT_NODELABEL(bme688));
sensor_sample_fetch(dev_bme);
sensor_channel_get(dev_bme,
                   SENSOR_CHAN_AMBIENT_TEMP,
                   &temp);
```

Figure 30: An example of initializing the BME688 and getting a temperature reading using the Zephyr device model.

such as threads, semaphores, mutexes, and other kernel services. In addition, Zephyr as built-in support for power management, a user shell, and logging functionality. Zephyr uses a CLI tool called `west` to build applications and flash them. Our team prefers this to using the standard STM32CubeIDE tools.

One of the major distinguishing features of Zephyr over other popular RTOSes, such as FreeRTOS, is its use of a consistent device driver model. Zephyr provides a consistent device model for configuring the drivers that are part of the platform and a consistent model for initializing all the drivers. This makes drivers more straight forward to implement. Additionally, devices become much easier to interact with in the application code. For example, in Figure **??**, we demonstrate how to initialize a device and take a reading from a sensor. Many details, such as the I2C address of the device, are abstracted away when using Zephyr.

5.9.4 Architecture Deliberation

One of the main considerations in choosing parts and in making software architecture decisions is power efficiency. A bare-metal architecture would give us the ultimate control in how the microcontroller behaves at the cost of flexibly, code re-use, code maintainability, and time required to build a complete prototype. A bare-metal architecture typically involves a main loop that polls various register values until certain conditions (application specific) are met, does a few actions, then repeats. Interrupt service routines (ISRs) are written specifically for that application. If more functionality needs to be added, it could touch every piece of the bare-metal architecture. These lead to more complicated implementations that don't tend to scale well. Adding a 'simple' feature could end up requiring a complete re-write of the application.

For simple applications, a bare-metal architecture is perfect. However, if there are lots of asynchronous events, such as I/O, it becomes more difficult to maintain. Also, by using a RTOS, it allows the programmer to use middleware, such as a FAT file system driver. So, ideally, we would like to use a RTOS, specifically FreeRTOS since it is officially supported by STM. However, RTOSes maintain time by interrupting the system at a set frequency, or a tick, to do job control and other maintenance. This tick-architecture prevents the microcontroller from going into sleep since the RTOS needs it to maintain time for process timers. Luckily, FreeRTOS has a tick-less mode

which allows the device to go into deep-sleep and standby mode. Therefore, since we need to deal with both USB and LoRa communication, as well as other asynchronous communication with the sensors, we chose to go with FreeRTOS for the main part of the front-end stack.

The middleware provided for free for the LoRa-E5 is the following: FatFS, FreeRTOS, LoRaWAN, KMS (key management service), cURL, mbed-crypto (cryptographic services). For the sensor nodes, we will be using FreeRTOS, LoRaWAN, FatFS for training the sensor ML models, STM FUOTA (Firmware Update Over The Air) and mbed-crypto for securely transmitting data over LoRa.

5.10 The Backend Software Stack

The backend software stack will be responsible for collecting data from the sensor nodes and displaying the air quality data on web page. It will also be responsible for sending alerts via email about poor air quality conditions. It will be the main way users will interact with their deployed sensors and view data. The backend is composed of three main components: (1) the LoRaWAN gateways running the LoRaWAN Network Server, (2) a server running on an IoT host provider that can collate data from the gateways and provide other backend services, and (3) a web user interface running on the IoT backend provider (such as AWS IoT or Azure IoT) that displays statistics and user specific information. Additionally, the LNS can optionally be ran in the cloud. In this scenario, the gateway runs a LoRa packet forwarder that understands the LNS protocol. Packets can be forwarded to the LNS data endpoint server, and the gateway is managed through a configuration and update server (CUPS) using the CUPS protocol along with HTTPS. While the following software solutions offer support for LoRaWAN versions 1.0.x and 1.1, since the chosen LoRa module only supports up to 1.0.3, the following discussions will be with LoRaWAN 1.0.3 as there are some differences with the architecture. The changes are in regard to the process of join the network.

There are two types of LoRaWAN networks: public and private. Public networks are networks akin to Verizon or AT&T. They provide coverage and networking. Two of the biggest LoRaWAN public networks are The Things Network and The Helium Network. Once registered on the network, LoRaWAN devices can operate with the developer's application server which is configured during registration. The other option is to configure a private network. When creating a private network, there are generally two main architectures to choose between in order to route packets to an application server. The first option, which will be referenced as local-LNS, is to run the entire LNS on a gateway device. This means that the gateway needs to manage more information about the end-devices and the state of the network. The second option, cloud-LNS, is where the gateway operates purely as an access point, and forwards packets to a LNS in the cloud. This offers greater scalability and lower compute and power requirements of the gateway device at the expense of added complexity. Luckily, there are many solutions available through service providers,

such as AWS and private configurations on The Things Stack.

5.10.1 LoRa Packet Forwarding

LoRa packet forwarding is the key function of a LoRaWAN gateway. As stated before, the goal of the LoRaWAN gateway is to take data being transmitted from LoRa end-devices and route it to the internet via another communication protocol, such as WiFi. To perform this action, a LoRaWAN gateway contains a software that is referred to as the packet forwarding. This software is responsible for demodulating LoRa RF packets received from end-nodes and transmitting them to the server.

Many different LoRa packet forwarding software exists already and can be used by anyone setting up a custom gateway. An example is the Semtech UDP Packet Forwarding. This is the original LoRa packet forwarding created by the company who created the LoRa standard. In fact, most off-the-shelf gateways include a pre-compiled version of this software. Chirpstack and The Things Stack use the LoRa Basics Station software for forwarding packets to their respective network servers. Chirpstack packets can be forwarded to another service running locally on the gateway or on a remote server. In the case of The Things Stack, the packets are forwarded to The Things Stack Cloud.

5.10.2 LoRaWAN Network Servers

The sensor nodes communicate over LoRa using the LoRaWAN protocol to gateways. These gateways are running what is called the LoRaWAN Network Server (LNS), which is a standardized protocol for connecting devices using LoRaWAN to the internet. LoRaWAN network servers are the bridge between LoRa and the Internet. LoRa network servers can be public or private, and end-devices must first be registered on the network before it can join it. All packets sent by an unregistered and unverified end-device will be discarded.

As previously mentioned in the LoRaWAN standardization section, devices communicating over LoRaWAN perform a handshake (joining the network), and exchange a set of IDs that identify the device, the join server, and the application/network session. These IDs are: `DevEUI`, `JoinEUI` (known previously as the `AppEUI`), and `AppKey`, respectively. The remainder of the keys are generated from this set of keys. The `AppKey` is the only key that isn't transmitted over the network, and is stored on both the end-device and the network server. For LoRaWAN 1.1+, an additional key is required to be stored on the end-device and the network server: the `NwkKey`. The `NwkKey` is like the `AppKey` in that it is never transferred and is used as an extra encryption layer. The network server will need to be integrated and capable of communication with the application server, which in for the project, such as Amazon AWS, Microsoft Azure, MQTT, or REST webhooks.

For public networks, which are basically like Verizon or AT&T for LoRa, the network server software is provided to users who want to setup a gateway. Once setup,

anyone with reception will be able to use the network. The Things Community Stack is a free and optionally public network. Users that create a gateway are not financially incentivized, but only are if they have a end-device they would like to use in their area. However, it is optional to configure a gateway as public on The Things Stack. On the other hand, The Helium Network provides financial incentives without a central authority through a specialized blockchain. For people and companies that would like to setup a private network, an open-source network server, called Chirpstack, is available.

The Chirpstack Network Server Stack The Chirpstack LoRaWAN network server stack is an open source implementation of the LoRaWAN network server protocol standard. It provides integration with every aspect of running a private LoRaWAN network from the packet forwarding server to a web interface to control the configuration of the network and user registration onto the network. It supports all classes of the LoRaWAN protocol (A, B, and C), including adaptive data-rate. The Chirpstack can be setup to run on one machines, or spread out over multiple. Meaning, that the packet forwarder, network server, and application servers can be completely decoupled. The Chirpstack supports the MQTT protocol for event-based communication, and has built-in integration with AWS, Azure, and many other back-end service providers.

5.10.3 Public LoRaWAN Networks

Public LoRaWAN networks allow users to connect their own LoRa end-devices to them and begin sending data to the internet. These networks make it easier start using LoRaWAN devices and removing the need to purchase, set up, and maintain large amounts of LoRaWAN gateways. Some public networks, like The Things Community Network, are entirely free to connect to and use. However, there are others, such as The Helium Network, that take advantage of blockchain technology to charge users for the usage of the network. This functions as a way to provide a monetary incentive for users to add gateways to the network, making it larger and increasing it's coverage.

The Things Community Network The Things Community Network is a public LoRaWAN network that uses The Things Network Stack. This network was previously known as simply The Things Network. The software for this network is entirely open source, so it would be possible for someone to run their own instance of this network on their own servers. Currently, The Things Network has over 21,000 gateways in operation providing great network coverage. However, the vast majority of network coverage is in Europe. In the United States, coverage is largely centered around the northeast, with some coverage in Florida.

The Things Community Network provides a tool for adding LoRaWAN devices to the network known as The Console. It is a web application which can be used to register applications, end devices or gateways, monitor network traffic, or configure network related options, among other things. When adding a device to the The

Things Community Network, there are preset device profiles for mainstream, off-the-shelf devices to make set up very straightforward. Otherwise, if a user is attempting to add a custom device, they will have to manually provide their device information (`DevEUI`, `JoinEUI`, etc.). Due to the free and public nature of The Things Community Network, they specifically mention that users should take actions to limit network use, such as limiting data transmission frequency, optimizing message encoding for size, and avoiding confirmed uplink messages [31]. They recommend these practices to ensure that no single user is hoarding the usage of network resources. These sort of policies and suggestions are necessary when there is no cost to use a network.

The Helium Network The Helium Network is a public LoRaWAN network that takes advantage of blockchain and cryptocurrency technology and is made up of so-called Helium Hotspots. Hotspots produce and are compensated in HNT, which is the native cryptocurrency of the Helium blockchain. The goal of the Helium blockchain is to incentivize people to set up LoRaWAN gateways to increase the coverage of the network itself.

The Helium blockchain itself is based off of a concept known as Proof of Coverage. The goal of the Proof of Coverage algorithm is to verify that Hotspots are located where they claim to be located, as well as that they are accurately reporting the area that they claim to cover. The algorithm attempts to continuously verify these details. The Proof of Coverage does this by taking advantage of the unique properties of radio frequencies. Due the fact that the strength of a radio frequency signal is inversely proportional to the square of the distance from the transmitter and the fact that radio frequencies travel at the speed of light with effectively no delay, the Helium network can use these properties to issue challenges to the Hotspots. Completing challenges is used to determine how much HNT a Hotspot is paid.

The developers provide a tool designed to assist users in registering their devices for use on the Helium Network. This tool is known as the Helium Console. Helium requires the creation of a user account. To add a device to Helium, users must either provide the `DevEUI`, `AppEUI`, and `AppKey` that came on the device or use the one auto-generated by the Console. All devices are also provided a unique identifier when connecting to the network. The Helium network supports any LoRaWAN capable device meeting the LoRaWAN v1.0.2 specification. The LoRa module we chose for our design meets this specification.

In order to discuss how to set up your own LoRaWAN gateway as a Helium Hotspot, we must first describe the different classifications of Hotspots that Helium describes. The first type is known as a Full Hotspots. Full Hotspots maintain a full copy of the HNT blochain. This class of Hotspot is able to participate in Proof of Coverage rewards and receive awards for forwarding data packets. However, in order to be classified as a Full Hotspot, the hardware vendor of the gateway must submit an approval form that is approved by the Helium Community and the Decentralized Wireless Alliance (DeWi). The second type of Hotspot is known as a Light Hotspot.

Table 18: Summary of Helium Hotspot types

Rewards Type	Data Only Hotspots	Full Hotspots	Light Hotspots
Network Data Forwarding	Yes	Yes	Yes
Proof of Coverage	No	Yes	Yes

Table 19: Microsoft Azure pricing structure

Pricing Tier	Standard Tier 0	Standard Tier 1	Standard Tier 2
Price per device per month	$0.08 per Month	$0.40 per Month	$0.70 per Month
Monthly device message allocation	400 messages	5,000 messages	30,000 messages
Included free quantities per application	2 free devices	2 free devices	2 free devices
Overage pricing per 1K messages	$0.07	$0.015	$0.015

Light Hotspots use Validators to get information about the HNT blockchain. This class of Hotspot is able to participate in Proof of Coverage rewards and receive awards for forwarding data packets. The final type of Hotspot is known as a Data Only Hotpsot. These Hotspots can only receive HNT through participating in data packet forwarding. There is no community permission required to add a Data Only Hotspot to the Helium Network. A summary of these Hotspot types can be found in Table 18.

It is important to note that there is an additional entity that is a part of the Helium Network, but it is not a Hotspot. It is known as a Validator. A consensus group of Validator nodes receive Proof of Coverage and device-related transaction requests from Light Hotspots and both verifies these requests and reaches an agreement on the ordering before forming a new block and adding it to the blockchain.

5.10.4 Application Service Providers

In this section we will describe two of the major competitors in the cloud hosting business, Amazon's AWS and Microsoft Azure. We will use the infrastructure from one of these two companies to host the software that will manage the data received from LoRaWAN gateways as well as to host the web-based user interface that will display statistics and information related to data collected from the sensor nodes. We will also need some compute medium in order to perform AQI or other calculations pertaining to our design and data flow. It should be noted that while we could explore the option where we perform our own hosting, the required initial monetary and time investment is far too great for our needs. This is in addition to the fact that both of these cloud service providers have free usage pricing tiers that include a feature set that is more than enough to meet our requirements. However, the pricing tiers are complex, and we may not necessarily stay within the free-tier because of unknown fees.

Microsoft Azure Microsoft Azure is a cloud computing service that is operated by Microsoft through their own data centers. Azure is composed of large number of different services. One of those services is known as Azure IoT. This is the service that we would primarily be using for our project. Azure IoT itself is made up of a large quantity of tools and sub-services that were created to assist users of Azure IoT with building their application. These would be the features of Azure that we would need to use and interact with to build our project.

IoT One of these tools is known as Azure IoT Central. This is a tool that allows for the rapid and easy creation of web applications due to the plug-and-play nature and large quantity of templates. This tool makes it easy to quickly setup a web application that can display data from our sensor nodes.

Another tool is known as Azure IoT Hub. The goal of this tool is to enable secure and reliable communication between an IoT web application and the devices that it manages. It is essentially a cloud-hosted backend that connects virtually any device. Azure IoT Hub also supports the ability to send device-to-cloud messages. These messages allow the user to understand the state of the device and define message routes to other Azure tools and services. Azure IoT Hub also supports the ability to send cloud-to-device messages. These messages allow the user to reliably send commands and notifications to any connected devices and track message delivery with acknowledgement receipts. This allows for bidirectional communication between devices and the cloud.

Besides features, another critical factor when deciding what cloud hosting service we are going to use is price. Azure IoT has three pricing tiers. As you move up in the tiers you gain access to resources, meaning you can connect more IoT devices and send more messages, however, you start to pay more per month. For our purposes, Standard Tier 0 is more than enough. We plan to connect no more than two to three devices (one or two sensor nodes and a gateway). This would mean that we stay within the "2 free devices" portion of the tier and even if we were to connect more, the price would still only be $0.08 per additional device per month. Additionally, our design does require the transmission of that many messages per day, meaning that should stay well within the limit of 400 messages per month. The breakdown of the pricing scheme can be found in Table 19.

Amazon AWS Amazon AWS is another major cloud computing service provider, especially known for their mature products and services, as well as consistent and dependable uptime. Amazon owns FreeRTOS, so there is dedicated middleware support for connecting to the cloud and using a variety of services. This includes update over the air and digital signing service functionalities. Amazon offers dedicated IoT services, as well as anything else to meet the needs of your application. The entire software stack, ranging from IoT device management, IoT event transactions, and IoT analytics to the SQL or NoSQL database, website hosting, and social integration, such as Twitter, email, or SMS. AWS most importantly has direct support for LoRaWAN

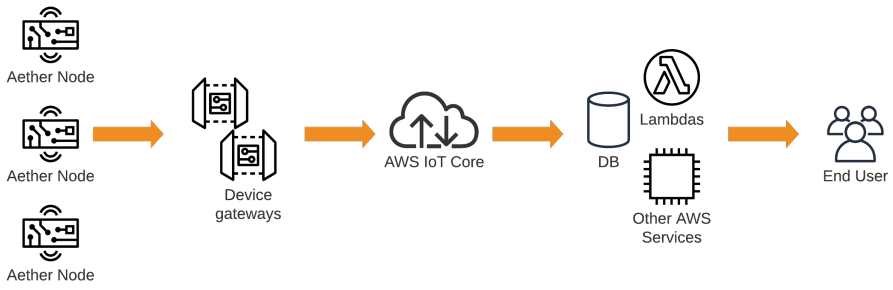


Figure 31: Amazon Web Services IoT Core Network Diagram

IoT devices. This means that you can run a LNS in the cloud, further securing the LoRa network, unlike Microsoft Azure which requires the use of a 3rd-party LNS without deep integration with the rest of their services.

IoT Because the LoRa network server doesn't have to run on the gateway, the gateway can have lower processing requirements and the data on the network is less susceptible to be able to be compromised since the gateway won't be managing the state of the network. In this scenario, the gateway would only implement a LoRaWAN packet forward, otherwise known as a *bridge*, to purely translate packets between LoRaWAN and the internet protocol (IP), and forward them to the LNS running in the AWS cloud. AWS' IoT Core allows the LNS service to integrate with the rest of an application's services through the MQTT protocol. MQTT stands for "Message Queuing Telemetry Transport", and it is a protocol built for IoT devices to provide an easier-to-use, light-weight publish/subscribe message transport protocol to take place of HTTP/HTTPS. It's perfect for resource-limited IoT devices where mainly all they need to do is inform a server of some event that happened. The MQTT protocol runs on top of the TCP/IP protocol. The IoT core also provides AWS Lambda execution, which are quick start up, server-less, and reactive compute service that automatically scales based on need. Lambdas, and other forms of server processing, can be triggered based on event rules. Rules are filters and binning services that allow an application to trigger events and/or run certain functions when an IoT message matching a rule-set is received. Rules can be created to filter messages based on device, node types, device groups, or any other field in MQTT messages.

Through AWS' IoT device management service, devices can be registered in bulk. It also allows users the ability to organize, track, and manage their devices depending on their business and security needs. Through the manager, firmware upgrades can be dispatched, and automatically manage which specific devices need to be updated. Commands can be operated on fleets of devices, and allow for secure remote execution. The IoT device manager has integration with many of Amazon's other IoT specific and non-IoT services to provide a cohesive experience for the developer, and speed users expect.

5.10.5 Server or Serverless

An another consideration that affected design decision making and cloud service providers is the available options of serverless options, and whether they would be a good fit for our project. Serverless computing refers to "function as a service" instead of the server model, "backend as a service" [8]. Serverless allow the developers to not have to think about server provisioning, running code across multiple server, managing data across those servers, and design complex scaling solutions. Serverless computing allows the developer to provide some function, specify some configuration, and have the code execute and scale automatically without worrying about server provisioning [52].

Since the backend for our project will mainly need to either: (1) respond to data packets from the sensor nodes or (2) respond to REST API requests created from the web application. For this reason, and the fact that serverless computing is easier to configure and get running, is why we opted to use serverless computing. Both Amazon and Azure offer serverless computing options. Serverless computing also allows for more reliability since there is less coupling between components of the backend system. Generally, specific operations, such as decoding a binary packet or responding to a specific REST endpoint, are done in separate serverless function instances (Lambda for AWS and Function for Azure). So, if one Lambda (or Function) has an error, it won't necessarily bring the rest of the system to halt since the other Lambdas (or Functions) are running independently and new instances can be created automatically to possibly recover from the error.

5.11 Software Tools

For this project, we used a variety of software to assist in different areas of the project. These areas in include software for project management, communication, project development, and documentation. Choosing the correct software tool for the job is critical to ensure that we are able to use our time as productively and efficiently as possible. Without all of these pre-existing tools, we would likely spend far more time on tasks unrelated to working on our design.

5.11.1 Team Communication and Project Management

It is important for teams of any nature to stay organized and have structured and efficient ways of communicating with one another. Before we even got started in the design process, we made sure that we had a tool to organize our project, share files, as well to communicate with one another.

Discord Discord is a free, web-based, team communication platform. Discord supports multiple voice communication and text channels. We took advantage of these feature by creating multiple text channels that were each centered around different topics, such as parts, hardware, and software discussion. We primarily used this tool for general communication, as well as quick file sharing. Our biweekly team meetings,

if we hold them online, are held over Discord. Discord also supports screen-sharing, which was useful when conducting our team meeting as it allowed us to show to each other what we were working on or to help us solve project issues as team. One of the limitations of Discord is its chatroom-style messaging system. While this is useful when having back-and-forth conversations, it is less ideal when wanting to permanently display certain information. In addition to this problem, Discord also has a file-sharing limit of 5 GB. This is where other tools are needed.

Google Drive Google Drive is our primary method of file-sharing and storing project documents. Google Drive is a cloud-based, online storage service operated by Google. This allowed us to efficiently organize documents, such as data sheets, white papers, and any other project related documents. This is also where we stored all of our administrative and financial content, such as the project budget and purchase receipts. In comparison to Discord, Google Drive supports a much larger maximum file size of up to 5 TB.

ClickUp ClickUp is the tool we are using for project management. ClickUp allows users to manage projects through the creation of tasks. These tasks can be presented in multiple different ways, such as an AGILE style board, Gantt chart, and a calander view. Tasks can also have dependencies. This can help show how certain tasks are related to one another. Each task also has its own page where users can post comments, create subtasks, show task history, and create documentation for the given task. For our project, we have only have one main list of tasks that includes everything we are working on. However, in the future, once we get further into actually designing and building the project, we will likely add more lists for different areas such as hardware and software.

5.11.2 Project Development

This section details most of the major programs we used to assist in the development of our project. We did not include supplementary programs, such as text editors or operating systems, as we considered that software to be interchangeable and not specific to this project.

KiCad KiCad is a free and open source software suit for electronic design automation (EDA). The software is used for creating circuit schematics and converting them into PCB designs. KiCad supports the importing schematic footprints designed for EAGLE. KiCad also supports a 3D viewer for PCB designs. There are variety of part libraries online that contain parts from major electronics distributors such as DigiKey. SnapEDA, which is website that catalogs millions of electronic components and includes schematic footprints, symbols, and 3D models, also has support for KiCad. KiCad also has comprehensive official documentation. This will make using the software much easier. In addition, there is a large amount of community support, meaning that if we run into any issues that aren't currently documented, we should be able to find assistance. The large community behind KiCad and free and

source nature of the program is what lead to our decision to use this software for our schematic generation and PCB creation needs. Another benefit of KiCad is the fact that it can be version-controlled with Git. Due to the way in which KiCad stores its project files, they are easily tracked by Git like any other plain text file. This will make it easier to work on our KiCad PCB design asynchronously with the members of our team and integrate nicely into our existing Git workflow.

Zephyr's Meta-tool (`west`) We will be using the west meta-tool that is used in conjunction with our RTOS, Zephyr. West is a command line tool that has a variety of functions. The primary ways that we used west was for building our project and flashing the board. To build a Zephyr project using west, simply run `west build -b aether app`. To flash a Zephyr project to a board, simply run `west flash`. Additionally, west is used to setup the Zephyr environment and install Zephyr. West is also used to keep the local copy of Zephyr up-to-date via `west update`.

Neovim and Vim Neovim and Vim are terminal-based text editors. They are used in the development of all software in this project, including both the firmware and the website. Vim uses keyboard shortcuts that make it faster to use than more popular GUI-based text editors.

STLink We will be using an open source version of STM's programmer, called "stlink" [2] and debugger GUI software that is usable from the command-line, and integrates well with CMake, writing automation scripts, and, of course, Vim. stlink provides debugging and programming capabilities. It also is able to trace code execution, and provide detail information about the debugger and MCU state.

Git Git is a free and open source, distributed version control software. It used by major free and open source software projects and by software companies alike. Git allows us to work on projects asynchronously. Git operates on a series of changes, known as "commits", that make up the entirety of a project, which is contained in a repository (commonly referred to as a repo). A commit is simply a change to a single file or multiple files. Each commit is hashed, which gives it a unique value. New commits that a user makes locally can "pushed" to a remote repository so others can view them. Conversely, new commits from other users can be "pulled" back to a local user's machine. A major appeal of Git is being able to go back in time to an earlier commit, if the need arises.

Another useful feature of Git is the ability create branches. One can think of a branch as simply a side-version of the master repository. Branches are usually used when testing new features or iterating upon the current stable version of a given software. Eventually, when the branch being worked on is finished, it can be merged back into the master branch. For our project, we utilized Git to control our source

[2]Found at: https://github.com/stlink-org/stlink

code for the remote server and the sensor node. We also used it to manage the source code for this report, which was written in .

GitHub As mentioned in the previous section, Git requires a way for users to access the same repository in order to collaboratively work on it. GitHub provides the solution to this problem by hosting millions of Git repositories online, for free. We decided to use GitHub to host our Git repositories for this project because of the fact that it is free and the fact that all of our group members have use it before and are familiar with it. Overall, GitHub allows for easy and free hosting for all of our project's repositories.

