Breath of Fresh Air

Authors & Team Members:
Ian Anderson
Jonathan Francis
Connor Redmon
CPEN498: Computer Engineering Capstone
Department of Physics, Computer Science and Engineering
Christopher Newport University

Final Project Status Report
(August 22, 2016 – April 19, 2017)

1. Abstract

The product created is an ambient air quality sensor network that is capable of providing information regarding four potentially hazardous greenhouse gases. The gases monitored by the product are CO2, NO2, O3, and Particulate Matter (PM). The sensors are part of a network of nodes that are constructed with Arduino Uno microcontrollers and Xbee communication devices. The Xbee modules are compatible with the Arduino microcontrollers via the Xbee shields. Data collected from the sensor nodes is relayed through a router node to the home node which is constructed with a Raspberry Pi 2. Data received by the home node is uploaded to a server database. A user interface then allows approved users to view the current data collected and provides the ability to create tables, graphs, and reports on the data based on parameters given.

2. Problem Statement

A major problem that has started to affect the billions of people that inhabit Earth is air pollution. In 2012, one out of every nine deaths was the result of air pollution-related conditions [1]. Of those deaths, around 3 million are attributed solely to ambient outdoor air pollution [1]. Health problems stemming from air pollution can range from acute lower respiratory disease and chronic obstructive pulmonary disease to strokes, ischemic heart disease and lung cancer [1]. Christopher Newport University promotes a healthy and sustainable environment for all university members to enjoy. This project will create an ambient air quality sensor network that will monitor for harmful pollutants in the air surrounding the university. This project is important and beneficial to the university in ways that it could provide CNU with the knowledge it needs to maintain a safe working environment for the coming years.

The final product from this project could be used in multiple ways to monitor the surrounding air quality. CO_2, NO_3, O_3, and Particulate Matter (PM) all contribute to air pollution as a result of road traffic and energy production [10]. With the Newport News Shipyard and Langley Air Force Base (LAFB) in the surrounding area, air pollution from facility production and air traffic is potentially a major threat to the surrounding population. With air pollution constantly being emitted by cars, trucks, the shipyard and LAFB, the system being developed could monitor air quality to ensure the levels of CO_2, NO_2, O_3, and PM do not reach an excessive amount. If any of the gases were to reach hazardous levels, the user would be able to look at the data output from the system and relay that information to a city official who could determine the next course of action.

Although the system is being built as an outdoor air quality network, with a few minor changes, a future capstone group could make the system compatible for indoor use. The gas sensors would need to be updated and swapped out with more relevant indoor hazardous gas sensors to make this network work properly and provide useful information to the user.

3. Requirement Analysis

In order for this project to be seen as a success, the following requirements must be met:

3.1 Nodes

- Four working nodes
    - One central node will be located within Luter Hall to collect all incoming data from the sender nodes, which will then be uploaded to a server database for users to view the information. The Sustainability Coordinator, Ryan Kmetz, suggested a central node so later he could potentially move the node to his office in Forbes.
    - Two data collection nodes, with two sensors attached to each node, will be placed in two locations across the CNU campus. These nodes will also need to be weather resistant so no damages can occur while they are in their location collecting data.
    - The nodes should have a working distance of about 1000 feet with a direct line of sight. Due to the quantity of buildings and trees on CNU's campus, 1000 feet is the max distance estimated when relaying the information collected back to Luter Hall.

3.2 Data Collection:

- Data collection from the sensor nodes back to the home node should be completed while the power source batteries contain enough voltage and mAh to keep the nodes on. Once the batteries die, the user can then unplug them from the node, recharge them, and move the nodes to a new location for new results.
- The user should have the ability to view a data sample in real time. Data samples in real time are taken every 30 seconds and then relayed back to the home node so that information can be uploaded.

3.3 User Interface/ Database:

- The administrative side of the user interface will need to be accessible to those who have authorization and the data collected should be presented in an organized manner.
- Authorized users will be able to create graphs, tables, and reports using the data.

3.4 Power

- The power source for the nodes should have the capacity to last about a week. It should supply only the needed current to operate the node at 100% efficiency and should rechargeable so the user does not have to continuously buy new batteries when the power source dies.