Fusion360 Fusion360 is a CAD tool developed by Autodesk. We used Fusion360 to model and develop our enclosure. We also used it to export the relevant files used for 3D printing. It has cloud features that allowed us to easily collaborate and work together on the design.

5.11.3 Documentation

Before beginning our project, we realized that documenting our work would be critical to our success. This meant that it was necessary to properly plan and organize how we would write our documentation. Therefore, we found it necessary to move away from traditional word processors to a document preparation system that would make collaboration easier as well as make writing very large documents a simpler overall task. This made LaTeX and Overleaf the obvious choices.

LaTeX is a document preparation system. Traditional word processors, like Microsoft Word, are referred to as What You See Is What You Get (WYSIWYG), meaning that as you are typing the document you see exactly what the final product will look like. This is in contrast to LaTeX, where you write the document using a markup language in plain text and you only see the final document when you compile the source code. This is why the goal of LaTeX is to allow the user to focus solely on the content and not the formatting. This ends up make the appearance of LaTeX document far more consistent in appearance. LaTeX is also ideal for the typesetting of complex mathematics, which is why it often used in technical papers and documentation. Another reason we decided upon LaTeX for our use in our writing Senior Design reports because it automatically keeps tracking of the numbering of tables figures, ensuring that you never accidental refer to the wrong table or figure from within the text.

Another key feature of LaTeX that factored into our decision to use it was the ease of integration with Git. Since LaTeX operates using plain text files and not binaries like Microsoft Word, this allows us to use the version control features of Git to manage the different versions of our Senior Design reports and to work asynchronously within our team. LaTeX also supports automatic bibliography formatting and styling, making it easy to switch between different formatting, such as APA or IEEE, as well

as easily adding or removing new sources. Overall, due to the size, complexity, and the need to collaborate on the same document with our entire pushed us towards the use of LaTeX for our documentation needs.

Overleaf Overleaf is an online LaTeX integrated development environment (IDE). Overleaf also has a comprehensive library of user-friendly LaTeX documentation. While LaTeX can be installed locally on all operating systems, we decided to use Overleaf to make it easier for some members of our team. Some members of the team had not used LaTeX previously and were not as comfortable with Git, so to make it easier for them we decided on also incorporating Overleaf into our documentation workflow. This allowed them to not have to worry about the sometimes challenging nature of working with the numerous LaTeX packages or getting the correct version of LaTeX installed on their computers and instead allowed them to immediately get into writing our project documentation using LaTeX.

We also were able to integrate our Overleaf project with our Git repository, allowing the members of our group who preferred using their own local LaTeX environment to pull any new changes from the Overleaf document. This made for seamless collaboration between those on our team using Overleaf and those using a local environment.

6 Design

In this section, we present our proposed design for the sensor node, remote server, and gateway. We present this design only after thoroughly performing background research related the Air Quality Index, researching various hardware and software engineering standards, and exploring various potential components. This design combines the information that was collected in previous sections along with our defined requirements and specifications. In order to fully detail our design, we will be both describing our and design as well as including relevant hardware schematics and software flowcharts and diagrams. These will assist in the clarification and understanding of our design.

6.1 LoRa-E5 MCU

The ultra-low power LoRa-E5 (STM32WLE5JC) will be the microcontroller responsible for providing long-range wireless communication from the nodes to the gateway. It's a Ultra-Thin Profile Fine Pitch Ball Grid Array (UFBGA) with 73 ball arrays and a 28 pin layout. The chip is powered through VCC Pin 1 which takes 3.3V from the main power rail. Pins PB15 and PA15 are the microcontroller's serial clock and serial data I2C pins which are used to communicate with the sensors in the system. The respective pins on each sensor will be connected to PB15 and PA15 so sensor data can be read at defined intervals. Pin 15 is the RFIO pin which will be connected to the antenna. Additionally, 3 GPIO pins were used to read the 3 status states of the battery charge controller. Pin 21 (PA9) is connected to the MCP73871 PG pin which indicates the system is receiving power. GPIO pin 13 (PC0) on the microcontroller is connected to pin 8 (STAT1/LBO) on the MCP73871 which indicates to the microcontroller when the battery output is low or the system is charging. The third GPIO used is pin 20 (PB10) on the microcontroller which is connected to pin 7 (STAT2) on the MCP73871 which indicates to the microcontroller when the battery has been fully charged. The nodes will have LEDs to indicate battery status to the users in addition to having the ability to remotely monitor battery status through the LoRa microcontroller.

6.1.1 Memory and DMA

The STM32WLE5JC's flash memory controller features 256 KB of flash with 72-bit read and write bus (8 being for ECC (Error Correctly Control). It also has 64 KB of SRAM with 32 instruction caches lines of 256 bits (1 KB in total) and 8 data cache lines of 256 bits (256 Bytes in total). The upper 31 KB are reserved for system information, option bits, and system memory. The system memory, the largest section of 28 KB, is used to store the boot loader, which is capable to reprogram the flash memory through USART, I^2C, or SPI. The region can be protected through to avoid anyone from being able to be able to write to it. The ability for the MCU to load the bootloader into RAM and use that to reprogram the flash will be important to satisfying one of our stretch goals of being able to do remote firmware upgrades

[56].

6.1.2 Hardware Semaphores

Hardware support for semaphores is key for synchronization amongst multiple threads, such as when using FreeRTOS since it relies on having at least two threads: idle and a single application thread, as described in Section 5.9.2. The STM32WLE has 16 32-bit hardware semaphores, and can be locked using the core ID (COREID) and process ID (PROCID). Once locked, they can only be unlocked with a matching COREID and PROCID [56]. Semaphores will be heavily used in our design for allocating resources (binary semaphores) and signalling threads.

6.1.3 Clock Sources

The microcontroller also has support for a real-time-clock (RTC), which is important for timestamping LoRaWAN packets, required by the LoRaWAN protocol and for location services. The RTC can be used to wake up the devices based on program intervals or at a programmed time. The RTC never stops, even in the lowest power mode, standby. The RTC will be used in the design for waking up the microcontroller from the low-power standby mode. The RTC gets its clock source from an external 32.768 KHz crystal, referred to as the LSE (low-speed external) clock. The other clock, the HSE, (high-speed external) clock, oscillates at 32 MHz, and is used to clock the rest of the chip including the RF module. The RF module is sourced from an internal RC (resistor-capacitor) clock derived from the HSE clock, and oscillates at 13 MHz. During sleep modes, the RF module can opt for the 64 KHz oscillator. Other clock sources include the LSI (low-speed internal), MSI (multi-speed internal - a configurable RC clock), HSI (high-speed internal) 16 MHz clock, and the PLL (phased-lock loop) configurable clock. Several peripherals have specific requirements about which clock sources they can use. As for the MCU core, the core can accept any of the previously mentioned clock sources, but is limited to which source it can use depending on the run mode it's currently operating in. For example, sleep mode (stop mode 0) can only use the HSI clock [56].

6.2 Low Power Modes

Switching between different system operating states will be important to extend the microcontrollers battery life. We have done multiple case studies to show how important it will be to not only keep the sleep power low, but to shorten periods of high power operation, such as when operating the particulate matter sensor or transmitted data over LoRa (the most costly). The microcontroller supports many low power modes with varying power usages and limitations. Table 20 summarizes the low power modes detailed in the data sheet [56]. In all cases, the core clock is off. The current draw was the max current draw at 25 degrees Celsius at 3.0V, and the max clock speed for that mode.

Table 20: The STM32WLE5JC's Low Power Modes

Power Mode	Max Clock (MHz)	Current Draw (μA)	Peripheral Limitations
Sleep	HSE	2100	None.
LPRun	2 MHz	270	PLL disabled, lower RF module power, low-power voltage regulator used for main power.
LPSleep	2 MHz	95	Same as LPRun, core disabled.
Stop 0	HSI16	540	PLL, MSI, HSI16, and HSI32 are disabled. Peripherals not supporting wakeup are power gated. Wakeup capable use HSI on request.
Stop 1	HSI16	6.40	Same as Stop 0, but with slower wakeup time.
Stop 2	HSI16	1.40	Core power domain is disabled. Only some peripherals keep their state. The radio can remain on.
Standby (with SRAM2 and RTC)	MSI 4 MHz	0.345	Disabled voltage regulators, SRAM1 off, only can use RTC and external pin interrupts.
Standby (with RTC)	MSI 4 MHz	0.260	All SRAM contents lost. Otherwise same as Standby (with SRAM2).

6.2.1 I/O Peripherals

In our design, we will using many of the peripherals available on the microcontroller to provide the functionality we need. The following table summarizes the functionalities of the different peripherals we will be using in our design, along with the section we detail its functionality.

Table 21: Used MCU Peripherals

Peripheral	Usage	Section
GPIO	LED status, power subsystem status and control, reset, USB external wakeup	6.5, 6.4
Sub-GHz RF	LoRa and LoRaWAN	6.6
I^2C	Sensor communication	6.3
UART	Communication over USB	6.4
ADC	Battery capacity monitoring	6.5

6.3 The Sensing Subsystem

This section will look at the integration of LoRa-E5 MCU and the three main atmospheric sensors in the Aether nodal system. The MCU in the Aether node will use I2C communication protocol to communicate with the sensors and gateway module. The Aether Node has a Sensirion SPS30 particulate matter sensor, a Rene-

sas ZMOD4510, and a Bosch BME688. The particulate matter sensor is capable of detecting both particulate matter 2.5 (PM2.5), and particulate matter 10 (PM10). Renesas' ZMOD4510 sensor is equipped with non-selective nitrogen dioxide (NO2) measurement capabilities with a selective measurement of ozone (O3) as well. The Bosch BME sensor gives the Aether node the flexibility to measure multiple types of gases including Volatile Organic Compounds (VOCs), volatile sulfur compounds (VSCs) and other gases such as carbon monoxide and hydrogen in addition to pressure, temperature, and humidity sensor through AI and machine learning models. The MCU will be connected to an antenna on the outside of the enclosure to communicate through the LoRaWAN network.

6.3.1 Bosch BME688 Sensor

The Bosch BME688 detects Volatile Organic Compounds (VOCs) and Volatile Sulfur Compounds (VSCs) and other gases through the use of sophisticated AI embedded in the system and machine learning algorithms. This sensor comes in a 3.0 mm x 3.0 mm x 0.93 mm metal lid 8-pin LGA. Aether's Bosch BME688 sensor supports two digital communication interfaces: I2C and SPI. As previously discussed, I2C will be the main source of communication in our overall design and the BME688 I2C interface can support the Standard, Fast and High Speed modes of the sensor. To activate the I2C interface on the Bosch sensor, the VDDIO pin 6 is connected the chip select status (CSB) pin 2. To power the sensor, VDD pin 8 is connected to the 3.3V power rail and tothe VDDIO pin. Both VDD and VDDIO are paired with decoupling capacitors connected to ground to help oppose fast changes of voltage. The pins dedicated to the I2C communication interface are serial clock (SCL), data (SDA), and slave address LSB (SDO). Because SDI is bi-directional, it is connected to VDDIO via a pull-up resistor. Once the I2C communication interface is activated on the chip, the next step is to make the appropriate connections to the LoRa microcontroller. The serial data input (SDI) pin 3 is directly connected to the serial data (SDA) pin 6 on the microcontroller. The serial clock (SCL) pin on the Bosch sensor is connected to pin 5 on the microcontroller. The SDO pin on the sensor is tied to ground for the default slave address.

6.3.2 Renesas ZMOD4510 Sensor

The ZMOD4510 is a 12-pin LGA assembly with dimensions of 3.0 $\times$3.0 $\times$0.7mm equipped with an I2C communication protocol. This is the sensor responsible for detecting nitrogen dioxide and ozone in the the Aether node. The serial clock and serial data pins for the I2C interface on the sensor module is connected to the respective pins on the LoRa microcontroller, pins 5 and 6. VDD is the voltage supply for the ZMOD4510 and VDDH is the voltage supply for the integrated heater in the ZMOD4510. The VDDIO pin is the voltage supply for I/O-interface in ZMOD4510. These 3 voltage supply pins are all connected to the 3.3V power rail. Pin 3 on the ZMOD4510 is the interrupt pin on the sensor. This pin will read HIGH when a measurement is running and LOW when a measurement has finished. It is be connected

to GPIO pin 11 on the LoRa microcontroller. The VSS pins for the ZOMOD are the sensors ground references and will therefore be tied to ground. The ZMOD is equipped with a RES function on pin 11. This is an active low pin connected to the 3.3V power rail via a 10KOhm pull-up resistor.

6.3.3 Sensirion Particulate Matter Sensor

Aether's SPS30 sensor will be responsible for collecting particulate matter data for the node. Unlike the other 2 sensors discussed, the SPS30 will not be soldered directly onto the main PCB board. The interface is a 5-pin JST connector located at the side of the sensor opposite to the air inlet/outlet. Because this sensor relies on a built-in fan to collect ambient particle samples, it is placed at the very edge of the inner housing for the node. Wire headers soldered on the board will then be used to connect the sensor to the 5-pin ZHR-5 connector on the sensor. SPS30 offers both UART and I2C communication. To activate the I2C communication protocol on the SPS30 sensor, the select (SEL) pin must be tied directly to ground. This particulate matter sensor operates with a 5V input, which is the reason a 5V power rail was designed in Aether's power schematic through a boost converter. The VDD pin on the SPS30 is the supply voltage pin that will be powered by this 5V rail. Pins 2 and 3 are the SDA and SCL pins on the SPS30, respectively. The I2C pins are connected to the respective pins on the LoRa microcontroller and tied to 10kOhm pull-up resistors connected to the 3.3V power rail. A 3.3V input was used instead of 5V for the SDA and SCL pins because the pins are Low Voltage Transistor to Transistor Logic (LVTTL) 3.3V compatible. Using 3.3V instead of 5V for the I2C pins ensures no over-voltage damage is done to the microcontroller since it operates under the 5V range. The overall schematic design for the Aether node MCU and sensor system can be seen in Figure 32.

6.4 System I/O

The only I/O that will be exposed from the device will be the USB port. Internally, there will also be a SWD programmer header for ARM programmers and debuggers.

6.4.1 USB-UART Bridge

Another design section of Aether's power system is the IC used to translate USB communication data to UART interface. This is essential because host communication and software implementation will be done through USB input. However, Lora-E5 will only take UART, therefore a conversion IC is needed. This design uses Silicon Lab's USBXpress Family CP2102N chip. The Data(D-) and Data(D+) pins on the bridge are connected to the same pins on the USB and are separated by the protection circuit previously discussed. The TX and RX pins are connected to the USB serial RX and TX pins on the micro-controller. LED diodes are set in place to indicate a proper communication link has been established. The features of the CP210N chip

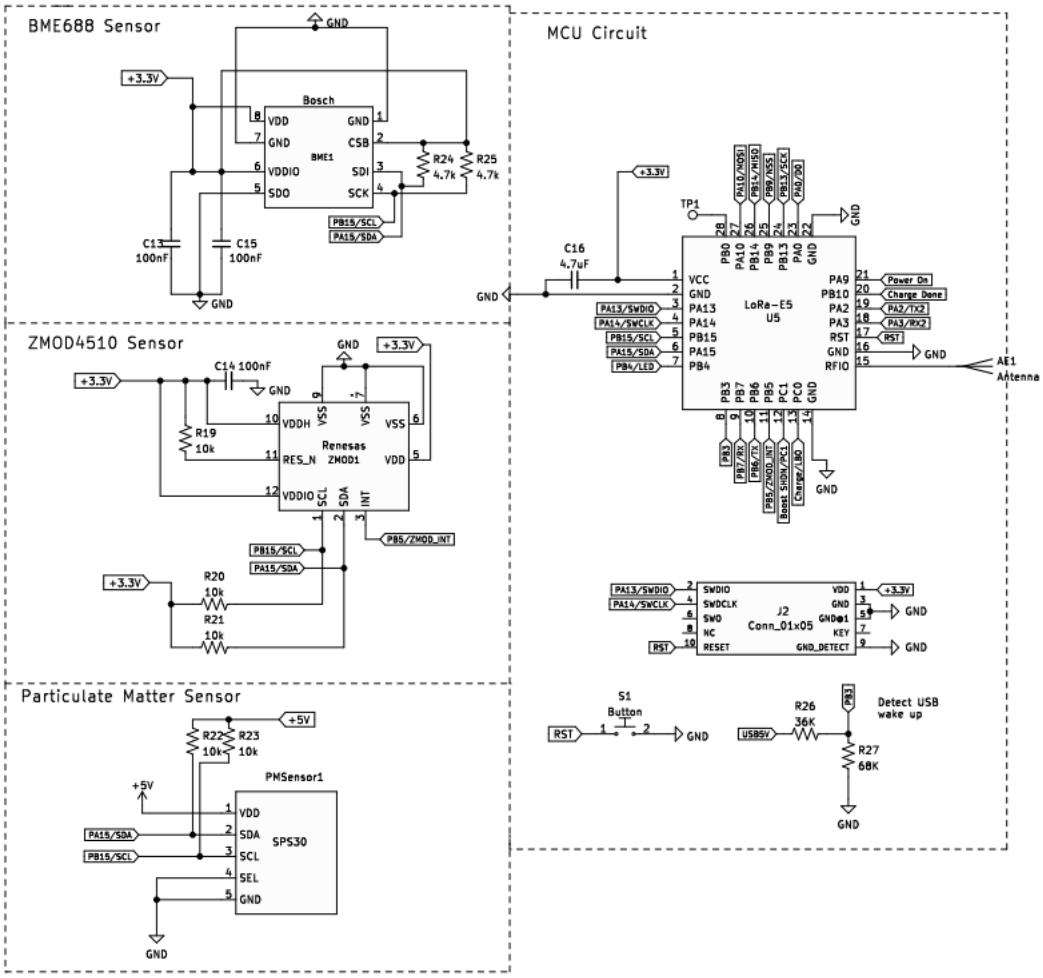


Figure 32: Aether Node MCU and Sensor System Schematic

eliminates the need of firmware and driver development and provides a simple and quick method for USB connectivity.

The next two pins connected on the microcontroller deal with the USB communication interface. As discussed before, this microcontroller is not equipped with the ability to communicate directly through USB. Once the USB-to-UART conversion circuit was designed the TX and RX outputs on the USB-to-UART bridge were connected to 2 GPIO pins on the microcontroller. Pin 10 (PB6) is responsible for taking USB TX data and Pin Pin 9 (PB7) takes in USB RX output. The TX/RX pins are responsible for transmitting and receiving USB data. Pin 12 (PC1) on the microcontroller is a GPIO pin dedicated to the SHDN pin for the boost converter that takes the Li-Io battery as an input and outputs a consistent 5V for the PM sensor. The microcontroller will be responsible for initiating a low-power mode state for the nodal system which shuts down the PM sensor remotely as on of the means to conserve power. Lastly, pin PB3 is an MCU GPIO configured as an EXTI (external interrupt) source. This pin is responsible for detecting when the USB source is connected and instructs the microcontroller to wake up.

6.4.2 The SWD Programming Header

The SWD (Serial Wire Debug) will be used for debugging and programming the device during development, and in the final version of our project. The SWD interface is simpler than JTAG, and only consists of a ground pin, to match the MCU's ground, serial data (SWDIO), serial clock (SWCLK), and a reset pin to drive the MCU in reset in the correct sequence to begin flash programming. The header will use a simple 5-pin 2.54mm male header. Table 22 maps the LoRa-E5 pins to the STM32WLE5JC pins (internal to the LoRa-E5) [56].

Table 22: The SWD Header Pin Mappings [56][57]

Function	LoRa-E5	STM32WLE5JC
Reset	17	RST
SWDIO	3	PA13
SWCLK	4	PA14
Ground	2	GND
Reserved	Not connected	Not connected

6.5 Power System

This section will look at the overall design that will power the nodes in the Aether Sensing Network. The Aether nodes consist of a solar panel capable of powering the system while also charging the Lithium-Ion battery that is used if the load demand exceeds the capabilities of the solar panel. Additionally, the node is equipped with a USB port to charge the battery and power the system as a second option. This gives

users of the design the flexibility to directly charge a depleted battery and maintain the system operational in situations where the solar panel cannot provide sufficient power. A battery charge management controller is be responsible for a voltage proportional charge control feature that is optimize for the solar cell and will also allow the USB input to power the system. Automatic charging current tracking through the battery charge management controller IC is also incorporated in the design. At the output end of the charge controller is a linear dropout regulator (LDO) to drop down the voltage to a constant 3.3V which is used as the main power line for the system.

6.5.1 Overall Features

A 3.7 V/4.2 V Lithium-ion battery is be the main battery source in this design. The battery can be charged through one of 3 methods:

- A 6V solar panel
- USB Input
- A 5V DC adapter in place of the solar panel

The design incorporates automatic charging current tracking through the battery charge management controller IC. This allows the system to operate at high efficiencies with solar panels of varying wattages. Three LED's are used to indicate battery charge status to the user. The orange LED indicates the battery capacity is low and it's charging; the green LED indicates the battery has been fully charged; the red LED indicates power is good and the system is on. A maximum charge rate of 500 mA is outputted by the charge controller and it is adjustable from 50 mA to up to a 1A charge rate.

Our system is designed to draw the most amount of current possible, depending on the set maximum charge rate, from the solar panel to meet load requirements. If the load requires more current it is then turn to the battery. This smart load sharing automatically uses input power if available which prevents the battery from frequently charging and discharging and therefore extending the battery life.

Since the Aether nodes are also intended for outdoor use, battery temperature is a factor that had to be considered. If the Lithium-ion battery temperature exceeds a certain limit, charging it could potentially be dangerous. Therefore, the design included a 10 kΩ NTC thermistor connected to the charge controller IC to monitor battery temperature and automatically stop charging the battery if temperature thresholds are met.

6.5.2 Solar Panel Charging

The nodes in the Aether Sensing Network use Voltaic Systems' 5.5 W 6 V solar panel to charge and power the node. A DC jack input will be used to connect the

solar panel to the system. Charging the system with a solar panel means a filtering capacitor is needed to stabilize the solar panel. A relatively large 4700 microfarad capacitor is used on the PCB for this design. The biggest question of this solar powered system was whether to incorporate Max Power Point Tracker (MPPT) into the design. MPPT works by monitoring the voltage and current output curves of the solar panel in order to maximize the total power output of the panel. Since the voltage and current outputs of a solar panel are dependent on the amount of light outside, solar panel readings need to be consistently monitored. Typically, MPPT is done through the use of DC/DC buck converters so when solar panel voltages vary the buck converter will maintain the current draw to maximize the power output of the solar panel. Although MPPT increase solar efficiency, there were some critical downsides if used in this power design. The first is that DC/DC converters would increase the cost of the design as well as the size and circuit complexity, while only providing a maximum of 30% increase in efficiency to small voltage solar panels. Additionally, when compared to applications with small voltages and currents, MPPT is not much more efficient than a linear converter. A 6V panel responsible of charging an approximately 4V battery will have an MPPT around 5V, since the linear charger will have an input diode with a 0.5V dropout, the inefficiency of a DC/DC converter is about the same as that of the voltage dropout of the diode. Its for this reason that MPPT is often used in systems with 12V lead-acid batteries and very large solar panels requiring multiple chargers. If the panel voltage can be maintained at approximately 1 volt above the battery voltage and the current draw can be kept steady under 1 ampere, near-MPPT performance can be obtained with the right charge controller.

Unlike DC and USB power inputs, voltages and currents are constantly varying in solar panels. The instability of solar panels can cause issues with battery chargers as they constantly turn on and off while trying to draw excessive current from the solar panel causing the voltage to drop. A single solar cell has an open circuit voltage 0.5V when it is not drawing any current. As current begins to increase the voltage begins to collapse and reach 0V at the short circuit current for the solar cell. As light changes the battery charger begins to demand more current from the panel exceeding its short circuit current causing the voltage to collapse and the charger to turn off. Therefore, a charger capable of monitoring the voltage of the solar panel so that it does not demand current past the solar panels short circuit current is needed. With the right battery charger and the capacitor in place to further stabilize the solar panel solar optimization can be achieved. As shown in Figure 33, the connection for the solar panel can be seen. A Schottky diode charges a 4700uF capacitor for the solar panel, the diode prevents any charge in the capacitor to drain back to the panel.

6.5.3 USB Power Input

Aether nodes also have the capability of being powered and charged through direct USB-C input. In certain conditions we have found having another potential source of power can come in handy. Namely in cases where solar power is no longer functional or in cases where we want to manually charge the battery. As a result,

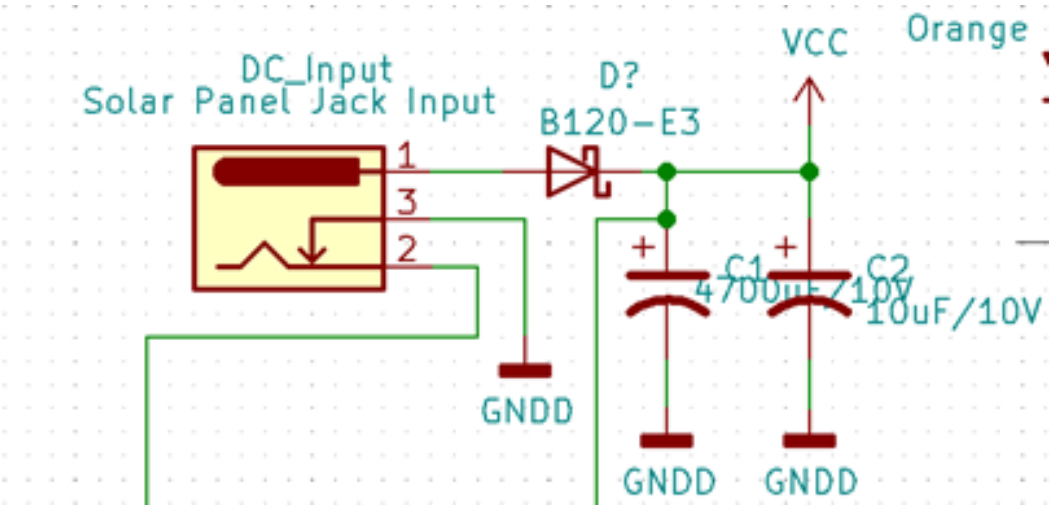


Figure 33: Schematic for Solar Panel DC Input

we have found USB charging the most efficient way to accomplish this. This design incorporates USB input to the MCP73871 main input pin. The system operates depending on which power input is selected. The design will incorporate a USB-C input. This source of power is easy to construct, maintain, and also reliable. The node takes on power from a USB source where an IC will then take in the power and effectively convert it to appropriate levels to supply the load and charge the battery alone or in conjunction with our solar source. While the chip is being supplied with the USB power, the MCP73871 has additional precautions monitoring the output current to the battery and load ensuring there's over current protection from the USB power source. USB charging, on average, supplies its demand with 2.5 watts of power, should this be maintained through effective usage of the MCP73871 we could find days of charge on the battery in only a matter of hours while simultaneously powering our system.

The second major function of the USB-C power input is to serve as a communication link to the Lora-E5 micro controller. Not only has USB become a standard for charging devices but USB has also become an industry standard interface for transmitting data. The power design of this circuit includes a USB-C port connector connected to a USB-to-UART Bridge IC. This IC is necessary since the Lora-E5 micro controller can only communicate through a UART protocol and not USB. A crucial factor to consider when adding a USB system that interfaces with an external environment such as in Aether, is electrostatic discharge (ESD) and other forms of electrical interference. The Aether power system design features an ESD protection circuit integrated with electromagnetic interference (EMI) protection as well. Protection circuitry will be vital to ensuring uninterrupted operation of the USB communication interface and providing ESD discharge robustness to the USB system. Although ESD risk are present throughout all USB systems, particular caution needs to be taken when using USB-C. The USB PD high power delivery in USB-C in addition to its smaller more compact design, place the system at more of ESD related issues. Pro-

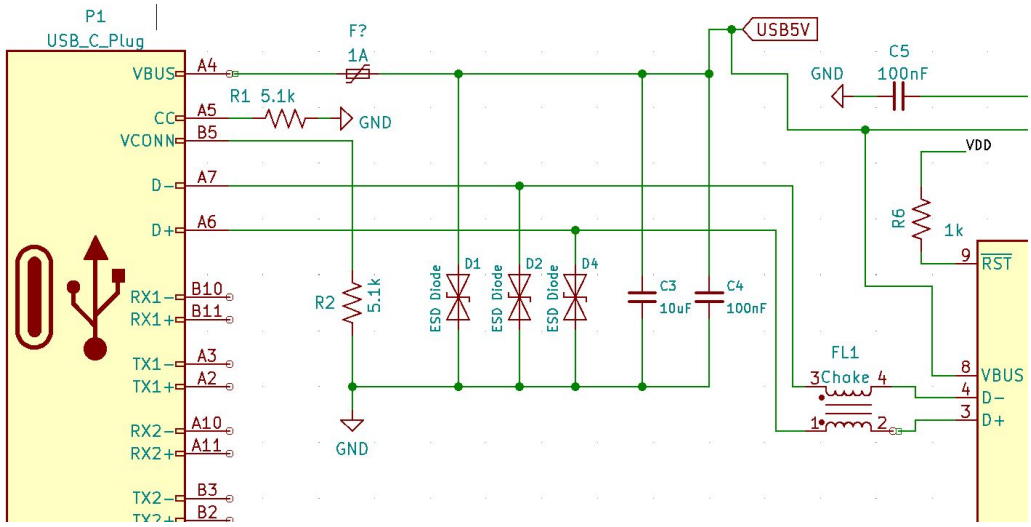


Figure 34: USB-C Protection Circuit

tection circuitry also prevents damage to downstream components that may result from short circuitry between the connector's adjacent VBUS and CC or SBU pins. There are several different circuit design methods that can be implemented for ESD and EMI protection. Aether's USB power design uses an integrated EMI suppression with ESD protection. The principle components used in this protection circuit are bidirectional TVS surge protection diodes, a choke filter, and capacitors. The TVS diodes are used to limit fast transients at low voltage levels while the choke filter attenuates high frequency and fast voltage peaks to protect against common-mode disturbances. When placed appropriately, capacitors play a crucial role in minimizing EMI in this design by forming low-pass filters bypassing the RF signal to ground. A 1A fuse is placed on the line out of VBUS to protect from over voltages. With the appropriate protection in place, the USB output of 5V can be sent to the charge controller circuit in to charge and power the system. The protection circuit designed can be seen in Figure 34.

6.5.4 Aether Power Schematic

Charge Controller The heart of this power system design is the solar/USB charge controller MCP73871. One of the biggest concerns with designing a solar-powered system is preventing the voltage from collapsing when there is a high current draw from the load. A charger is needed that is capable of monitoring the solar panel voltage and ensuring demand stops when the voltage begins to drop. This prevents the battery from constantly charging and discharging. The MCP73871 uses a pin called VPCC to achieve this. Depending on the voltage input to this pin, the charger makes sure to draw as much current as it can from the solar panel while adjusting the charge rate to maintain a solar panel voltage above the voltage of VPCC. For this design, a VPCC voltage of 4.2V, just above the Li-Io battery voltage, was used. To do this, a voltage divider was implemented using a 270 kΩ and 100 kΩ resistor.

Pins 6, 7, and 8 on the MCP73871 controller are used as system status lines. LEDs along with respective resistors are connected to these pins indicating either a low battery output/charging state, a fully charged state, and a power on state. . In accordance with the data sheet of for the MCP73871, pins 3, 4, 9 and 17 were connected to power. PROG1 or pin 13 on the chip layout, is used to determine charge rate of the controller based on the formula of 1000V divided by the resistor value connected to PROG1. In this design, a 2 kΩ resistor was used, setting a charge rate of 500 mA. This charge rate can be adjusted depending on the resistor value used. THERM pin 5 on the IC is used for the Li-Io temperature monitoring feature of the system. By connecting a 10 kΩ NTC thermistor to the charge controller it is able to detect of the battery temperature is too hot or too cold to safely and effectively charge. The battery is connected to battery pin 14 and 16 on MCP73871.

Lastly, pin 1 is the output pin of the charger where the system load is connected to. Most of the micro-controllers and devices used in the Aether node network ideally operate at 3.3V. Therefore a 3.3V output rail from the charger would be most desirable. However, because the solar panel supply voltage can vary up to 6V in bright daylight conditions, provisions needed to be set so that the micro-controllers did not receive a voltage input above their max rating. This was achieved by using the LP2985-3.3 linear dropout (LDO) regulator. By properly connecting the pins with the appropriate capacitors listed in the data sheet for the LDO,varying output voltages from the MCP73871 could be converted to always output a consistent 3.3V rail. This 3.3V rail will serve as the main power rail in the system. When higher or lower voltages are needed for other system components, the appropriate circuitry will be set in place to achieve the desired voltage. The overall Aether solar/USB charging and power circuit can be seen in Figure 35.

Boost Converter A 5V input is needed to properly power Sensirion's Particulate Matter Sensor. The main power line in the circuit is 3.3V since that's what the majority of the chips use. To get a 5V output the LTC3525-5 DC/DC Boost converter was used with the Li-Io battery as the source voltage. The battery was used for Vin instead of the 3.3V power rail because it provides greater efficiency. Perhaps the most crucial feature of this power subsystem is the remote shutdown feature that is used to turn off the converter, essentially turning off the particulate matter sensor. The SHDN pin is connected with a pull-down resistor to a GPIO pin on the micro controller that will be able to remotely deactivate the sensor when the system wants to enter a low-power operating mode.

6.6 Embedded Software

This section will cover our design for the firmware running on the Aether Node's microcontroller. Due to the need to run various different tasks in parallel, we will be utilizing Zephyr to aid in more effectively scheduling them. The design of the firmware is best thought of in terms of three separate control loops. Those controls are the sensor control loop, the LoRa control loop, and the serial command control

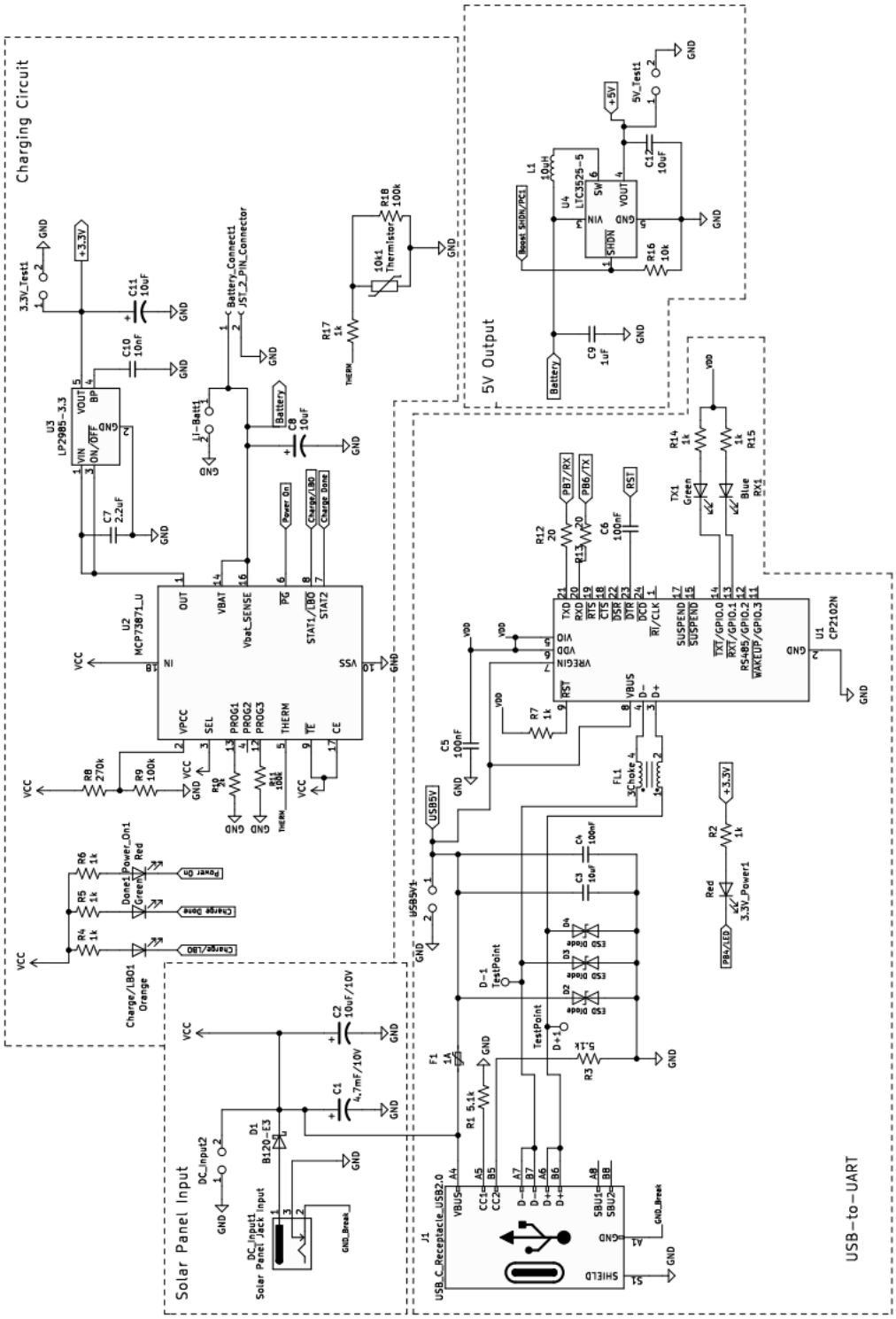


Figure 35: Schematic for Aether node power system

```
const struct device *dev_bme;
dev_bme = DEVICE_DT_GET(DT_NODELABEL(bme688));

while(1) {
    sensor_sample_fetch(dev_bme);
    sensor_channel_get(dev_bme,
                    SENSOR_CHAN_AMBIENT_TEMP,
                    &temp);

    k_msleep(DELAY_15_MINUTES);
}
```

Figure 36: A code snippet that shows the basic structure of the sensor reading loop. This example is for the BME688 and only shows getting a temperature reading.

loop. Additionally, there is a power management component that is implemented in the firmware. This power management involves putting the LoRa-E5 in a low-power state when the system is idling and cutting the power to the SPS30 when it is not taking readings.

6.6.1 Managing Sensors

The sensor control loop is responsible for collecting data from the three sensors used by the system (BME688, ZMOD4510, and SPS30). A thread is dedicated to each sensor for the execution of this control loop. Due to the fact that each sensor follows the same control flow, Figure 37 depicts each sensor thread running the same tasks in parallel.

The following description of the sensor loop follows the sample code seen in Figure 36. When the thread is started by the kernel, it begins by initializing the sensor. Next, the sensor takes a reading. In Zephyr, this is done by first calling `sensor_sample_fetch(dev)`, which calls the appropriate driver function to take a sensor reading. The results from that function are then stored in a sensor channel variable. To access that value, a call to `sensor_channel_get()` is made. After accessing all of the sensor readings for that sensor, the results are converted to the CayenneLPP[3] format and placed in the message queue, where the data will later be read by the LoRa thread. Finally, the thread is put back to sleep by calling `k_msleep(DELAY)`. In the case of our final design, this delay was set fifteen minutes. This also allows the kernel to schedule other threads to run.

6.6.2 LoRa Control Loop

This is the control loop that handles sending collected sensor readings to our server over LoRaWAN. It begins by first checking if there are any messages containing

[3]This is a simple data packet structure designed for storing sensor data. It follows the format: 1 byte for sensor type, 1 byte for data type, and then n bytes for the data payload.

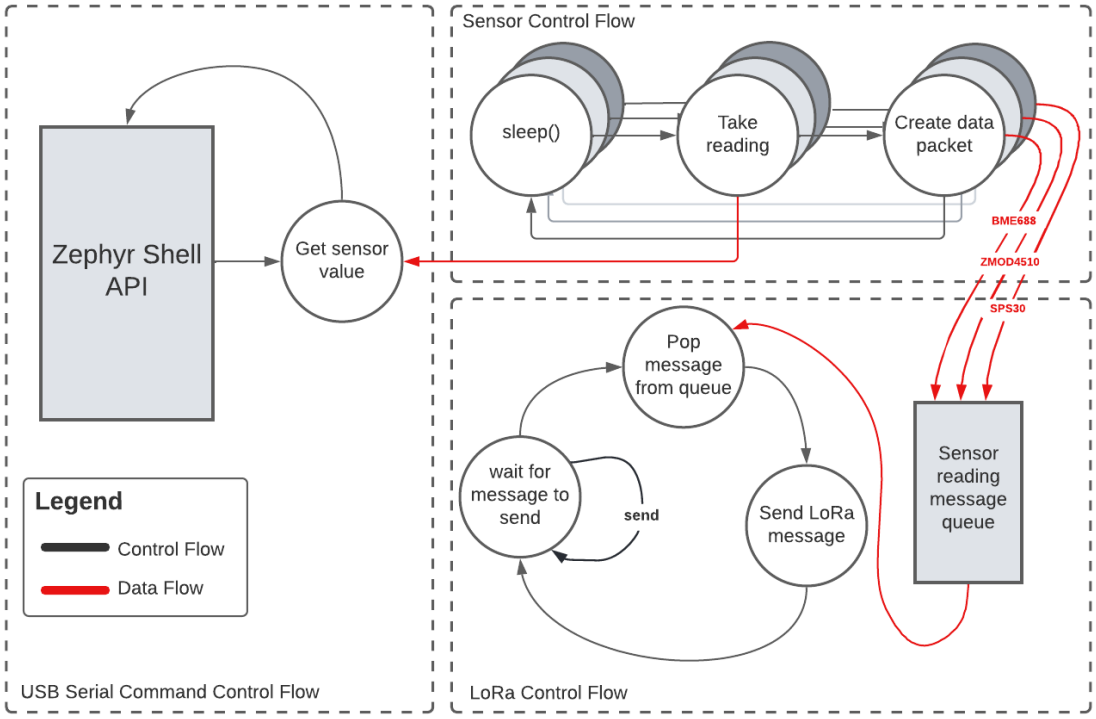


Figure 37: The flow diagram for the Aether Node firmware.

sensor data in the message queue. If the message queue is empty, the thread yields until notified by another thread when a message is inserted into the queue. If there is a message, it removes it from the queue, and as many messages that can fit within the LoRaWAN packet as defined by the current data-rate, and then attempts to send the LoRaWAN message. The loop then waits for the message to send successfully. If it does not send successfully the first time, the thread will attempt to send the message up to 10 times.

6.7 Serial Command Control Loop

This is the control loop that handles accepting user commands from the UART shell. As shown in Figure 37, the design design was able to use Zephyr's built-in shell API. Adding the shell to the firmware only involved adding a few options to the project configuration file. Regarding the functionality of reading data from connected sensors, the default Zephyr shell already included the ability to read sensors connected to the I^2C bus.

6.7.1 Power Management

Power management was necessary to allow the Aether Node to operate for a longer period of time. The two primary power management features implemented in the firmware was power-gating the SPS30 and putting the LoRa-E5 in a sleep mode

when not taking sensor measurements.

The power-gating of the SPS30 was implemented by using the Zephyr power domain API. The power domain toggles a GPIO pin on the MCU that toggles the power to the SPS30. Before calling the sleep function in the SPS30 thread, a function is called to turn the power domain off. Similarly, before calling `sample_fetch(dev)`, the power domain is turned on.

The LoRa-E5 was put into a sleep mode when not taking sensor measurements. This was implemented using the Zephyr power management API. This involved setting up multiple power states by defining them in the Devicetree file. We defined both `enable_sleep()` and `disable_sleep()` functions using the appropriate API calls. This allowed us to call `disable_sleep()` before starting to sensor reading and then call `enable_sleep()` before putting the sensor thread back to sleep.

6.8 Enclosure Design

Our sensor node design will require a physical enclosure to protect the sensitive electronic components from damage from various harsh outdoor conditions. To accomplish this designed a simple 3D printed enclosure out of a plastic material.

The material used is known as poly ethylene-glycol (PETG). The reason we used this material for our enclosure is due to the fact that the material is naturally UV resistant. This means it will not degrade over time and become brittle when exposed to direct sunlight over long periods of time. This is critical since our design is intended to be used outdoors, especially in Florida where sunlight is common and intense. PETG also has a high tensile strength that is comparable to ABS, another common 3D printing material. Another benefit of PETG is that it is recyclable and biodegradable, helping to reduce the impact of our design on the environment.

The enclosure was designed with the goal of being able to be mounted on a vertical wall or other flat surface. Holes were included for the USB-C connector, solar panel connector, and antenna. Additionally, specific intake and outtake holes were created for the SPS30, in accordance with the guidelines specified in the datasheet[50]. Two extrusions were made on the bottom part of the enclosure to keep the SPS30 from sliding horizontally within the enclosure. A third extrusion was made coming down from the cover to keep the SPS30 from moving vertically. The design includes four holes on each corner for inserting screws, in order to lock the cover in place. The full enclosure design can be seen in Figure 38. The 3D CAD model of the design can be seen in Figure 39.

6.9 Gateway

This section we will be detailing how we will use a LoRaWAN gateway in our design. We will not be designing the hardware in the same way that we are designing the hardware for the sensor node i.e. designing the PCB, etc., but we will need to build

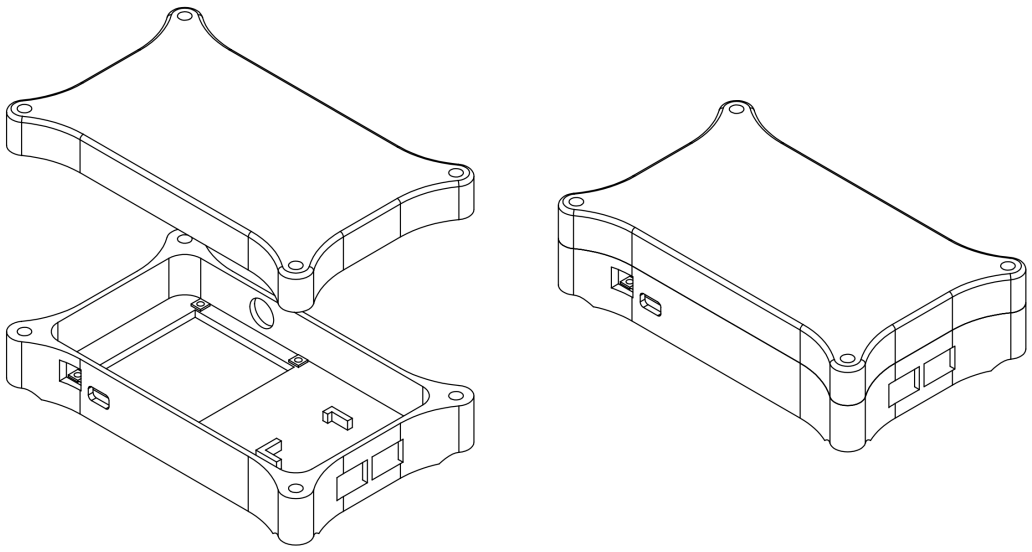


Figure 38: An exploded, isometric view of the enclosure design.

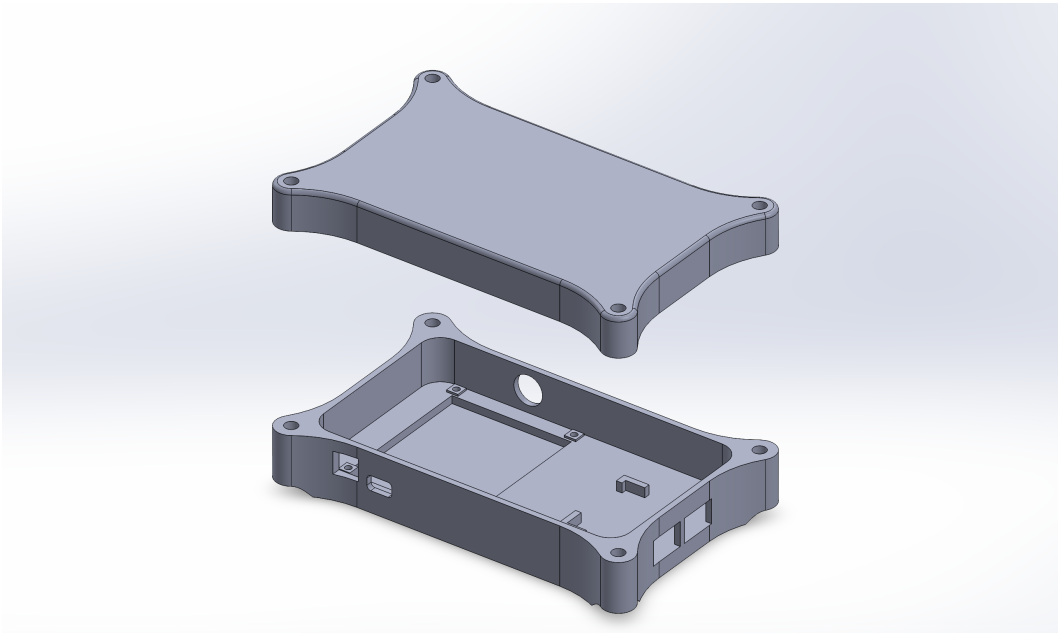


Figure 39: A 3D rendering of the enclosure design.

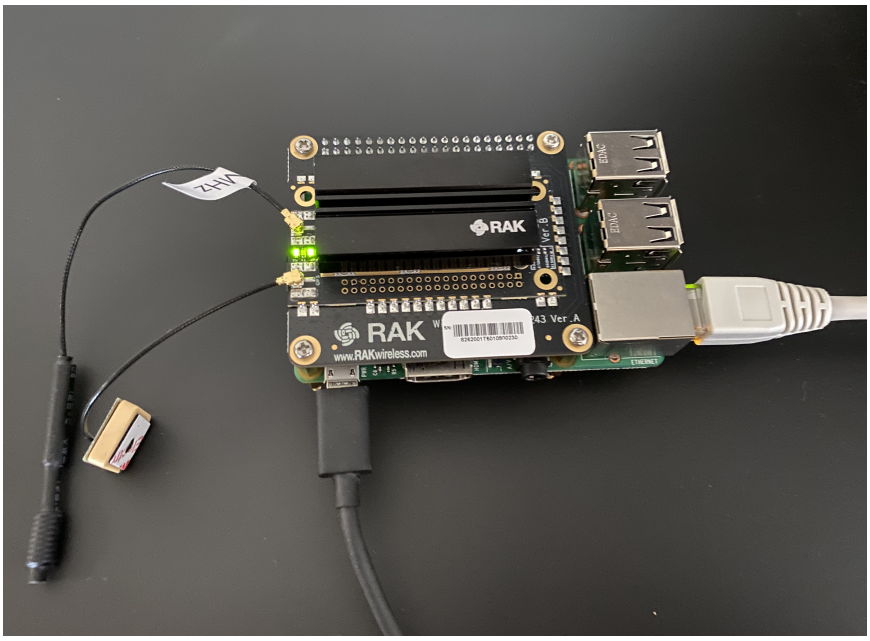


Figure 40: An image of our setup gateway. It shows the Raspberry Pi 4 connected with the RAK 2245 Pi Hat.

a functioning LoRaWAN gateway in order for sensor node to be useful. This section will focus on what we will need to do to set up the physical gateway components, which are a Raspberry Pi 4 and a RAK 2245 Pi Hat. We will also discuss how the software will need to be set up on the gateway in order to be able to connect it to our AWS instance using AWS IoT Core and The Things Network.

6.9.1 Gateway Hardware

The hardware assembly for the gateway is incredibly simple. All that needs to be done is to attach the RAK 2245's connector to the Raspberry Pi's 40-pin GPIO header. After this, it is simply a matter of connecting the Ethernet output to an internet router and plugging in the micro-USB cable for power. Additionally, the GPS antenna and the LoRa antenna that come with the RAK 2245 must be attached to it in order for the gateway to function. The only other component that is required is an SD card loaded with a Linux image developed by RAKWireless designed for the gateway. Our setup gateway can be seen is Figure 40.

6.9.2 Gateway Software

Configuring the gateway to function with the various LoRaWAN networks we desire is also fairly straightforward. Once we burn the previously mentioned OS image to an SD card and insert it into the Raspberry Pi, the device will create a new WiFi access point. On a separate computer, we will be able to connect to this WiFi access point and then can gain access to the gateway via an SSH connection. Within this SSH connection is where we will be able to configure that gateway to connect

to any of the LoRaWAN networks we choose. In the next few sections we will detail how we will connect our gateway to The Things Network and directly to our AWS server instance.

Connecting to The Things Network Connecting to The Things Network is simply a matter of configuring the LoRaWAN gateway and using The Things Network's online configuration tool to add your gateway to the network.

To configure the gateway itself, one needs to connect to the gateway via SSH and then run the `gateway-config` command as follows. Also, run the `gateway-version` command. The information from this command will be used to add the gateway to the network.

```
$ ssh pi@192.168.10.10
$ sudo gateway-config
$ sudo gateway-version
```

This will bring up a menu GUI. In this menu, you select option 2, "`Setup RAK Gateway LoRa concentrator`. Then select the server plan, which is `Server is TTN`. Next, you select the appropriate region and frequency region, which in our case is `US_902_928`. After this, configuration on the device itself is finished. We will need to move to the Things Network's website next.

To finish setting up our gateway with The Things Network, we must navigate to The Things Network's website[4] and create an account. Once logged in, navigate to the Console, select your region, and then click "Add gateway." Once there, fill out the appropriate information for the gateway including the `Gateway ID` and appropriate frequency plan. After completing this, any LoRaWAN end-device that is correctly configured to connect to The Things Network should be able to see your newly setup gateway and use it to connect to the internet.

Connecting Our AWS Server Instance Connecing our gateway to an AWS server instance will be somewhat similar to how we set it up to connect to The Things Network. It will require installing a custom operating system on the Raspberry Pi and performing some configuration on both the Pi and on AWS IoT Core, which is what we will be using to manage the gateway.

To begin configuring the Raspberry Pi, we are going to use Basic Station[5], which is an open-source LoRaWAN gateway implementation. This will be installed on the Raspberry Pi. In order to use Basic Station with the RAK 2245, we have to make a few changes to the operating system. We must remove the following libraries:

[4]The website for The Things Network is `www.thethingsnetwork.org`

[5]The source code for Basic Station can be found at `https://github.com/lorabasics/basicstation`

```
$ rm ./build-rpi-std/lib/liblgw.a
$ rm ./deps/lgw/platform-rpi/libloragw/libloragw.a
```

We also need to modify the `SPI_SPEED` variable from within the `loragw_spi.native.c` file. We are then ready to compile Basic Station by running the following command:

```
$ make platform=rpi variant=std
```

Once compiled, we have to create an `init.sh` script as well as directory to store configuration and credentials (keys, IDs, etc.) for AWS IoT Core. The code for the `init.sh` script is provided to us by AWS and is simply a matter of copying and pasting the script into the home directory of the Raspberry Pi. Creating the directory is just as simple as running `mkdir`. Once this is done, we will then run the `init.sh` script. We can then start Basic Station with the test server and credentials using the following command:

```
$ cd $HOME/basicstation/examples/live-s2.sm.tc
$ RADIODEV=/dev/spidev0.0 ../../build-rpi-std/bin/station
```

We will use the output of these commands to determine our gateway's `EUI`. To begin the process of connecting our gateway to AWS IoT Core, we will navigate to the AWS IoT Core Management console. From there, we will click "Add a gateway." This is where will provide IoT Core with our gateway's `EUI`. Then, IoT Core will prompt us to generate gateways certificates. This will ensure that our gateway can connect to our server securely and verify it's identity. We will have downloaded the following files, which should then be copied to our Raspberry Pi and placed in the configuration directory we created earlier:

- `nnnnnnnn-nnnn-nnnn-nnnn-nnnnnnnnnnnn.cert.pem` - This is the gateway device certificate file
- `nnnnnnnn-nnnn-nnnn-nnnn-nnnnnnnnnnnn.private.key` - This is the gateway device private key file
- `lns.trust` - This is the trust certificate for the LoRaWAN Network Server (LNS) endpoint
- `cups.trust` - This is the trust certificate for the CUPS endpoint

We will also need to rename the `.cert.pem` and `.private.key` files to `cups.crt` and `cups.key` respectively. Once all of this is completed, we should end up with the following files within our configuration directory:

- `cups.trust`

- `cups.uri`
- `cups.crt`
- `cups.key`
- `station.conf`
- `version.txt`

We can then start up the Basic Station software using the command we ran earlier. At this point, the Basic Station software should begin to establish a connection AWS IoT Core for LoRaWAN. Additionally, if we have our end-device connected to AWS IoT Core, we should be able to see log entries every time the device is sending packets through the gateway.

6.10 The Server Application

Initially, we had chosen to use AWS for the entirety of our backend. In this section we will go over the initial design with AWS, the challenges we faced, and why we ultimately decided to move a majority of our backend to Supabase.

6.10.1 The Design at a High Level

Based on our requirements of being able to receive LoRaWAN messages, store the sensor readings, and display them on a web application, we came up with the following requirements for our backend:

- Handle LoRaWAN uplinks
- Host a website
- Host an API service
- Store data somewhere

Data Flow It is also important to design how data will flow throughout a backend design. The requirements above translates to the following data-flow:

1. The Aether Node successfully sends a LoRaWAN packet consisting of one or more sensor readings in the cayenne format to the gateway
2. The gateway connects to the LoRaWAN network server and forwards the packet
3. The LoRaWAN network server forwards the packet to the application server
4. The application server's uplink handler ingests and decodes the base64-encoded,

cayenne-formatted binary packet into individual reading(s)

5. The individual readings from the packet are stored and processed (calculate AQI, check user alerts, etc)

6. The updated data is displayed on the web application

Backend Components The backend is responsible for implementing numbers 3, 4, and 5 in the data flow enumerated above. The main components to implement the data flow parts are:

1. Uplink handler - A method to handle uplink packets and store them in a database

2. Alert handler - A method to trigger and send alerts

3. A database

4. A static website

5. Website domain certificates

6. Client facing API

Choosing a Database As for storing data, we made the early decision to use a relational database (RDS). Specifically, we chose PostgreSQL since it's open source and has a huge ecosystem of extensions, such as PostGIS for geometry and geography math. Another big reason for initially choosing PostgreSQL was Postgraphile - a GraphQL server that could introspect your PostgreSQL database schema and automatically create a CRUD (create, read, update, and delete) API. This would save tons of development time and let us focus on other more prominent aspects of our project, such as the website, firmware, and PCB. Also, using a RDS over a document store, or NoSQL, was made for three reasons: *i)* IoT data is inherently relational. Each reading originates from a sensor on a specific device which has a particular location at the time the reading was taken at. And, each device has a current location and belongs to a specific user, etc. *ii)* Our prototypical use case does not demand high scalability. NoSQL databases you to optimize your access patterns and schema specifically for NoSQL to get high parallel performance. We don't know all of our access patterns and how the schema might evolve over the course of the project. Using a SQL database is more than enough, and allows us a lot of flexibility. *iii)* We wanted to learn SQL as it is widely used in across industries and is a valuable skill to have.

6.10.2 The Initial AWS Design

Initially, the entire backend infrastructure was on AWS. The infrastructure was defined and written in Typescript using AWS' Cloud Development Kit (CDK), which is an Infrastructure as Code (IaC) approach to defining, configuring, and deploying AWS cloud resources. The overall design is shown in Figure 41.

Now, we will explain each component in the AWS design.

The LNS The LoRaWAN network server was hosted on AWS using the IoT Core for LoRaWAN service. The gateway would be configured to run LoRa Basics Station and to forward packets to AWS. Beforehand, the CUPS, public, and private keys were created and distributed to the gateway to authenticate the gateway to connect to our LNS instance. Devices and gateways are added under IoT Core for LoRaWAN, and are created IoT Core Things, which store additional metadata for that particular device. IoT Core for LoRaWAN was configured to integrate with AWS Lambda to forward all uplinks to the uplink handler Lambda function.

The Uplink Handler The uplink handler was designed as a Typescript Lambda function which would decode the base64-encode, cayenne-formatted payload. It would then connect to our PostgreSQL database, and store the readings. It was created to operate inside of a new virtual private cloud (VPC) that was used for all other infrastructure in order to connect to the database.

The Database A PostgreSQL version 13 database was instantiated through AWS RDS, and was specified to operate inside of the VPC previously mentioned. The initial entity relational diagram (ERD) is shown in Figure 42.

The Static Website The website is simply hosted inside of a S3 (simple storage solution) bucket. However, public connections are blocked. Instead, we have configured CloudFront to accept public connections and deal with loading data from the S3 bucket. CloudFront is AWS' content delivery network (CDN), which is a worldwide cache for accessing web content.

Website Certificates The domain for our website, `aethersensor.network`, was bought through AWS using the Route53 service. The hosted zones were configured to have a subdomain for the API, `api.aethersensor.network`, link it to the main domain, and to forward to corresponding traffic to it.

GraphQL API There are many factors that went into choosing GraphQL over REST that ended up proving to major pain points. One of those was that we wanted to avoid writing an API from scratch. For this, we decided to use Postgraphile. This was before we learned about PostgREST, a PostgreSQL extension that created a REST API from your database, akin to Postgraphile. However, we thought GraphQL

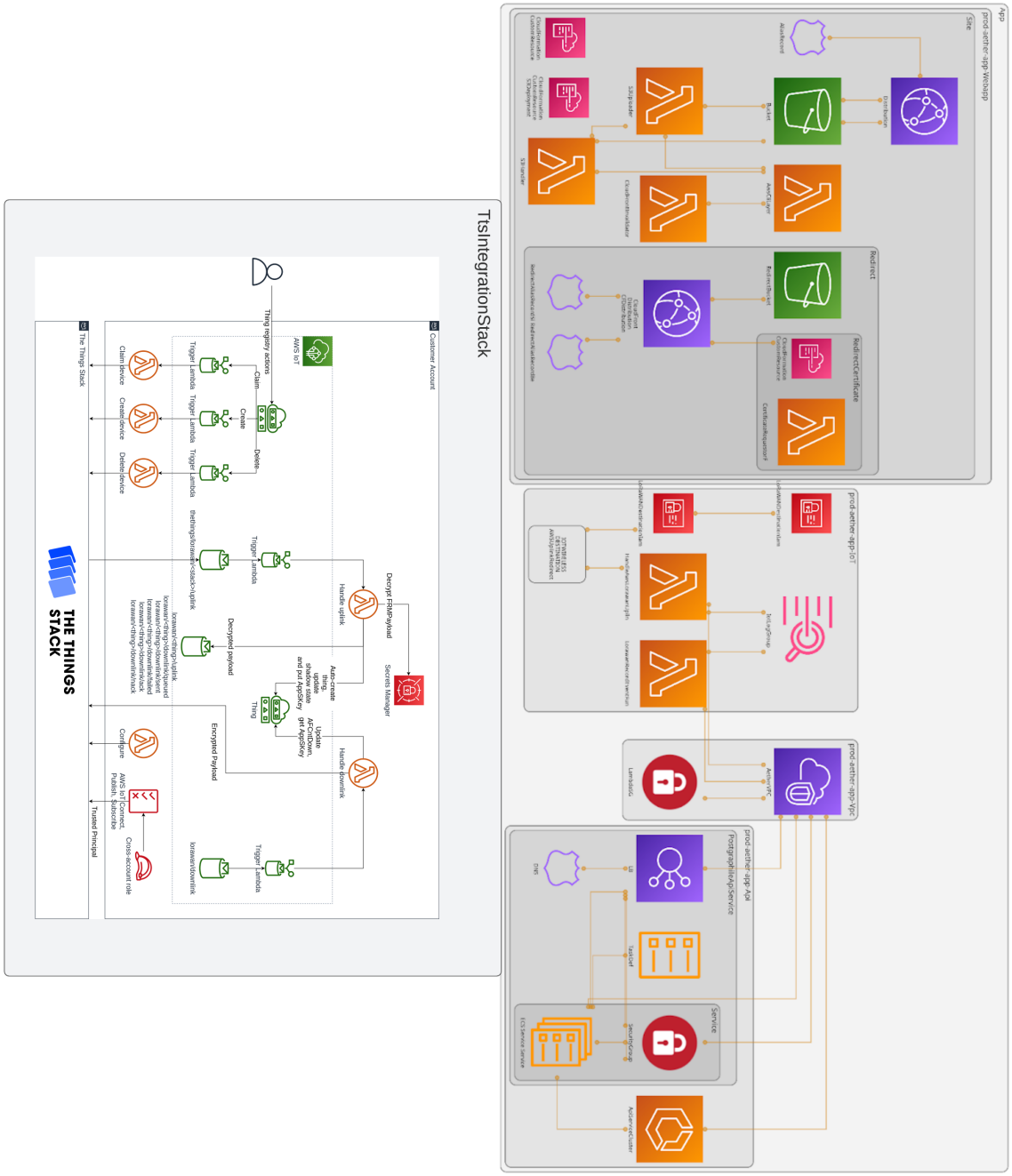


Figure 41: Initial AWS Design

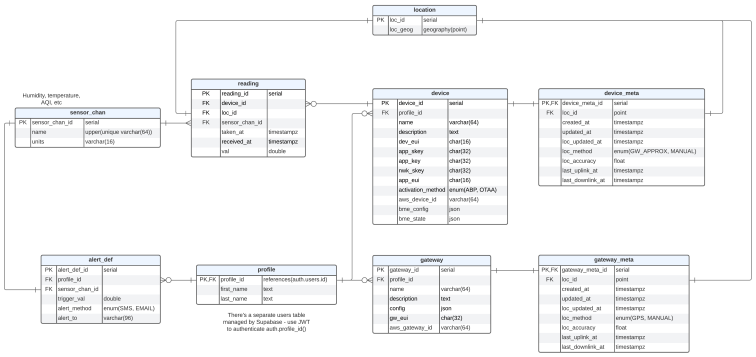


Figure 42: Initial ERD

would be better for our application since we have many types of data to view on the web application and it might show up all over the place. For instance, the sensor readings can be displayed on the map, in a graph, or in logs. Each of those readings is associated with a device and we may want to view all readings for a particular device or within a geographical area. However, through practice and as later talked about in the Supabase design, this was due to not fulling understanding the web application access patterns.

The GraphQL server was hosted on a EC2 instanced and managed by AWS Fargate, a service for launching Docker containers.

6.10.3 AWS Challenges

There were many challenges with using AWS ranging from surprise bills, time to get something working, and general costs of operation. AWS is well known for its surprise bills due to its incredibly complicated billing structure. In many such cases, it's easy to veer outside the free-tier without knowing it unless you kept a close eye on your resource usage. This was exacerbated by using the CDK IaC framework since some CDK components would spawn other resources when deploying something else.

Also, on the topic of costs, AWS can get costly very fast. They are very good at locking you into using their service and making it opaque on what a free-tier even means. For instance, we had originally used PostgreSQL version 13, but the free-tier specified in fine print that they only offer version 12. So, wondering why the free-tier wasn't working, amongst with having to manually manage a database made us search for other options. This is where we came across Supabase, the managed and pre-configured PosgreSQL database. We started prototyping on Supabase, but in order for Lambda functions and EC2 instances to access the internet, they need to first send their traffic through a NAT (network address translation) service. This service has no free-tier, and while we should've verified beforehand, we had started racking up bills. But, it was too far into development, and we were making far more progress on Supabase, so we decided to stay with Supabase. In the end, we shut down all infrastructure on AWS besides anything pertaining to the static website.

As mentioned, AWS also takes a long time to learn and develop on. However, it is extremely flexible in what you can make, granted there's the time and money. This is why services such as Supabase and MongoDB will create services that are actually deployed on AWS or Azure. When trying to implement email alerts, it proved to be too complex for our use-case. Therefore, we sought after a transactional email service. A transactional email service is a emailing service with a simple REST interface. Regarding IoT Core for LoRaWAN, it is completely doable to build an application with it given more time. However, for such a small team with the time we had for this project, it proved to be too much to implement (learning the complex API for managing devices, gateways, things, LNS configuration, etc). Therefore, we opted to use The Things Stack, which is a free and managed solution.

6.10.4 The Design with Supabase

Supabase is an open source, managed PostgreSQL service. They provide additional services, such as OAUTH2 authentication, real-time updates over websockets, edge functions, which are similar to Lambda functions, and storage. While just otherwise a PostgreSQL database, it can be quite powerful, especially with extensions which Supabase provides pre-configured. They also provide an isomorphic JavaScript client for accessing the database through REST (PostgREST extension), triggering edge functions, accessing storage, subscribing to tables (real-time), and authentication. We had the following extensions enabled in our design (default included):

- HTTP: Make HTTP requests from inside PostgreSQL
- PLV8: Write PostgreSQL functions with the JavaScript
- POSTGIS: Geometrical and geographical spatial types and operators

The LNS Since, IoT Core for LoRaWAN was not used for the final design, TTS was used instead. This integrated smoothly with Supabase since The Things Stack had webhook integration, which is just a fancy way of saying POST requests to some specified endpoint.

When transitioning the backend from AWS to Supabase, we had to migrate the LNS once on IoT Core for LoRaWAN to The Things Stack (TTS) and implement all of the logic in the database. Which proved to be its own challenge, but immensely simpler than AWS. However, for our use-case, the overall design wasn't overly complex to do in the database, but it's still to complex to show the entire ERD. Instead, the overall data-flow is shown in 43.

The Uplink Handler Upon being processed in The Things Stack, the packets are packaged into JSON blobs and POSTed to the `handle_lorawan_uplink` database function exposed as a REST endpoint that gets executed as a remote procedure call, `/rest/v1/rpc/handle_lorawan_uplink`. The uplink handler reads the The Things

Stack uplink event JSON file and decodes the Cayenne-formatted, base64-encoded payload using another database function, but this time written in JavaScript using another Postgres extension. Postgres has extensive support for JSON and JSONB (binary JSON) data types and makes it easy to work with, since there are special operators and functions for manipulating JSON blobs. The data-flow for this can be seen in Figure **??**.

Once decoded, all sensor samplings are inserted into the `reading` table in the database. Triggers are setup to check for alerts, update device and user metadata, and update the hourly values for each sensor channel in a table called `hourly_reading_stats`. Finally, once the hourly statistics are updated, a trigger updates the hourly raw AQI. The raw AQI is the AQI directly calculated from the hourly reading statistics, and does not perform any linear regressions. There is also a trigger on the `raw_hourly_aqi` table to check for alerts on the different types of AQI. If we had more time, we would have liked to implement the EPA NowCast, which uses the partial lease squares linear regression. Alerts are sent as POST requests to a transactional email service, MailerSend. MailerSend is configured to send emails from our domain.

The real-time service uses the replica-set features of Postgres, and must be enabled on a table-by-table basis. For our application, we receive updates for the `reading`, `hourly_reading_stats`, and `raw_hourly_aqi` tables and display the updates on the map and real-time graphs.

Sending Alerts Sending alerts were implemented with a transactional email service, MailerSend. Once an account was created, API keys were stored inside the database and the PostgreSQL open-source schema, supabase-mailer for MailerSend [6] was applied. Emails are sent by posting a JSON to the MailerSend email endpoint `https://api.mailersend.com/v1/email` with a custom message defined in the database. HTTP requests are done through the HTTP PostgreSQL extension. Every time a new reading is added to the database, alert definitions, which are what is created when users create alerts, are checked for each layer and then send emails for the appropriate definitions, as shown in Figure 43.

6.11 The Web Application

The web application is a core part of our design, and is how the user will mainly interact with the Aether Node. The web application should allow the user to:

- View their devices on the map
- Add and modify their devices
- View the AQI and other pollutants on the map with real-time updates

[6] https://www.mailersend.com

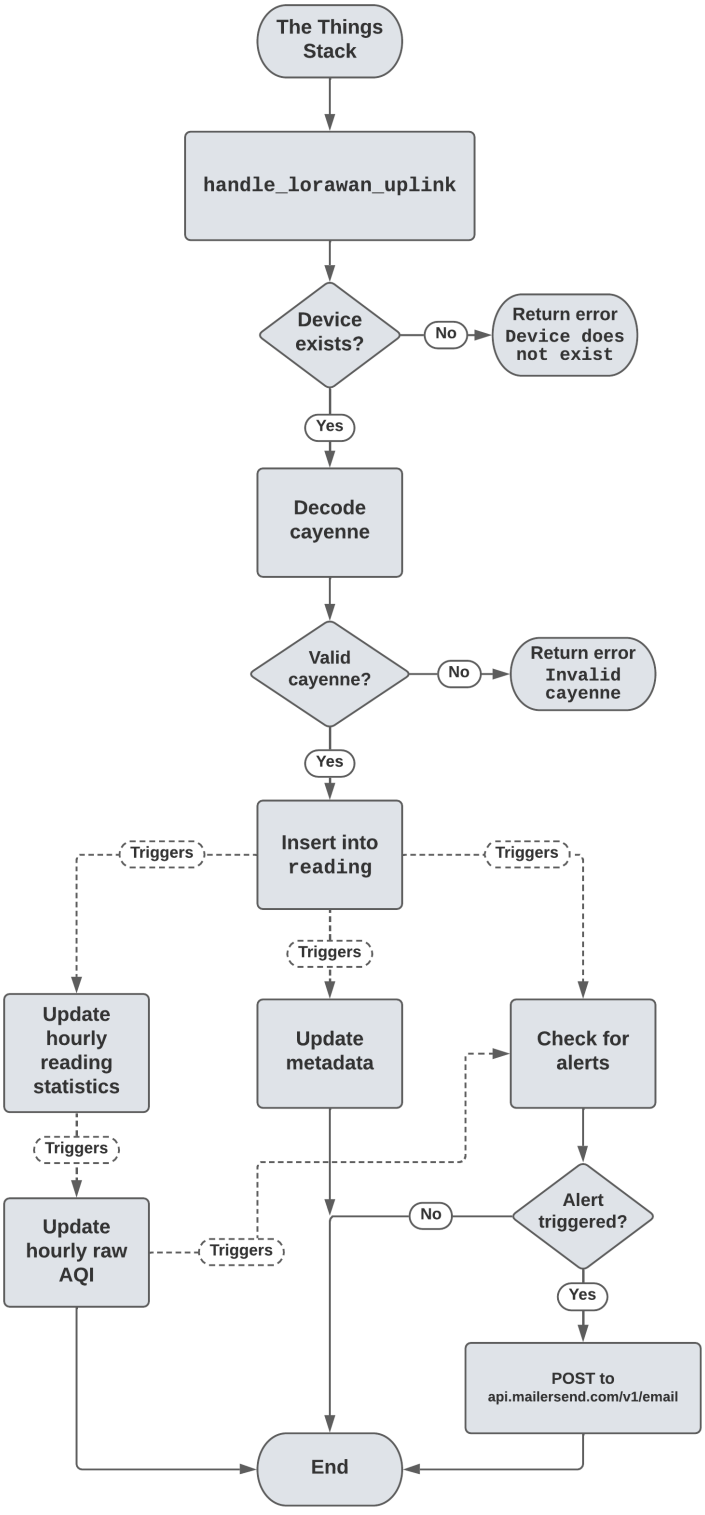


Figure 43: Supabase Design

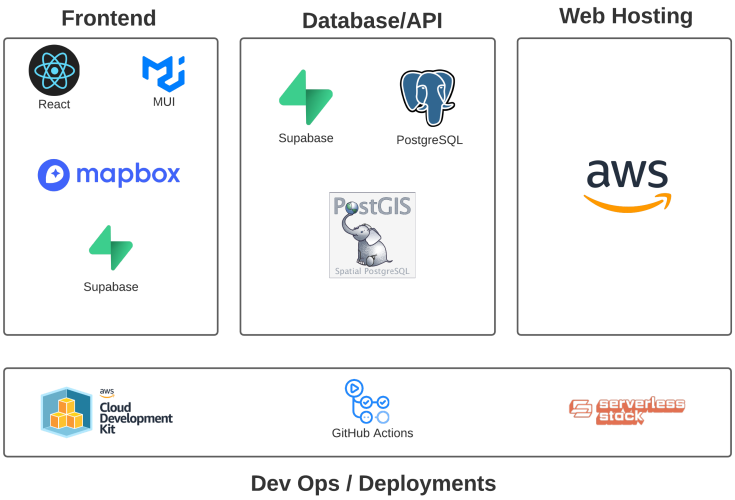


Figure 44: The Aether Web Application Stack

- View historical data on the map
- View real-time and historical data for each device
- Login and sign-up

6.11.1 Stack Design

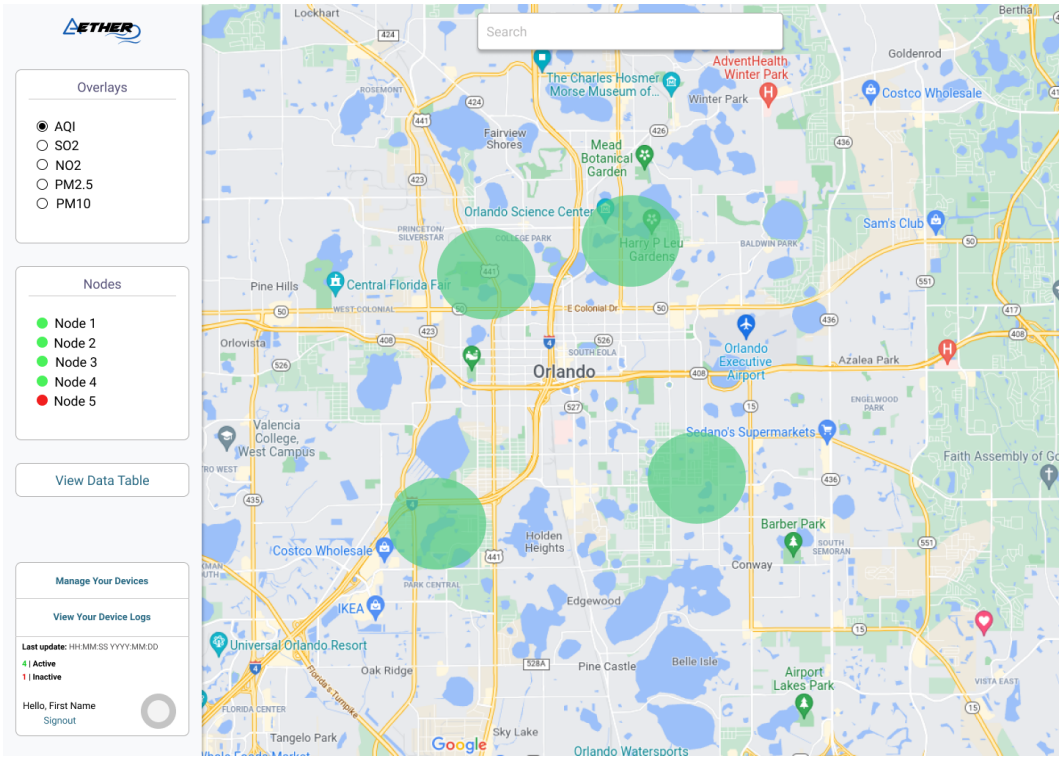
Figure 44 shows our overall stack design. Supabase is at the heart of the back-end providing user authentication, the database, the API, and finally, the real-time support.

We will be designed the web application using the JavaScript framework, ReactJS, along with Typescript. Users will be able to view one or many sensor data map overlays, and can toggle specific overlays, such as SO_2 or NO_2. An account won't be required to view sensor data, but an account will be required to setup an Aether Node. Users that are owners of sensor nodes will be able to manage their devices and get detailed status information. Our initial prototype of how we wanted the website to look is shown in figure 45.

The web application will use an assortment of JavaScript and React libraries to speed up development as our goal is to not start from scratch, but rather create a useful tool for users. It's common to use a wide-variety of libraries since they have usually been "battle-hardened" from their prolific use, such as React. The map service was initially provided by Google's Maps API [8], but the free-tier ended up being extremely limited. While the Google Maps API allows developers to draw shapes, change the styling of the map, adding custom markers, adding custom info windows,

[7]Designed with Figma, https://www.figma.com
[8]https://developers.google.com/maps/documentation/javascript/overview

Figure 45: Web Application Map View Prototype[7]



(a) Light mode

(b) Dark mode

doing searches, and much more. Google also provides a simple React Google Maps wrapper component [9]. However, we didn't want to spend a single dollar on the map service since this is a prototype, so we moved over to use Mapbox: a (free-er) open-source mapping library [10]. Mapbox is more extensible and customizable than Google Maps, and allowed us greater freedom in the styling the look of the map.

We also utilized a library called Deck.gl which provided OpenGL accelerated map layers. Layers are renderings overlaid on the map canvas to provide data visualizations. Deck.gl provides both 2D and 3D map visualizations, which allows us to create performant and visually pleasing data visualizations of the air quality [11].

As for the main UI design, the React Material UI (MUI) library [12] will be used. MUI provides pre-made UI React components that can be extensively customized. It also supports advanced theming to create custom color palettes, UI transitions, and the ability to dynamically change the color scheme of the UI (light or dark mode). MUI also provides a webapp to customize and generate a color palette that adheres to color theory.

React React is a component-based JavaScript framework for design web application user interfaces. React components encapsulate the state of a particulate UI component, such as a search box, list, or toolbar, to make state management and interactivity easier and more responsive. Components in React are used together in a hierarchical structure, and data is only passed down the hierarchical tree. Components can be instantiated multiple times allowing for more code re-usability and code sharability. There is a enormous collection of open source React components available online on GitHub and NPM (NodeJS Package Manager). Since most React apps are single page applications, meaning there is only one main server call to get the static assets (i.e. photos, HTML, CSS, JS), and the rest of the user interface control flow is handled by the client. This is called client routing. React Router is a library for easily creating dynamic routes for a React app. Different routes are displayed in the URL (the page isn't reloaded), and dictate which components are shown.

User Authorization User authorization will be carried out with Supabase, which handles user authorization, user creation, session management, and social logins, such as Facebook or Apple. Supabase ties into our overall backend and frontend design. Supabase provides us with access to our database, but also user authentication. This greatly simplifies our frontend design and lowers the amount of services used. However, it will be a stretch goal to have social login. The login and sign-up UI is handled by a Supabase library.

[9] https://www.npmjs.com/package/@googlemaps/react-wrapper
[10] https://www.mapbox.com
[11] https://deck.gl
[12] https://mui.com

6.11.2 User Interaction