4. Design Alternatives

4.1 Hardware:

4.1.1 Microprocessors:

The microprocessor is the brains of the sender and receiver nodes. Besides the communication aspect of the network, this was the most important component. For the sender node, the project requires a small and powerful board that does not consume a lot of power. Another key component the board needs to fulfill is compatibility with each air quality sensor. As for the receiver node, the team needs something that is able to connect directly to the Internet. The two boards that have been considered for the sending and receiving nodes were the Arduino Uno Rev3 and the Raspberry Pi 2. These two boards were chosen for the low price and the widely useful and helpful tutorials for novice Arduino and Raspberry Pi users.

Design Alternative 1: Arduino Uno Rev3

The Arduino Uno is a great microprocessor because it provides a powerful board that meets all the project's requirements for a microprocessor, at a low cost. With a cost that ranges from 20 to 30 US dollars it is one of the cheapest experimentation boards on the market. The Uno is an 8-bit microcontroller board that is based off the ATmega328P design. It has 32KB of flash memory and 2KB of RAM located on the board. The board has 6 analog input/output ports, which are more than enough for the sensors that will be attached to the board. The Uno runs at an operating voltage of 5V. Its dimensions are 68.6mm x 53.4mm, making it easy to cover and protect, as well as accessible. Figure 1 shows the layout of the Arduino Uno Rev3 [2].

However, despite fulfilling most of requirements the team needs, the Arduino does not include direct access to the Internet. In order for the Arduino Uno to be compatible with the Internet, an aftermarket adapter would have to be purchased. Because of this drawback, the team decided to not use the Arduino for our receiver node. Fortunately, the low cost, low power, and easy compatibility with the sensors, made it a great choice for the board being used as part of the sender nodes.

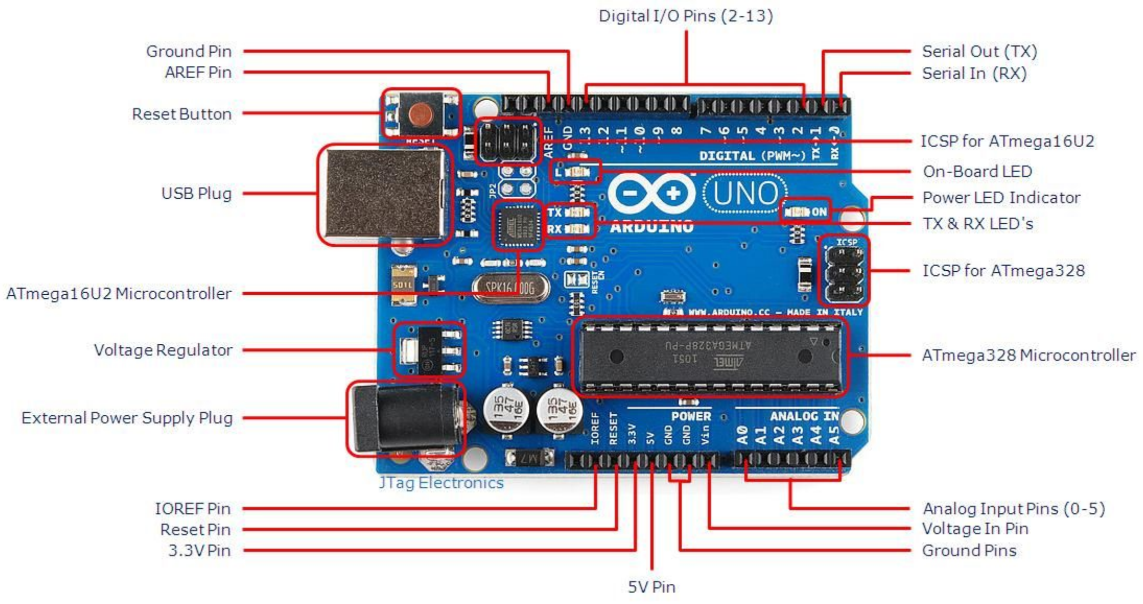


Figure 1: Layout of the Arduino Uno Rev3

Design Alternative 2: Raspberry Pi 2

The project requires that the data collected from the sender nodes be directly uploaded to a cloud database. Knowing this, the team looked for a microcontroller that could easily connect to and access the Internet. The Raspberry Pi 2 does that and much more. The RPI2 operates using a 900MHZ quad core ARM Cortex- A7 CPU with 1GB of RAM. It also has 4 USB ports, a full HDMI port, and most importantly, an Ethernet port. This will allow us to upload our data readings to our cloud database. Figure 2 shows the layout of the Raspberry Pi 2 [3].

The team decided to not use the RPI2 for the sender nodes because it was simply more than we needed. The Arduino Uno offers everything we wanted at a cheaper cost. The price difference between the two is about 10 US dollars with the RPI2 being about $35 and the Arduino Uno being $25. As for the receiver node, the Raspberry Pi 2 was a perfect fit. It provided direct access to the internet allowing us to upload incoming data directly to a cloud database. For this reason, we selected the Raspberry Pi 2 for the receiver node.

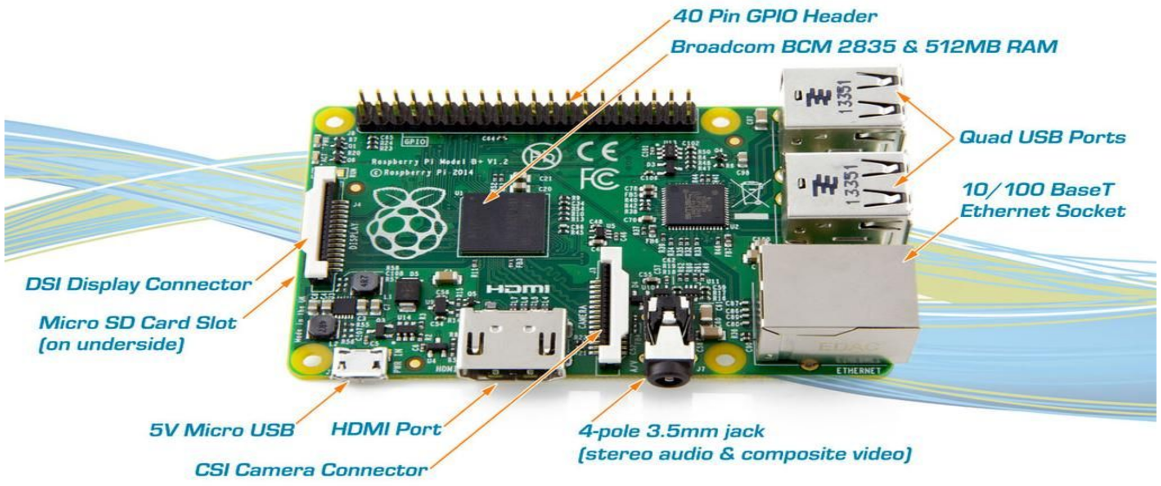

40 Pin GPIO Header
Broadcom BCM 2835 & 512MB RAM
Quad USB Ports
10/100 BaseT Ethernet Socket
DSI Display Connector
Micro SD Card Slot (on underside)
5V Micro USB
HDMI Port
CSI Camera Connector
4-pole 3.5mm jack (stereo audio & composite video)

Figure 2: Layout of the Raspberry Pi 2

4.2 Communication:

The communication aspect is a huge part of the project. Without a stable communication link between the sender and receiver, there is no sensor network. In order to fulfill the requirements from the client, the team had to find a way the nodes can communicate wirelessly. The team needed something that is cheap and can reach up to distances of a mile. The three options considered by the team were Wifi, Bluetooth, and Zigbee Xbee.

Design Alternative 1: WiFi Communication

WiFi is one of the more stable forms of linking and communicating between devices as long as each device has access to the network. One advantage immediately considered when researching WiFi was the ability for the nodes to directly upload from their location without needing a central node. However, Arduino microprocessors do not come Internet compatible, so if the Arduino was chosen as the sensor node microprocessor, the acquisition of an Arduino WiFi shield would be necessary. Arduino WiFi shields are $49.95, not including shipping and tax. If the Arduino shields were purchased and WiFi was inaccessible in the areas of choice for the nodes, this would be a waste of $200. When considering WiFi, the first concern was the collector nodes reliability to connect to the WiFi network. With locations such as the baseball field and the Ferguson Center parking garage, WiFi reliability in those locations were questionable, creating one of the multiple problems WiFi presented. WiFi testing was done over in the parking lot of the baseball field and no WiFi signal could be picked up. Another reliability issue with WiFi was having to use CNU's WiFi network. CNU's WiFi is not the most reliable network as it crashes at times and bandwidth speeds vary each day depending on the location and the number of people using the network in said location. This did not seem like an inconvenience that would be handled well within the network.