The web application[13] is where anyone can view the air quality in their local area, given that there is an Aether Node nearby. The web application shows a map and sidebar on the homepage, as shown in Figure 46a. Each type of data to be displayed on the map, is known as a "layer". There are layers for the different types of AQI (O_3 and O_3 + PM), PM, O_3, temperature, relative humidity, and air pressure. For each layer, there is a unique legend that maps the colors of the data on the map to a numerical value. If a user does not have an account, they can view data, but they cannot see device or user specific information. Users with an account can see and modify their own devices. Devices can be viewed on the map by enabled the "Show" toggle in the sidebar. Users can see device metadata and view real-time and historical values by clicking on their device name in the sidebar.

In Figure 46b, a part of the device view is shown. Here users can see device metadata (not shown), real-time data feeds (not shown), historical data for AQI, gas pollutants, particulate matter pollutants (shown), and an event log (not shown). Users can edit basic fields, such as the location and name of the device.

Lastly, as shown in Figure 46c, all of the user alerts are tabulated and the new alert UI is shown. Users can create alerts based on a particular device and layer triggered off of a particular value. Optionally, users can add a description.

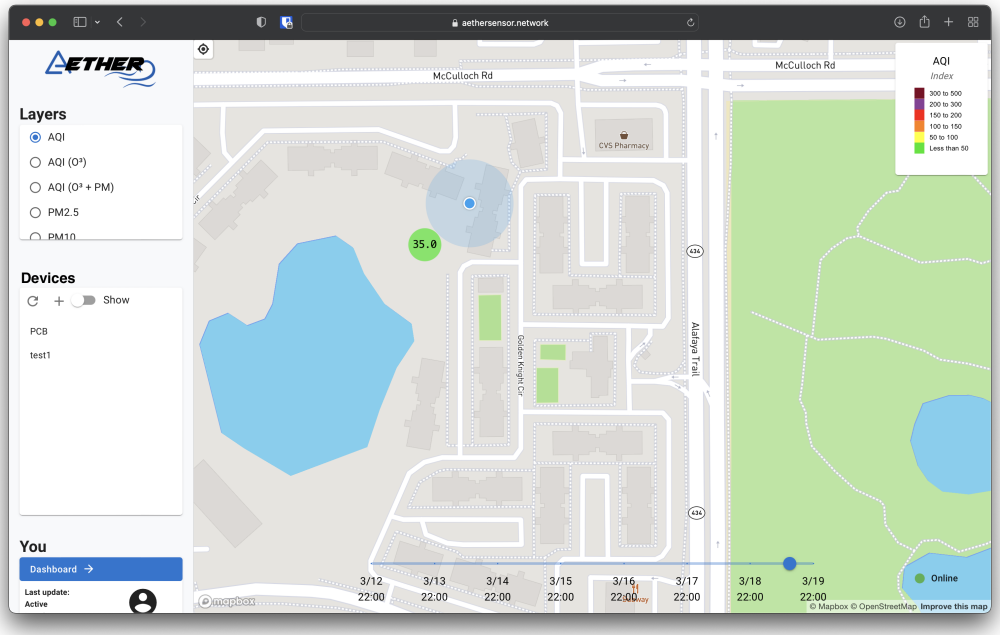
6.12 Design Summary

Aether's power design for the node consist of two main power inputs: a 6V solar panel and a USB-C input. The node will be able to operate using either of these inputs and when one is connected it will be simultaneously powering the system while charging the battery. The USB-C protection circuitry is designed to prevent against ESD and EMI damage with bidirectional transient-voltage suppression (TVS) diodes. Additionally, since the LoRa microcontroller can not communicate directly through USB protocol, a USB-to-UART bridge was implemented into the design to allow for proper communication between USB and the LoRa module. The next step in the power design for the Aether node was having a charge controller IC capable of taking two varying power inputs and efficiently charge the battery. A major challenge to this design was finding a charge controller IC that would not demand excessive current from the solar panel and continuously charge and discharge the battery, decreasing the overall system efficiency. The MCP73871 is capable of using as much current as the solar panel can output without reaching its drop out voltage. It is responsible for passing power from the battery or solar-panel using its load-sharing functionality.

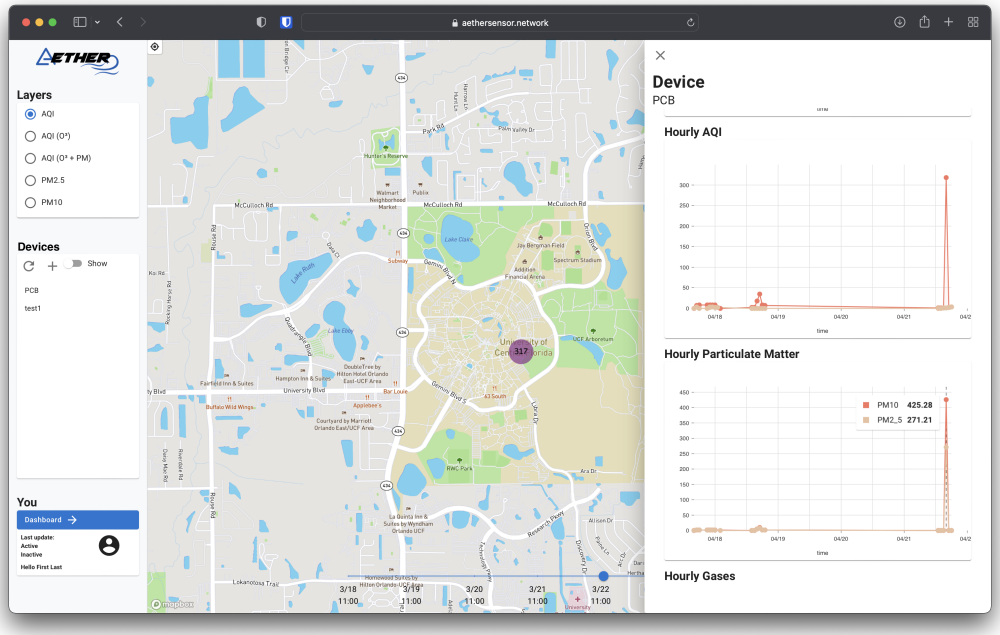
In the Aether power design, when the solar panel cannot efficiently meet system load demands, the MCP73871 charge controller will take the remaining necessary current from the Li-Io battery to feed the load. If the USB is used as the power

[13]The Aether website can be be accessed at the following address: `https://aethersensor.network`

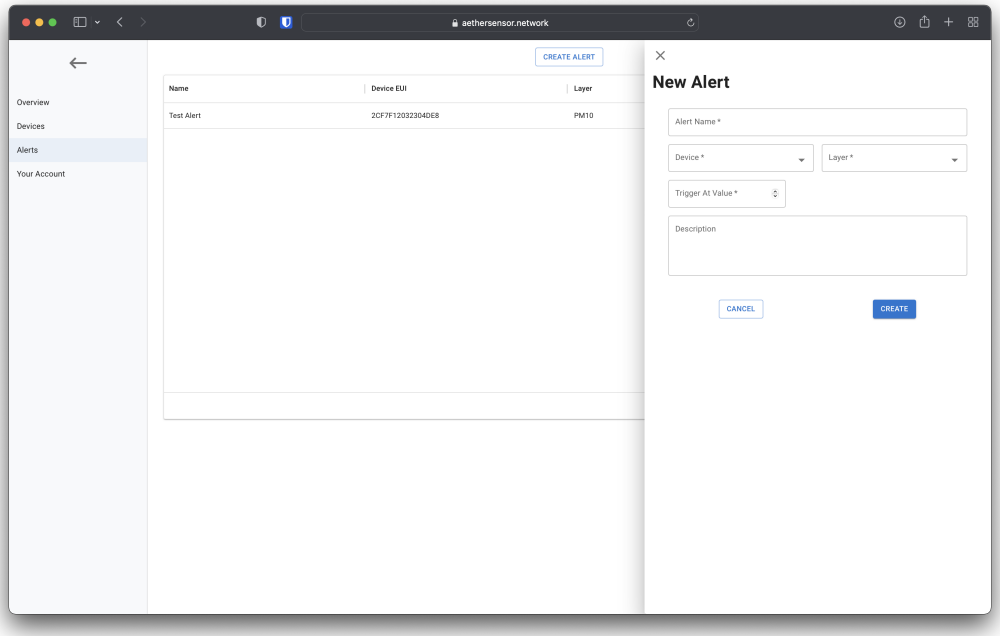
Figure 46: The final web application



(a) Map view



(b) Device View



(c) Alerts view

input, the charge controller will efficiently charge the battery and the USB input will be used to power the system. Since the LoRa module and most of the sensors in the system operate at a 3.3V range, a linear voltage drop out (LDO) regulator was needed at the MCP73871 output to design a 3.3V power line for the system. The particulate matter sensor in the node requires a 5V input so a boost converter was used, taking the battery as an input to output a consistent 5V power rail. The boost converted has a shutdown pin which is used by the microcontroller to completely shut off power to the SPS30 since it has a relatively high sleep current. Additionally, a therm-resistor is connected to a pin on the MCP73871 to monitor battery temperature and ensure the battery is not charge under unsafe conditions.

For the Aether MCU and sensor subsystem, the 3 main sensors in the Aether node are connected to the LoRa-E5 MCU and communicate through I2C protocol. The Bosch BME688 sensor and Renesas' ZMOD4510 sensor operate using the 3.3V power rail while the SPS30 Particulate Matter sensor uses the 5V power input from the boost converter. The sensors are configured to I2C communication and the respective SCD and SCL pins are connected to the LoRa module. An antenna which will be used for the LoRa receiver is connected to the MCU. Several GPIO pins on the LoRa module are used for communicating with the system, including USB communication, battery status, and remote shutdown of the SPS30 sensor for low-power mode operation. One GPIO previously mentioned is used for turning on and off the power to the SPS30. Additionally, an MCU GPIO pin is configured as an EXTI (external interrupt) source to detect USB input and wake up the microcontroller.

7 Prototyping

In this section, we describe what we did to implement the design discussed in the previous section. Prototyping our design involved ordering and assembling multiple printed circuit boards (PCB). Therefore, we will discuss what is involved in going from a circuit schematic to a functioning PCB. We will also discuss how the PCB will be fabricated. Additionally, we will discuss the development board we used to prototype the firmware that eventually ran on our finished design. It was necessary to use a development board so that we were able to work on designing both the hardware and software in parallel. This section will also cover what went into putting together the Aether Node enclosure. We sourced parts from mostly Mouser and Digikey, however some were bought off of Amazon for their 1-2 day shipping.

7.1 Printed Circuit Board

A PCB is an essential part of this design. Therefore, it will be critical to understand the details of how PCB functions so that we can best translate our circuit design on to one. There are many different parameters that go into PCB that we discuss in the following section.

7.1.1 What is a PCB?

Before we get into the details of the PCB design first lets go over what exactly is a PCB or a printed circuit board. When computers were first constructed they took up massive rooms and could provide much less processing power than they could today. However after decades on innovation and inventions we can now fit more processing power than the original space shuttles comfortably in our pockets. One such invention is the PCB. In appearance a PCB resembles a flat green board but under a microscope there is so much more detail. Within a PCB there are several layers that make up its form and function. The first one is the mounting components. These are the top most and bottom most layers that provide placement for various components such as microchips, resistors, capacitors or anything else that may require placement on the PCB. The next two layers well discuss are the power plane and ground plane. As their name suggests these provide power and ground respectfully to any components that require as such. Finally we have the signal trace layers. These layers are composed of copper, insulating fiberglass, and an epoxy resin that prevents any flow of electricity. PCB usually are composed of anywhere in between 2-50 signal trace layers as these layers connect components of the PCB to one another.

Now that we know that PCBs are comprised of layers that provide power, ground, and communication between devices on the motherboard an appropriate question to ask is how layers connect to one another between the insulating material. This is due to VIA's or Vertical Interconnect Access. VIAs are drilled and metal plated holes that are used to connect two or more layers on the PCB. There are three types of VIAs those being a through hole, blind VIA, and buried VIA. A through hole goes

completely through the PCB from the top layer to the bottom layer. A blind VIA connects the top or bottom layer to a layer part way through the PCB. Finally buried VIAs connect two or more central layers.

7.1.2 PCB Function

PCBs are integral to almost every modern day electrical system. A PCB allows seamless and streamlined communication between system components, effortless powering of circuits, and allows for a much more compact and professional looking design for any electrical system. All in all, while a PCB is not necessary to make circuit designs work, without them any system that could otherwise be implemented on a PCB would be many more times the size and complex to construct.

7.1.3 PCB Design

The PCB was designed on KiCad. When selecting the parameters of our PCB some information had to be decided upon. Decisions made were on RF routing, number of layers, trace thickness, and SMD passive sizes and footprints. The first decision made is on trace thickness. Most of the signal traces on the board were 0.25mm thick, sufficient enough to handle the system's current loads. For the power signals and RF signals, thicker traces of 0.5mm were used since these signals can dissipate more heat and require larger track widths to preserve signal integrity. Due to the number of components with a high number of pins and overall number of items that will need to be placed on the board its been decided that having a pin density of around 0.5 would be best. As a result the system will be utilizing a 4 signal layer PCB. The top and bottom layers will be used to route signals, the second layer will be used as a 3.3V plane and to route other power signals, and the third layer will be a ground plane. When it comes to SMD passive sizes the SMDs used in the design will be mostly size 0805, with a few 0402 components. This will allow the design to be compact while also making construction more streamlined as any smaller would create more difficulties to solder by hand.

A big decision considered in this design was RF routing. The beginners essential to RF design is minimizing cross talk. This is when more than one RF signal can tamper with one another due to E and H fields. The way this will be minimized is by utilizing the 4 trace layer design. While the two inner most trace layers will be dedicated to low speed digital signals the top layer will be dedicated to our RF trace. This will provide an adequate amount of space between layers that will eliminate cross talk allowing the board to effectively separate the signals. In addition to this, multiple ground vias surrounded RF signal to dissipate heat and preserve signal integrity. Another consideration was trace length. The design makes sure to minimize the length of the RF trace as much as possible by keeping it as close to the RF pin on the MCU as the physical design constraints allowed. Figure 47 shows the final design of the PCB using KiCad. The final design of the PCB is shown in Figure 48. As seen from Figure 48, the final revision was modeled almost exactly as shown in the

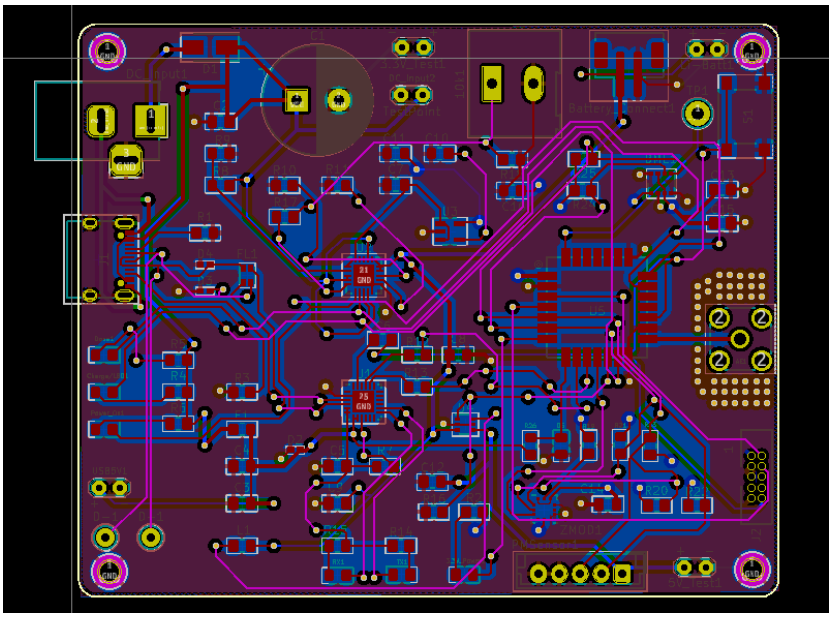


Figure 47: PCB design

3D rendering.

7.1.4 PCB Model

Once the design was complete for the PCB, 3D modeled components were imported into KiCad and KiCad's 3D rendering feature was used to export a 3D model of the assembled PCB. This model not only gave the team an idea as to how the PCB will look but also allowed for an enclosure to be designed around it. The final dimensions of the PCB came out to be 63mm x 80mm. This met the design requirements allowing the overall system to be compact and easily portable. An image of the PCB 3D model can be seen in Figure 49

7.2 Bill of Materials

The bill of materials for the Aether Node can be found in Figure 50 and Figure ??. These BOMs are generated from our schematics in KiCad and list all of the major components, such as ICs, as well as all minor components, such as specific capacitors and resistors. The passive components are simply labeled as `R_US` or `C_Small`. The generated BOM also lists the estimated costs of each component, with current estimated total cost of a single Aether node being $178.03.

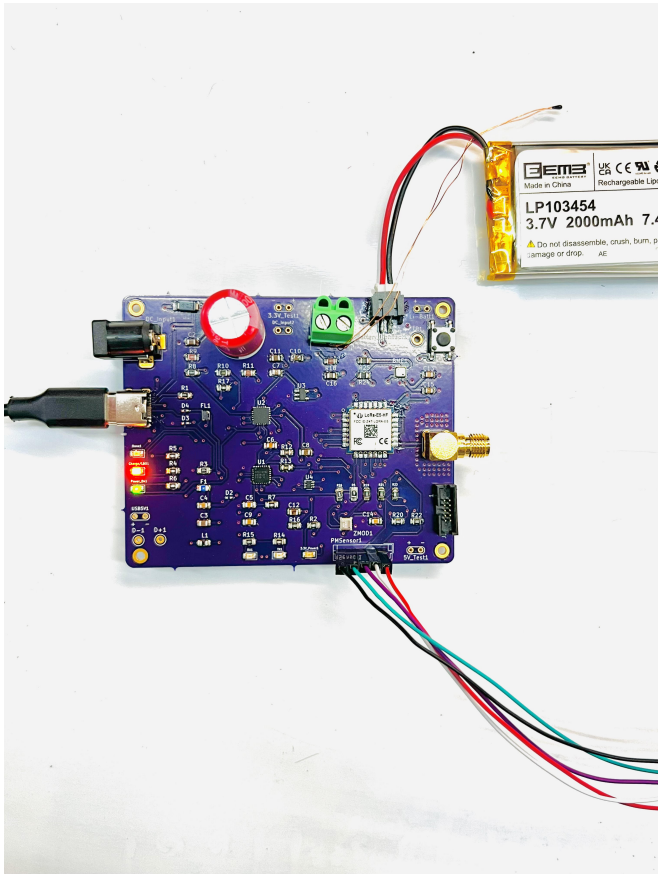


Figure 48: Final PCB Design

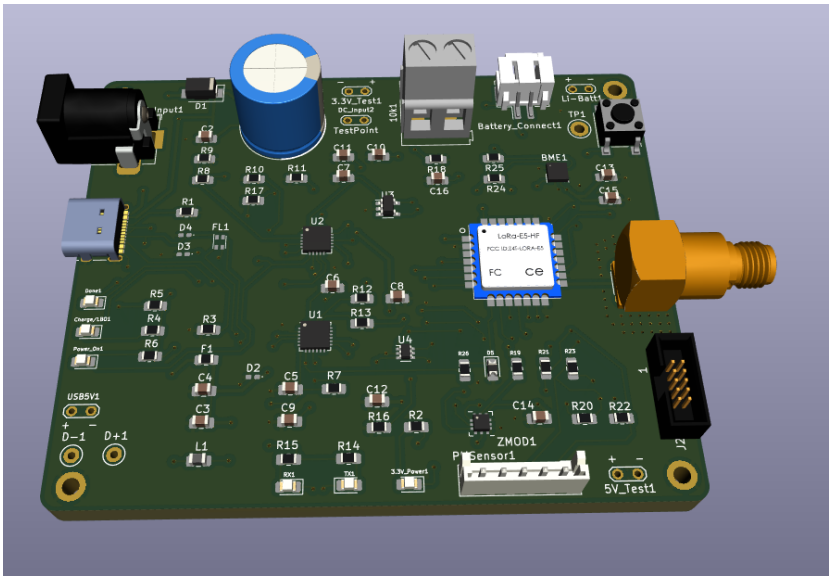


Figure 49: PCB 3D Model

Aether AQI Sensor Network

Assembly Name :	Aether Node
Assembly Revision :	12/4/2021
Part Count :	73
Total Cost :	$178.03

Part Name	Part Value	Library Part	Qty	Unit Cost	Cost
10k1	10K Thermistor	Device:Thermistor	1	$ 5.00	$ 5.00
AE1	Antenna	Device:Antenna	1	$ 7.48	$ 7.48
BATT_(+)1, BATT_(-)1,	TestPoint	Connector:TestPoint	4		$ -
Battery_Connect1	JST_2_PIN_Connector	Connector:Conn_01x02_Female	1	$ 0.17	$ 0.17
BME688	Bosch	sensors:Bosch	1	$ 20.50	$ 20.50
C1	4700uF/10V	Device:CP1	1	$ 0.15	$ 0.15
C2	10uF/10V	Device:CP1	1	$ 0.15	$ 0.15
C3, C12	10uF	Device:C_Small	2	$ 0.15	$ 0.30
C4, C5, C6, C13, C14,	100nF	Device:C_Small	6	$ 0.15	$ 0.90
C7	2.2uF	Device:C_Small	1	$ 0.15	$ 0.15
C8, C11	10uF	Device:CP1	2	$ 0.15	$ 0.30
C9	1uF	Device:C_Small	1	$ 0.15	$ 0.15
C10	10nF	Device:C_Small	1	$ 0.15	$ 0.15
C16	4.7uF	Device:C_Small	1	$ 0.15	$ 0.15
Charge/LBO1	Orange	Device:LED	1	$ 0.15	$ 0.15
D1, D3, D4	ESD Diode	Diode:ESD9B3.3ST5G	3	$ 0.15	$ 0.45
D2	B120-E3	Diode:B120-E3	1	$ 0.15	$ 0.15
D5	3.0V	Device:D_Zener	1	$ 0.15	$ 0.15
DC_Input1	Solar Panel Jack Input	Connector:Barrel_Jack_Switch	1	$ 0.76	$ 0.76
Done1, LED1	Green	Device:LED	2	$ 0.15	$ 0.30
F1	1A	Device:Polyfuse_Small	1	$ 0.15	$ 0.15
FL1	Choke	Device:Filter_EMI_CommonMode	1	$ 0.70	$ 0.70

Figure 50: The bill of materials for a single Aether node. (Part 1)

Part Name	Part Value	Library Part	Revision	Qty	Unit Cost	Supplier	Unit Cost2	Cost
L1	10uH	Device:L		1	$ 0.15			$ 0.15
LED2	Blue	Device:LED		1	$ 0.15			$ 0.15
P1	USB_C_Plug	Connector:USB_C_Plug		1	$ 0.76			$ 0.76
PMSensor1	SPS30	sensors:SPS30		1	$ 52.83			$ 52.83
Power_On1	Red	Device:LED		1	$ 0.15			$ 0.15
R1, R2	5.1k	Device:R_US		2	$ 0.15			$ 0.30
R3, R4, R5, R6, R13, R14, R16	1k	Device:R_US		7	$ 0.15			$ 1.05
R7	270k	Device:R_US		1	$ 0.15			$ 0.15
R8, R10, R17	100k	Device:R_US		3	$ 0.15			$ 0.45
R9	2k	Device:R_US		1	$ 0.15			$ 0.15
R11, R12	20	Device:R_US		2	$ 0.15			$ 0.30
R15, R18, R19, R20, R23, R24	10k	Device:R_US		6	$ 0.15			$ 0.90
R21, R22	4.7k	Device:R_US		2	$ 0.15			$ 0.30
R25	36k	Device:R		1	$ 0.15			$ 0.15
U1	CP2102N-Axx-xQFN24	Interface_USB:CP2102N-Axx-xQFN24		1	$ 1.26			$ 1.26
U2	MCP73871_U	Battery_Management:MCP73871		1	$ 1.89			$ 1.89
U3	LP2985-3.3	Regulator_Linear:LP2985-3.3		1	$ 0.80			$ 0.80
U4	LTC3525-5	Regulator_Switching:LTC3525-5		1	$ 0.53			$ 0.53
U5	LoRa-E5	LoRa-E5:LoRa-E5		1	$ 9.90			$ 9.90
U6	LP-103454	Li-Io Battery		1	$ 25.00			$ 25.00
U7	P105	6V Solar Panel		1	$ 35.00			$ 35.00
ZMOD4510	Renesas	sensors:Renesas		1	$ 7.50			$ 7.50
	Total			73				$ 178.03

Figure 51: The bill of materials for a single Aether node. (Part 2)

7.3 PCB Fabrication Services

Before designing the PCB, we made sure to assess the available PCB fabrication (fab) services so that we could configure the Design Rules Check (DRC) in KiCad to match the fab's capabilities. We sought after services than can provide a subjectively good quality board at less than $100 for the entire run. We did not require any special capabilities regarding vias, such as blind or buried vias. We looked at fabs in both China and The United States, but we knew ahead of time that the current market conditions might be adverse towards Chinese fabs even accounting for their cheaper boards. The fab services we researched were:

- JLCPCB *(Chinese)*
- PCBWay *(Chinese)*
- OshPark *(United States)*

The main considerations were cost with quality a secondary concern. However, the Chinese PCBs are oriented towards quick prototyping, so they are of less quality than the PCB fabs we analyzed in The United States. Table **??** compares the fabrication services we considered. We chose OshPark for their cheapest cost (print + shipping), good quality, and fast shipping and turnaround times at the time of ordering (January 2022). The quality, overall cost, and time-to-delivery outweigh only receiving 3 boards.

Table 23: Considered Fabrication Services

Fab	Quantity	Printing Cost ($)	Shipping Cost ($)	Turnaround Time (days)	Shipping Time (days)
JLCPCB	5	4	80	4	21 - 28
PCBWay	5	6	76	3	21 - 28
OshPark	3	80	0	14	4 - 7

7.4 Facilities Used for Prototyping and Development

For our project, we plan on using a variety of tools and lab spaces provided to us through the university. This will be the case primairly for developing and prototyping our hardware. We will have access to the UCF ECE Senior Design Lab. This lab will provide us access to a variety of critical tools for building and testing electronic circuits and hardware. This lab equipment such as oscilloscopes, function generators, and power supplies. In addition, there are important tools related to PCB assembly, such as SMD rework stations, solder and desoldering stations, and a digital microscope inspection station.

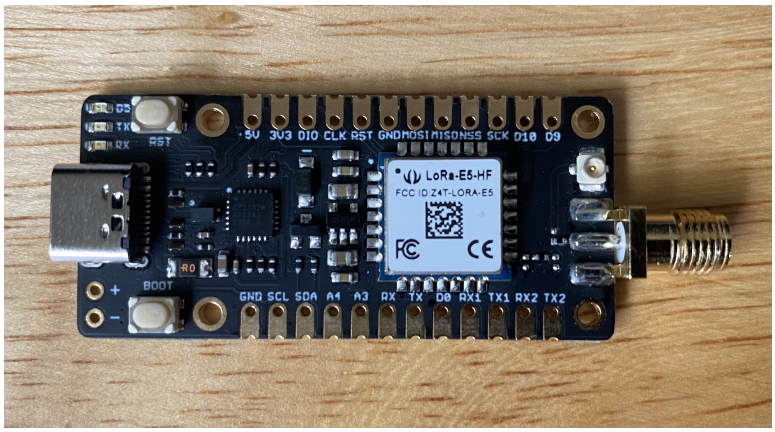


Figure 52: The LoRa-E5 mini development board that we used to prototype our software

7.5 Development Board and Firmware

In order to prototype our sensor node software, LoRaWAN gateway, and server, we are using a development board that uses the same microcontroller and LoRa module that we are going to use for our final design. That development board is the Seeed LoRa-E5 mini. The development board itself is fairly barebones, containing only the LoRa module itself (the STM32WLE5JC), some I/O pins, a USB Type-C input for power, and some antenna connectors. The board does not contain a USB-based programmer, only SWD pins. Instead we had to purchase an external STLink programmer to be able to connect the development board to our computer for programming. An image of the development board can found in Figure 52.

The initial prototype of the firmware was done on the development board in tandem with a dedicated sensor boards for both the BME688 and ZMOD4510 as well as the SPS30 and sending them over LoRaWAN to the The Things Stack. The firmware was developed using the STM32CubeIDE and FreeRTOS initially, but development was quickly shifted over to using Zephyr. Zephyr allowed to write drivers separately for each sensor and then abstract away the implementation details within the application code. This allowed us to write the drivers for different sensors simultaneously more easily without breaking existing code. Zephyr also provided us with out-of-the-box support for information and debugging logging over serial. We were also able to easily implemented a shell interface, as that was also included with Zephyr.

7.6 Enclosure

Building the enclosure prototype according to our design was fairly straightforward. Since we were 3D printing it, all we had to do was export our CAD model to a .stl file and send it to the 3D printer. Our enclosure design had two major components that had to be printed separately, the top and the bottom. We ended up using a black PLA+ material for printing the enclosure. We also had to make a few

Figure 53: The final print of our enclosure.

changes to our design since we realized there were some issues with hole clearance once the enclosure was printed. The bottom piece of the enclosure took around three and half hours to print and the top took around one and a half hours to print. The final print of the enclosure with the final PCB and components installed can be seen in Figure 53.

7.7 Backend and Webapp Prototype

As an initial prototype and proof of concept, the backend and webapp will implement very simple functionalities. The goal of the proof of concept (PoC) is to test simple integration with the development board, the AWS LNS, the database, and the React webapp. The backend services, initially, will only be setup to log decoded LoRaWAN packets in the AWS DynamoDB, and display the logs in a table using the MUI React UI framework. From there, work can be done on integrating the Google Maps API, and generating custom data overlays using tools available in the API. The differences between the proof-of-concept and the minimally-viable-concept phase are tabulated in Table 24.

7.7.1 The Proof of Concept

As aforementioned, the proof of concept will only consist of taking data readings on from the development board of a light sensor, transmitting it over LoRaWAN to the AWS LNS, decoding the binary packets and logging them into the AWS DynamoDB,

and then displaying the logs in the React webapp using the MUI framework. There will be no user authorization, or even a concept of a user at this phase. End devices will be manually configured on the AWS LNS through the AWS IoT Core service. As tabulated in Table 27, the PoC should be completed before the Spring semester, but ideally before the start of the new year (2022). The proof of concept will also include email and SMS notifications when a light threshold is met.

7.7.2 The Minimally Viable Product

The Minimally-Viable-Product (MVP) prototype phase implements the absolute basic requirements of the project, such as the interactive map. Users will be able to register their devices on the webapp, but not many other features will be available for controlling and configuring devices remotely. Also, only the interactive map with overlays will be created, and any static or real-time time-series graphs/overlays won't be covered in this phase. The user will only be able to see the status of their sensor node and see its exact location (whereas for non-owners, the location won't be pin-pointed). The interactive map will allow the user (doesn't require login) to toggle sensor readings on the overlay, and hovering over areas of the map will reveal an info window for more details about the sensor data in that area.

Feature	Proof of Concept	Minimally Viable Product
Logging	✓	✓
Email Notifications	✓	✓
AQI Sensor Data	×	✓
Data Map Overlay	×	✓
User Registration/Authorization	×	✓
Time Series Graphs	×	✓

Table 24: Differences between the two prototype phases

8 Testing

With so many moving parts in this project, it was imperative that a proper testing protocol and plan was created. The hardware and firmware are arguably the most important parts of the parts of the project since the remainder of the project is set up around the sensors and the services they're able to provide. Without proper testing, unforeseen problems would have been hard to isolate and resolve. The hardware needed to work correctly for the firmware to operate effectively and accurately. Without accurate sensing, the utility of the project vanishes.

The server and web application were tested to ensure a seamless experience for the user. The firmware will need to have some amount of unit testing for key, critical systems and functions. Major services will need to have some form of unit and integration testing. Our test plans will detail the tools and methodologies used in testing major parts of the project.

8.1 Microcontroller Testing

Our testing plan for the micro controller will mostly revolve around attempting to perform various simple tasks and checking if they work. First, we will test if we are able to power the micro controller on. Next, we will begin testing the functionality of the part. One of these tasks will be to see if we can program the micro controller over the programming pins. We will write a simple program that turns on and blinks them. Another task will be to see if we can send AT commands to the on board LoRa module.

8.2 Sensor Testing

When it came to testing the sensors the team was thorough and diligent. While every portion of the system was important this is what every other sub system is designed around. Should errors persist within any sensor the entire network could be compromised with inaccurate and inconsistent information. The main point of contention when it came to testing our sensors was simply their accuracy. Luckily for the team the EPA already has air quality sensors in our location that track similar measurements and as a result we have something to compare our sensors to. One of our requirements put us in a range where our sensors must match the readings of the EPA sensors within a margin of 25%. We demonstrated this and proved that our sensors were in fact accurate according to EPA measurements.

8.3 Power System Testing

The essential elements of testing our power sub system can be described as basic and comprehensive. If any single element were to introduce a factor to limit or boost power unexpectedly certain components may not function or become damaged beyond repair. As a result the upmost care has been taken into account when testing the

power subsystems of the device. The most effective method of testing in our opinion is to monitor the voltage and current starting at the power sources going all the way to the load. We showed that the current and voltage were both well within the ranges we set and are sufficient to provide power to the entire unit.

8.3.1 Source Testing

The first step we will undergo would be to test the solar panel. This will be done by using the panel, a multi meter, and a resistance load on the battery terminal for monitoring. We took the solar panel outside to demonstrate the direction and magnitude of the current flowing through the system. Through this test we showed that while the solar panel was plugged in during sunny conditions not only was the system being powered the battery was also receiving roughly 180mA of current. This demonstrates not only does the solar panel work, it also provided more than enough power to keep the system running indefinitely.

The next step to testing our sources would be to test the USB input. This is a far bit simpler as the USB input would have a standard input we could compare it to and any difference would indicate an inconsistent or unreliable element. Additionally we don't have to worry about a fluctuation in power as the USB input would stay steady at all times and any fluctuation would also indicate an undesirable element. This is much simpler as out testing came down to solely powering the unit with the usb and testing all of its functions. We noticed that the entire system worked perfectly upon the connection with the USB confirming its function.

8.3.2 IC Testing

When we get to the IC a new approach must be made. First, when monitoring the input from the sources care must be made to monitor the readings directly at the input of the IC as well as the output reading directly from the sources. This will tell us if there is a factor manipulating the power before it reaches its destination. The next step would be to observe the ICs output voltage and current. Now due to the nature of the system the input voltage of the IC may vary, but the same cannot be said about the output voltage. The next step would be to observe the output voltage of the IC as the solar panels input voltage changes. Should the output voltage change in sizeable way then the IC is not accomplishing one of its core purposes and should be examined and repaired or replaced. Upon testing we found that the IC was working as expected.

For ease of use, testing points have been added at select points that can be used, these points can be found in the schematic and are the points we will be using to test this subsystem. The next test that will be implemented will be for the other functions of the IC. For example the IC is supposed to begin drawing power from the battery when no other source is present so when the battery is charged the solar panel will be removed from all light sources and the USB will be disconnected. If the battery begins powering the system then we will have verified this function of the IC. From there

the battery will be constantly depleted to see if the IC will identify a low battery. If the output voltage of the battery is low then we can confirm independently that it has a low charge and the IC should indicate as such. If it does then this function is confirmed.

8.3.3 Battery Testing

In terms of testing the battery we the procedure is simple and straight forward. First to test that the battery is functioning properly then with no sources attached the battery is connected and if the system starts operating as expected with the correct battery voltage then we know the system is functioning correctly. Then with a source attached we observed that the battery voltage increases proportional to what the charge controller would output.

8.3.4 Power Rail Testing

Power rail testing is more straight forward. At this point, we have covered the sources, the IC, the battery, and all the connections in between. Should the rest of the system be predictable, stable, and expected now comes the load itself. The power rails comfortably change the voltage from the IC and battery to the voltage needed for every component.

The most straightforward way to test the railing would be to use a multi meter and observe the voltage at the output of the IC and battery to verify that it is indeed correct. Next, you would observe that the input voltage of each power rail is the same as the output voltage you just measured. If that is not the case, then an element may be present in the connection leading to the difference and the up most care should be taken to discerning the reason behind the change. Finally, for each power rail you would need to ensure that the output voltage of the rail is as predicted. For ease of use, testing points have been added at select points in our design. These points can be found in the schematic and are the points we will be using to test this subsystem.

8.3.5 Power Usage Testing

One of the required testing procedures as the project came on was finding a definitive calculation for the consumption of power in our system. As a result we had to test our initial power consumption calculations. To do this a small 6 ohm resister was soldered in between the positive and negative terminals to the battery create a system we could calculate. From there the battery became the sole source powering the system where we could get accurate reading on the power being drawn out of it. Next we set the node up to run for several hours. Connected to the resistor is an oscilloscope recording the voltage and current being drawn from the battery. With this we were able to determine that our power consumption estimations were far off. Originally we calculated that our battery could operate the node independently for months but 2 factors caused that calculation to be much higher than the real figure of just over 4.5 days. The first cause of the overestimation was a part change part way

through construction. We found part way through construction of the prototype we selected a battery with too many cells to be charged by the charge controller. As a result we have to get a battery with less cells and a lower power capacity. The other large reason for the over estimation was that we severely underestimated the time it would take for the system to take accurate readings. Originally we thought each sensor would only be active for a few seconds but in reality the whole system would take much longer than that. This causes the node to be operating on full power much longer which depletes the battery life much quicker.

8.4 The Firmware Test Plan

The firmware testing will utilize our existing CMake build system the Unity unit testing library. Tests will be split into two types: target-agnostic and target-specific. Target agnostic tests are tests that can run on any architecture that can compile C. They are tests that don't reach down into the microcontrollers HAL, low-level drivers, or hardware specific peripherals or registers. Target-specific tests are tests that use hardware specific features, registers, or peripherals. These tests might check the status of the LoRa RF module, correct operation of I^2C, or the correct operation of the sensors. Target-agnostic tests will be configured to run in a GitHub continuous integration pipeline to help automate and ease development of new features. Target-specific tests will have to be run manually. Since we are using FreeRTOS, we will need to create tests that can check for deadlocks, thread starvation, and other issues commonly had in concurrent programming. The firmware will also need to be error-tolerant, and correctly recover from as many errors as possible. Issues will be tracked using ClickUp or GitHub, and assigned to someone to solve.

Unit tests will only be written for key subsystems. Having too many unit tests creates a burden for the programmer, but just enough keeps the code base mostly free of certain types of errors. The general flow for developing and testing the firmware will be the following: first write the code and consider the downward effects of the code having issues and determine if it's a critical system that needs to be guarded by unit tests. After writing the unit tests, on each iteration of the firmware, the code will be automatically tested when pushed to GitHub using the target-agnostic tests using GitHub Actions. Before creating the next major version of the firmware, the target-specific tests will be run to ensure its functionality. If all tests are passing, the code will be merged into the main branch, and deployed either manually or automatically through LoRa OTA updates. Then process then repeats until the firmware is complete or near-complete. Some key systems that would be good to write unit tests for are:

- Drivers for each of the sensors
- The sub-GHz module and LoRaWAN drivers, including configuration
- Power management
- The command line interface (over serial - UART)

- Power management
- Each command the user is able to run over the CLI or remotely
- (Stretch goal) Over the air updates

8.5 The Backend Testing Plan

Since both the backend AWS services and web application will be written in JavaScript/TypeScript (NodeJS runtime for the server-side things), the testing software tools can be a unified set, simplifying development and testing. Like with the firmware testing, the backend and web application will have a simple continuous integration pipeline configured on GitHub to automatically test the code before merging changes into the main, or (hopefully) stable, branch on the repository. We will be utilizing the Jest unit testing framework since it has tight integration with Typescript, Babel, and React. It is also one of the most popular framework, and is widely used by big companies, such as Spotify, Facebook, Twitter, and Instagram. Jest was also chosen because it is easy to use with minimal or no configuration, it's well documented, works with asynchronous code, and provides code coverage statistics. It'll be imperative that our unit tests hit the following systems:

- Sending and receiving commands and data to and from air quality nodes
- Managing information regarding a specific user, their information, and configuration in the database and other systems that might touch the data
- Registering new devices with the AWS LNS
- API endpoints regarding geographical and AQI map information
- API endpoints regarding user registration, configuration, and updating and retrieving data

Integration tests would be nice, but due to time constraints, they might be very limited. The same is true for unit tests, but a balance will be found if time would be saved by having the test. Integration tests that might be nice to have would be:

- Registering new devices with the Helium Network or The Things Network
- Using map data from Google Maps or another map service provider

8.5.1 Jest

Each test in Jest is created with the `test(name, fn, timeout?)`. Inside the `test()` function, the test is written. To check and verify values, the `expect(value)` is used. The `expect()` function returns an `Expectation` object, with provides an interface to *match* the `value` with various conditionals, called "matchers". For exam-

ple, you could have a function that computes a result, and the function is operating correctly if the resultant value is between 0 and 100. Like, Unity, there is a huge set of matchers available to use. The `.toEqual(value)` matcher would be used, like in listing 5. Jest provides the following report after running all of the tests, as shown in listing 6

8.5.2 Testing the Serverless Application

Since we are opting for a serverless backend (using AWS Lambda), the Amazon Serverless Application Model CLI will be utilized for testing (and for deployment). Through YAML configuration files, a serverless application can be deployed with the right configuration automatically. It also provides a local testing environment for Lambda. The AWS SAM CLI also provides debugging capabilities that allow the developer to step through the code and view information about the current state of the program. For AWS Lambda, the main functionality will be decoupled as much as possible from the Lambda handler function to maximize the amount of testing that can be done locally and through the GitHub continuous integration pipeline without requiring access to AWS services.

9 User Manual

In this section, we provide a manual to assist individuals with using the Aether Sensor Network. This includes information on how to use the Aether website, update the Aether Node firmware, and to use the different ports on the Aether Node itself.

9.1 Using the Website

The Aether website has many features for viewing data coming from registered Aether Nodes. The data from a registered Aether Node will appear on the map as a colored circle with a number inside. The value of that number will change depending on the current layer that is being viewed. The currently selected layer can be changed using the Layers panel on the left-hand side of the screen. The available layers include AQI, AQI (O_3), AQI (O_3 + PM), PM2.5, PM10, O_3, temperature, and relative humidity. On the top right, you will see a legend that indicates what each circle color means depending on the currently selected layer.

To view more information about the data from a particular node, simply click on the name of the node under the Devices panel. A panel will appear on the right-hand side of the screen. This panel includes information about the particular Aether Node, including the Device EUI, when the device was created, when the last uplink was, and the device location. Under the real-time section, you will find a graph with real-time data coming from Aether Node. This section includes data on particulate matter, O_3, humidity, and temperature. The Hourly AQI section shows a graph with the AQI over time reported by this Aether Node. The Hourly Particulate Matter section shows a graph with the PM over time reported by this Aether Node. The Hourly Gases section shows a graph with the O_3 over time reported by this Aether Node. Finally, at the bottom of this panel you will find the raw data being sent by this Aether Node.

When visiting the Aether website for the first time, you will need to either sign in or create an account.

Create a New Account

1. Click the "Sign Up" button on the left-hand side of the screen
2. Fill in an email
3. Create a password
4. Click on "Sign up"

Register a New Aether Node

1. Click on the + under the Devices panel

2. Create a name for the device
3. Fill in the device's Device EUI
4. Enter the latitude and longitude of the device
5. Click create

Create an Alert

1. Click on the "Dashboard" button on the bottom right
2. Click on the "Alerts" tab on the left
3. Click on the "Create Alert" button on the top of the screen
4. Create a name for the alert
5. Select which device the alert will be created for
6. Select which layer the alert will be created for
7. Select the value the alert will trigger at
8. Optionally add a description of the alert
9. Click the "Create" button

9.2 Updating the Aether Node Firmware

Occasionally, the firmware running on the Aether Node may need to be updated. This requires the use of an STLink adapter and cable. Additionally, the newest version of the Aether firmware can be found on our GitHub.[14] Updating the firmware also requires that the user setup a local Zephyr environment in order to build and flash the firmware.

Build the Firmware

1. Begin setting up a Zephyr environment by installing a Zephyr SDK an ARM toolchain for your operating system.[15]
2. Run the commands in Listing 1.
3. From within the `aether-workspace/aether/` directory run `west build -p -b aether app`

[14]The Aether Node firmware can be found here: `https://github.com/UCF-Aether/aether-firmware`
[15]https://github.com/zephyrproject-rtos/sdk-ng/releases

Flash the Firmware

1. Connect one end of a micro-USB cable to a computer
2. Connect a micro-USB cable to the STLink
3. Connect one end of the SMD connector to the STLink
4. Remove the top cover of the Aether Node
5. Connect the other end of the SMD connector to the SMD header on the Aether Node
6. Navigate to the `aether-worksapce/aether/` directory.
7. Run `west flash`

Listing 1 Setting up a Zephyr Environment