Design Alternative 2: Bluetooth Communication

Bluetooth is a standardized protocol for sending and receiving information wirelessly. A Bluetooth Shield for an Arduino costs $20 which is cheaper than the WiFi shield but the main issue presented when looking into Bluetooth communication was the range at which Bluetooth functions properly. With collector nodes being roughly 0.5 miles away from the central node in some cases, this range seemed slightly excessive for the Bluetooth link to reach. Bluetooth protocol states that Bluetooth is an excellent choice for transferring data over a short range of less than 300 yards [8]. If Bluetooth communication was chosen, the collector nodes would have to be strategically placed around the Great Lawn of the campus to ensure the central node could receive all the information needed to present logical and useful information to the user. The problem with this, the Great Lawn is not a vast area of the CNU campus. The data would be relatively similar in all places around the Great Lawn meaning the data would not be an accurate representation of the entire campus and its surroundings.

Design Alternative 3: Xbee Pro Series 1

The Xbee Pro 60mW Wire Antenna (802.15.4) runs at an operating voltage of 3.3V @ 215mA. The Series 1 allows for a max data rate of 250Kbps. It has 6 10-bit ADC input pins and 8 digital IO pins. Each Xbee is easily configured using the XCT-U software that comes with it. This software allows consumers to set up a connection between multiple Xbee modules [9]. XBee Pro Series 1 modules have the capability of ranging up to one mile which is a suitable range for the collector nodes. XBee modules are also relatively inexpensive and offer compatibility to the Arduino Uno microprocessors. An XBee Pro Series 1 module costs $37.95 and with the XBee shield to make them Arduino compatible being priced at $14.95, the total for the combined module is $52.90. Although priced $2.90 more than the WiFi shield, the XBee modules offer a more reliable network link to transmit data. XBee modules also offer their own software to help set up the network as the user wishes. Each individual XBee can be set as a

Coordinator, Router, or End Device in the network. This allows full manipulation of the network to ensure data transfer follows the correct path. The only issue found with the XBee modules is the interference of walls and buildings. Because of this, router nodes are needed to ensure the data transfer from the collector nodes can be completed successfully back to the coordinator node. For the reasons of ease of network setup, cost and manipulability, the XBee modules were chosen as the communication link for the network. Figure 3 is a model representation of the XBee modules that will be used.

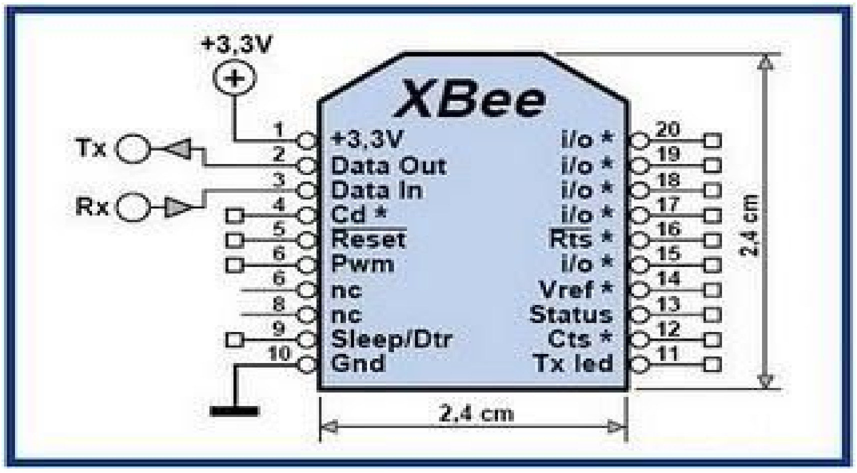


Figure 3: Pin out of an Xbee Pro Series 1

4.3 Power Supply

Design Alternative 1: 400-500k mAh power bank

For our power supply we chose two different power banks. One power bank was responsible for powering our sensor nodes and the other one provides power for our router node. Each of our sensor nodes require 400-500 mA of current. Our first idea was to have a power bank that would supply the nodes for close to a month without dying. Unfortunately, this required a power bank with about 500,000 mAh and these were on the market for $80-$100. After realizing these power banks were out of our budget, we began looking for cheaper alternatives. We found the RavPower Portable Charger. This power bank contains 22,000 mAh and supplies our nodes with a 5V output. Using their iSmart technology, the power bank will automatically be able to connect to our nodes and supply them with the optimal current. This battery, fully charged, will supply our nodes with power for approximately 65 hours.