```
$ west init -m https://github.com/UCF-Aether/aether-firmware aether-
    workspace
$ cd aether-workspace
$ west update
```

9.3 Using the Aether Node Hardware

The Aether Node itself only has a few key features that the end user needs to know about. There are two inputs and one output. There is a USB-C port, a DC jack for the solar panel, and a female SMA connector.

9.3.1 USB-C Port

The USB-C port on the Aether Node is used for charging and serial communication. In this section, we will explain how to perform both actions.

Charging the Aether Node

1. Ensure that the battery is connected.
2. Connect a male USB-C cable to the USB-C port of the Aether Node.
3. Connect the other end of the USB-C cable to a computer or an AC adaptor to a wall outlet.
4. The battery will begin charging and the charging LED will turn on.

Get Debugging Information Over Serial

1. Connect a male USB-C cable to the USB-C port of the Aether Node.
2. Using a UNIX-like operating system, open a terminal.
3. Run a serial terminal utility. for example `picocom -b 115200 /dev/<device_name>`

9.3.2 Solar Panel Connector

There is a DC jack on the Aether Node that is used to connect the solar panel to the device in order to charge the battery. Simply plug the connector coming from the solar panel into the Aether Node. The battery will begin to charge.

9.3.3 SMA Connector

There is a female SMA connector on the Aether Node that is used to connect an antenna for LoRaWAN communication. Simply take an antenna with a male SMA connector and screw it onto the female connector. Ensure that it is fully tighted and then power on the Aether Node.

10 Administrative

In this section, we discuss content related to the planning of our project. This will include items such as budgeting and project milestones. These items are key to any successful project to ensure that the design stays within the desired cost range and that the project is completed in time.

10.1 Estimated Budget

Here we display the budget for each of the major components of our project including the sensor node, one LoRaWAN gateway, and costs related to the server. Since the sensor node is the main design aspect of our project, it is composed of a custom PCB with a multitude of selected ICs and gas sensors. The budget for this sensor node is shown in Table 25. This is in contrast to the LoRaWAN gateway, which we decided we would not be designing. For this component, we simply used a Raspberry Pi, which one of our group members already owned, and a RAK 4600. The budget for the gateway can be found in Table 26. Similarly, we decided that we would not be building and hosting our own server cluster and would instead be using Amazon Web Services under the free pricing tier.

Component	Unit Cost
Solar Panel	$35.00
Battery	$16.95
LoRa-E5	$8.38
PCB	$26.45
Other PCB Components	$55.02
Sensors	$77.13
Enclosure	$3.04
Total	$221.91

Table 25: The budget for the Aether Node.

Component	Approx. Unit Cost	Quantity	Total Cost
Raspberry Pi 4	*free*	1	$0.00
RAK 4600	$120.00	1	$120.00
		Total	$120.00

Table 26: The budget for the gateway.

10.2 Milestones

In Table 27 and Table 29 we describe our current milestones for the fall and spring semesters respectively. The full descriptions for the Fall milestones can be found in Table 28 and the full descriptions for the spring milestones can be found in Table 30. The goal of creating these milestones is to ensure that we stay on track as the semester progresses and that we do not Fall behind. As is often the case, we will likely need to add new milestones as new challenges arise. It is important that we do as much research and design that we can in the first semester, and try to catch anything that might cause significant delays going forward in senior design 2. A big part of any project is figuring out the right requirements, and making sure you understand the problem fully as with anything new, it's going to take more time than originally thought to complete said item. By the end of the Fall semester, we want to have a design that is or is close to being fabricated some time before the start of the Spring semester. Also, it will be important that the backend AWS services be configured and tested with an initial prototype before the first PCB is sent to the fab so that it'll be ready and any bugs caught between sending the design to the fab and receiving it. We are planning on requiring a month and a half to have the second revision done , due to Chinese holidays and to have enough time to thoroughly test the first revision,

10.2.1 Fall Semester

Table 27: Fall Milestones

No.	Milestone	Date	Achieved	Deliverables
F.0	Sensing mechanism finalized	Sep. 2nd, 2021	Sep. 14th, 2021	Sensor requirements document
F.1	MCU and LoRa module finalized	Oct. 1st, 2021	Oct. 1st, 2021	Order confirmation and, if available, shipment information.
F.2	Sensors finalized	Oct. 8th, 2021	Oct. 27, 2021	Order confirmation and, if available, shipment information.
F.3	Prototype v1.0 PCB	Jan. 4th, 2022	-	Gerber files, part and PCB order confirmations
F.4	Software alpha 1.0	Jan. 4th, 2022	-	Node and base-station release binaries, React app, AWS backend code zip package

Note: the paper submission date is December 7th, 2021

Table 28: Fall Milestone Descriptions

No.	Milestone	Description
F.0	Sensing mechanism finalized	Research on how to properly measure the air quality has completed
F.1	MCU and LoRa module finalized	The computing and radio IC(s) have been researched and one (or many) have been chosen
F.2	Sensors finalized	The sensors needed to implement the sensing mechanism in F.0 have been researched and picked
F.3	Prototype v1.0 PCB	The first fully-featured prototype PCB is complete and ready to be sent to the fab
F.4	Software alpha 1.0	A version of firmware that is fully capable of reading sensor data and transmitting them over LoRaWAN to the AWS LoRaWAN network server. The AWS backend should be capable of collating the data and at least be able to tabulate the data. Displaying a map overlay would be a plus.

As said above, around a month and a half will be used to account for delays, shipping speed, and to give ample time to revise the schematic. After the second revision, the design should be as ready as possible for a production quality fab. For the final design, only a three to four weeks will be given for fab time, shipping, assembly, and testing. It will also be imperative that the firmware be capable of sending sensor data in early January, but a fully featured minimal viable product won't be required until later on in early February. The backend services and webapp will have minimal user interaction, but it will be important that the infrastructure created be able to scale for the rest of the scope of the project. The 3D printed enclosure will have an initial design in early January in tandem with the PCB design, as the placement of the sensors is important to getting accurate readings. The final revision of the model will be due at the same time as the rest of the component, on April 11th.

10.2.2 Spring Semester

Table 29: Spring Milestones

No.	Milestone	Date	Deliverables
S.0	Model and 3D print prototype enclosure	Jan. 30, 2022	3D model
S.2	MVP Firmware	Feb. 13th, 2022	Gerber files, part and PCB order confirmations
S.1	MVP Webapp and supporting backend infrastructure	Feb. 13th, 2022	React app, AWS backend zip package
S.3	PCB Revision 1.1	Feb. 13th, 2022	Gerber files, part and PCB order confirmations
S.4	Software alpha 1.1	Mar. 6th, 2022	Node and base-station release binaries, React app, AWS backend code zip package
S.5	Finalized PCB	Mar. 13th, 2022	Gerber files, part and PCB order confirmations
S.6	Stretch goal completion	Apr. 1st, 2022	Node and base-station release binaries, React app, AWS backend code zip package
S.7	Finalized design	Apr. 11st, 2022	Code binaries, 3D model, Gerber files

Table 30: Spring Milestone Descriptions

No.	Milestone	Description
S.0	Model and 3D print prototype enclosure	A 3D model has been designed to house the PCB, and has been printed
S.2	MVP Firmware	The firmware will be able to take sensor measurements from all sensors and upload them to the AWS backend over LoRaWAN
S.1	MVP Webapp and supporting backend infrastructure	The webapp will be able to allow users to view the map with sensor data overlays. Users can sign up to register their devices with the AWS LNS
S.3	PCB Revision 1.1	Fixes to the PCB design, and new prototype design has been ordered
S.4	Software alpha 1.1	Any bug fixes and stabilization, and a CLI and/or TUI Linux program has been created to interact with the device over USB
S.5	Finalized PCB	Any remaining fixes and final touches to the PCB, and ordered with high quality materials and production fab
S.6	Stretch goal completion	Complete as many stretch goals as possible. Ideally, integration with The Things Network and Helium would be nice, along with additional webapp features, such as: controlling devices remotely, viewing device logs, and view time-series graphs and map overlays of sensor data
S.7	Finalized design	Any final bug fixes to firmware and software, along with a finalized enclosure design with pivot-able solar panel fixture. All requirement specifications should have been met, along with as many stretch goals as possible, such as wired (USB) data uploads

11 Conclusion

The goal of our project was to design a sensor network that could monitor the concentration of various pollutants and calculate the AQI. The sensor node would be able to transmit this information using a LoRaWAN network to a remote server where it would be displayed on a website.

In order to meet this design goal, we first came together as a group to determine a set of requirements. Out of all the requirements we determined, the three most important ones that we established were that the Aether Node shall calculate the AQI within ±25% of the EPA's reported value, the Aether Node shall have an average power draw of less than 100 mW, and the Aether Node shall automatically take a sensor reading from each sensor at a specified time interval.

We knew that to create a design that met these requirements, we would need to perform more research in variety of areas. One of these areas we spent a lot of time researching was in the LoRaWAN standard itself. We familiarized ourselves with the benefits and limitations of the technology. Another key research area for us was in air pollution. Given that one of our design goals was to report on air quality and different air pollutants, we knew we would have to learn more about the various types of air pollution to know what types of sensors to purchase. Finally, we spent the majority of our research time on picking the components of our design. While not all components were fully chosen before working on the design, major components such as the LoRa networking and compute module and gas sensors were critical and the design was based around them, so these components were decided upon early.

After performing research, we determined that we would require three main components for our project to function: a sensor node, a LoRaWAN gateway, and a web server. Due to time and economic constraints, we decided to only design the sensor node and the website. This meant that we created a custom PCB and wrote all of the embedded software for the Aether Node. For the gateway, we decided to use off-the-shelf components. Those components being a Raspberry Pi and a RAK 2245 LoRaWAN gateway module. We also used open source software for running on the gateway. For the server, we would used the free-tier version of an Amazon Web Services instance running AWS IoT Core as well as Supabase. We also designed the web application that would display data collected from the sensors and that the user would interact with using React.

Next, we began the process of constructing the Aether Node prototype, as well as building the website. We started by writing all the firmware on a LoRa-E5 development board. At the same time, we were designing the circuit schematics and working on the PCB. Once the circuit was designed, we ordered all of the components. Once the PCB was designed, we placed an order for it. When everything was received, we assembled it. After this, we were able to test and debug the firmware on our PCB. Nearing the end of our project, we finalized the enclosure design and

had it 3D-printed. Multiple revisions of the enclosure had to be made due to design issues, but eventually we were successful. At this point the Aether Node itself was finished. Throughout the design and building of the Aether Node, we were simultaneously building and iterating on the website, which we completed around the same time as the completion of the Aether Node.

This two-semester project has been a valuable experience to grow both our team's technical knowledge as well as our ability to effectively function as members of larger project. The end result of our multiple months of effort was a fully- functioning prototype that met our requirements. Our team was able to successfully build both Aether Node, which included a custom PCB and enclosure, as well as the Aether website, which allowed users to view the data collected by the Aether Node.

References

[1] IEEE, "IEEE standard for information technology– local and metropolitan area networks– specific requirements– part 15.4: Wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (WPANs): Amendment 1: Add alternate PHYs," *IEEE Std 802.15.4a-2007 (Amendment to IEEE Std 802.15.4-2006)*, pp. 1–210, 2007. DOI: 10.1109/IEEESTD.2007.4299496.

[2] A. Hajat, C. Hsia, and M. S. O'Neill, "Socioeconomic disparities and air pollution exposure: A global review," *Springer International Publishing*, pp. 440–450, 2015. DOI: 10.1007/s40572-015-0069-5.

[3] H. Orru, K. Ebi, and B. Forsberg, "The interplay of climate change and air pollution on health," *Current Environmental Health Reports*, vol. 4, pp. 504–513, 2017. DOI: 10.1007/s40572-017-0168-6.

[4] M. Chiani and A. Elzanaty, "On the lora modulation for iot: Waveform properties and spectral analysis," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8463–8470, Oct. 2019, ISSN: 2372-2541. DOI: 10.1109/jiot.2019.2919151. [Online]. Available: http://dx.doi.org/10.1109/JIOT.2019.2919151.

[5] I. Manisalidis, E. Stavropoulou, A. Stavropoulos, and E. Bezirtzoglou, "Environmental and health impacts of air pollution: A review," *Frontiers in Public Health*, vol. 8, 2020. DOI: 10.3389/fpubh.2020.00014.

[6] E. P. Agency. (). "Technical assistance document for reporting of daily air quality," [Online]. Available: https://www.airnow.gov/publications/air-quality-index/technical-assistance-document-for-reporting-the-daily-aqi/.

[7] L. Alliance. (). "What is lorawan specification," [Online]. Available: https://lora-alliance.org/.

[8] S. Banerjee. (). "Server vs serverless cloud computing-comparing amazon ec2 and aws lambda," [Online]. Available: https://sudiptobanerjee14.medium.com/server-vs-serverless-cloud-computing-comparing-amazon-ec2-and-aws-lambda-f3684e9b840a.

[9] R. Barry. (). "Mastering the freertos real time kernel - a hands on tutorial guide," [Online]. Available: https://www.freertos.org/Documentation/RTOS_book.html.

[10] (). "Battery cell comparison," [Online]. Available: https://www.epectec.com/batteries/cell-comparison.html.

[11] (). "Best monocrystalline solar panels," [Online]. Available: https://alternative-energy-sourcesv.com/best-monocrystalline-solar-panels/.

[12] C. Burnett. (). "Serial peripheral interface," [Online]. Available: https://en.wikipedia.org/wiki/Serial_Peripheral_Interface#/media/File:SPI_three_slaves.svg.

[13] (). "Charging lithium ion," [Online]. Available: https://batteryuniversity.com/article/bu-409-charging-lithium-ion.

[14] Cypress. (). "Cypress cyb06447bzi-bldx datasheet," [Online]. Available: https://www.cypress.com/file/501811/download.

[15] (). “Differences between lithium and lead acid batteries,” [Online]. Available: https://ferkart.com/leadacid-vs-lithium/.
[16] (). “Differences between lithium ion and lead acid batteries,” [Online]. Available: https://ferkart.com/leadacid-vs-lithium/.
[17] EPA. (). “Basic information about carbon monoxide (co) outdoor air pollution,” [Online]. Available: https://www.epa.gov/co-pollution/basic-information-about-carbon-monoxide-co-outdoor-air-pollution#Effects.
[18] ——, (). “Health and environmental effects of particulate matter (pm),” [Online]. Available: https://www.epa.gov/pm-pollution/health-and-environmental-effects-particulate-matter-pm.
[19] ——, (). “Health effects of ozone in the general population,” [Online]. Available: https://www.epa.gov/ozone-pollution-and-your-patients-health/health-effects-ozone-general-population.
[20] ——, (). “Nitrogen dioxide basics,” [Online]. Available: https://www.epa.gov/no2-pollution/basic-information-about-no2#What%20is%20NO2.
[21] ——, (). “Sufur dioxide basics,” [Online]. Available: https://www.epa.gov/so2-pollution/sulfur-dioxide-basics.
[22] ——, (). “What is ozone?” [Online]. Available: https://www.epa.gov/ozone-pollution-and-your-patients-health/what-ozone.
[23] ——, (). “What is particulate matter?” [Online]. Available: https://www.epa.gov/pm-pollution/particulate-matter-pm-basics.
[24] A. EPA. (). “How smoke from fires can affect your health,” [Online]. Available: https://www.airnow.gov/air-quality-and-health/how-smoke-from-fires-can-affect-your-health/.
[25] ——, (). “Technical assistance document for the reporting of daily air quality,” [Online]. Available: https://www.airnow.gov/publications/air-quality-index/technical-assistance-document-for-reporting-the-daily-aqi/.
[26] (). “Find out how to prolong battery life by using correct charge methods.,” [Online]. Available: https://batteryuniversity.com/article/bu-409-charging-lithium-ion.
[27] (). “Github - usepa/o3-nowcast,” [Online]. Available: https://github.com/USEPA/O3-Nowcast/tree/master.
[28] (). “How is the nowcast algorithm used to report the current air quality?” [Online]. Available: https://usepa.servicenowservices.com/airnow?id=kb_article_view&sys_id=bb8b65ef1b06bc10028420eae54bcb98&spa=1.
[29] (). “How to charge a lead acid battery,” [Online]. Available: https://www.power-sonic.com/blog/how-to-charge-a-lead-acid-battery/.
[30] (). “I2c info - i2c bus, interface and protocol,” [Online]. Available: https://www.i2c.info/.
[31] T. T. Industries. (). “The things community network documentation,” [Online]. Available: https://www.thethingsnetwork.org/docs/.
[32] D. Instruments. (). “Davis airlink,” [Online]. Available: https://www.davisinstruments.com/pages/airlink.
[33] (). “Lead acid charging,” [Online]. Available: https://en.wikipedia.org/wiki/File:Lead-acid_charging.svg.

[34] (). "Lithium ari battery," [Online]. Available: https://www.flickr.com/photos/argonne/9969637604.
[35] (). "Lithium ion battery vs lead acid battery," [Online]. Available: https://www.twmtraffic.com/lithium-ion-battery-vs-lead-acid-battery/.
[36] (). "Lithium ion soc," [Online]. Available: https://www.powertechsystems.eu/home/tech-corner/lithium-ion-state-of-charge-soc-measurement/.
[37] (). "Lorawan network stack," [Online]. Available: https://www.researchgate.net/figure/Architecture-of-a-LoRaWAN-Network_fig1_341834399.
[38] MClimate. (). "Mclimate aqi sensor and notifier lorawan," [Online]. Available: https://docs.mclimate.eu/mclimate-lorawan-devices/devices/mclimate-aqi-sensor-and-notifier-lorawan.
[39] N. L. of Medicine. (). "Compound summary: Nitrogen dioxide," [Online]. Available: https://pubchem.ncbi.nlm.nih.gov/compound/Nitrogen-dioxide.
[40] ——, (). "Compound summary: Sulfur dioxide," [Online]. Available: https://pubchem.ncbi.nlm.nih.gov/compound/sulfur-dioxide.
[41] Midatronics. (). "Mindatronics mkr windy datasheet," [Online]. Available: https://midatronics.com/wp-content/uploads/2021/02/UG_MDX-MKR-STWLx_1.1.pdf.
[42] ——, (). "Mindatronics windy datasheet," [Online]. Available: https://midatronics.com/wp-content/uploads/2021/02/Midatronics_Windy-1.3.pdf.
[43] (). "Monocrystalline vs polycrystalline solar panels: Whats best?" [Online]. Available: https://www.ecowatch.com/monocrystalline-vs-polycrystalline-solar-panels-2654716551.html.
[44] Onethinx. (). "Onethinx core datasheet," [Online]. Available: https://github.com/onethinx/Core_Datasheet/raw/master/Onethinx_Core_Specification.pdf.
[45] (). "Prt-13856," [Online]. Available: https://www.digikey.com/en/products/detail/sparkfun-electronics/PRT-13856/6605200.
[46] Qoitech. (). "How spreading factor affects lorawan device battery life," [Online]. Available: https://www.thethingsnetwork.org/article/how-spreading-factor-affects-lorawan-device-battery-life/.
[47] Renesas. (). "Gas sensor module for oaq targeting no2 & o3," [Online]. Available: https://www.renesas.com/us/en/document/dst/zmod4510-datasheet?r=463746.
[48] (). "Secondary cell energy density," [Online]. Available: https://commons.wikimedia.org/wiki/File:Secondary_cell_energy_density.svg.
[49] Semtech. (). "Semtech sx1261/2 datasheet," [Online]. Available: https://semtech.my.salesforce.com/sfc/p/#E0000000JelG/a/2R000000HT7B/4cQ1B3JG0iKRo9DGRkjVuxclfwE3tfSUcGr.S_dPd4.
[50] Sensirion. (). "Particulate matter sensor for air quality monitoring and control," [Online]. Available: https://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/9.6_Particulate_Matter/Datasheets/Sensirion_PM_Sensors_Datasheet_SPS30.pdf.
[51] A. W. Services. (). "Freertos reference manual," [Online]. Available: https://www.freertos.org/Documentation/RTOS_book.html.

[52] ——, (). “Serverless on aws,” [Online]. Available: https://aws.amazon.com/serverless/.
[53] (). “Solar charge controller basics,” [Online]. Available: https://www.solar-electric.com/learning-center/solar-charge-controller-basics.html/#:~:text=What%5C%20is%5C%20a%5C%20Solar%5C%20Charge,panels%5C%20going%5C%20to%5C%20the%5C%20battery.&text=Most%5C%20batteries%5C%20need%5C%20around%5C%2014%5C%20to%5C%2014.5%5C20volts%5C%20to%5C%20get%5C%20fully%5C%20charged..
[54] (). “Solar panel construction,” [Online]. Available: https://www.cleanenergyreviews.info/blog/solar-panel-components-construction.
[55] (). “Solar panel types,” [Online]. Available: https://solar2power.pt/solar-panel-types-mono-poly-and-thin-film-pv-modules/.
[56] STMicroelectronics. (). “Stm32wle5jc datasheet,” [Online]. Available: https://www.st.com/resource/en/datasheet/stm32wle5j8.pdf.
[57] S. Studio. (). “Seeed lora-e5,” [Online]. Available: https://files.seeedstudio.com/products/317990687/res/LoRa-E5%20module%20datasheet_V1.0.pdf.
[58] C. Svec. (). “Freertos,” [Online]. Available: https://aosabook.org/en/freertos.html.
[59] M. Technology. (). “Microchip atsamr34j18 datasheet,” [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/SAM-R34-R35-Low-Power-LoRa-Sub-GHz-SiP-Data-Sheet-DS70005356C.pdf.
[60] ——, (). “Microchip wlr089u0 datasheet,” [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/70005435B.pdf.
[61] (). “Thin film solar cell,” [Online]. Available: https://en.wikipedia.org/wiki/Thin-film_solar_cell.
[62] (). “Types of solar cells and application,” [Online]. Available: https://www.researchgate.net/figure/Structure-of-thin-film-solar-cells_fig3_281148723.
[63] (). “Usb-c,” [Online]. Available: https://en.wikipedia.org/wiki/USB-C.
[64] M. VanderVoord, M. Karlesky, and G. Williams. (). “Unity - unit testing for c,” [Online]. Available: http://www.throwtheswitch.org/unity.
[65] ——, (). “Unity test - github,” [Online]. Available: https://github.com/ThrowTheSwitch/Unity.
[66] (). “What is the manufacturing principle of silicon solar cell,” [Online]. Available: https://www.dsneg.com/info/what-is-the-manufacturing-principle-of-solar-c-36849269.html.
[67] R. Wireless. (). “Rak4600 wisduo datasheet,” [Online]. Available: https://docs.rakwireless.com/Product-Categories/WisDuo/RAK4600-Module/Datasheet/#overview.

A Appendix

A.1 Image Permissions

- Figure 4: Used with permission from [30].
- Figure 5: Used with permission under the CC BY-SA 3.0 license [12].
- Figure 7: Used with permission under the Creative Commons Attribution 4.0 International license [37].
- Figure 10: Used with permission under the CC BY 3.0 license [33].
- Figure 11: Used with permission under the CC BY 3.0 license [34].
- Figure 12: Used with permission under the CC BY-NC 3.0 license [36].
- Figure 13: Used with permission under the Creative Commons Attribution 4.0 International license [48].
- Figure 14: Used with permission from [15].
- Figure 15: Used with permission from [26].
- Figure 16: Used with permission from [29].
- Figure 17: Used with permission under the Creative Commons Attribution 4.0 International license [45].
- Figure 18: Used with permission from [55].
- Figure 19: Used with permission from [66].
- Figure 20: Used with permission from [11].
- Figure 21: Used with permission under the CC BY-NC 3.0 license [54].
- Figure 22: Used with permission from [62].

A.2 Listings

Listing 2 Example Unity Test Report Output [64]

```
testUnit1.c:21:test_AverageThreeBytes_should_AverageMidRangeValues:PASS
testUnit1.c:22:test_AverageThreeBytes_should_AverageHighValues:PASS
-----------------------
2 Tests 0 Failures 0 Ignored
OK
```

Listing 3 Unity Basic Assertion Example [64]

```
int a = 1;
TEST_ASSERT( a == 1 ); //this one will pass
TEST_ASSERT( a == 2 ); //this one will fail
```

Listing 4 More advanced Unity Assertions [65]

```
TEST_ASSERT_EQUAL_INT(2, a);
TEST_ASSERT_EQUAL_HEX8(5, a);
TEST_ASSERT_EQUAL_UINT16(0x8000, a);
TEST_ASSERT_EACH_EQUAL_INT(expected, actual, num_elements);
TEST_ASSERT_MESSAGE( a == 2 , "a isn't 2, end of the world!");
```

Listing 5 Jest Unit Testing Example

```
function calc() {
  return 50;
}

test('calc() test', () => {
  expect(50).toEqual(50);
})
```

Listing 6 Jest Unit Testing Example Output

```
> jest
 RUNS  ./calc.test.js
 PASS  ./calc.test.js
   calc() test (2 ms)
```

```
Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.445 s, estimated 1 s
Ran all test suites.
```