Design Alternative 2: 9V battery

The second battery we chose was for our router node. This power bank did not require as many mAh as the first one because our router node does not consume as many milli Amps. The team began looking into the 9V battery for the router node. After some minor testing, we realized the 9V battery would not be practical as it would die within hours of initial use. The power bank we chose was iKits Panasonic Battery Fast Charge Power Bank. This power bank contains 10,200 mAh and supplies a 5V output. This bank, like the previous, will detect our node and provide it with an optimal current. Using this battery, we can power our router node for approximately 90-100 hours.

5. Plan for Solution

The final result of the network should produce an ambient air quality sensor network consisting of two sensor nodes, one routing node, and one coordinator/home node. Figure 4 is a diagram representation of the final product.

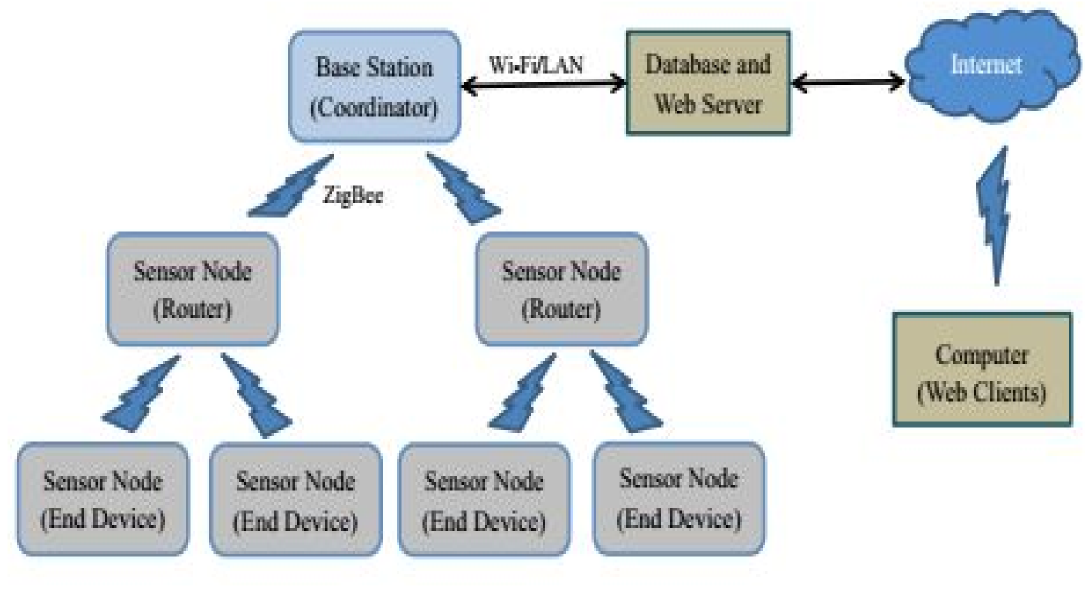


Figure 4: Model of System Diagram

The final product has been produced over the course of the previous academic school year. Some parts of the final product have been worked on during the first semester such as creating two nodes, one for receiving data and one for sending data. The nodes were constructed using the Arduino Uno microprocessors, the XBee shields, and the XBee Pro Series 1 modules. The nodes were produced by inserting the legs of the Xbee shield into the corresponding ports on the Arduino. Next, the Xbee legs were inserted into the corresponding ports on the Xbee shield and this created our nodes.

Using the nodes already constructed, a gas sensor was wired to the sender node using the analog pin outs from the Arduino/XBee shield combination. The Arduino IDE is being used to write scripts that sends the information collected from the gas sensor on the sender node to the coordinator node.

When the two sender nodes were constructed and each had 2 gas sensor wired to the Arduino, the Arduino IDE was used to write more advanced scripts that will call the sender nodes to relay the information gathered from the sensor and send it back to the home node.

The sender nodes are powered with a battery to power the Arduino and the gas sensor. The node will be placed in sleep mode, using the internal interrupt service routine, when data is not being acquired by the sensor. This way the battery will last as long as possible and will not need to be changed every couple of days. If the user wishes to get a real time reading of the gas values, this should be done from the home node, calling one node at a time to acquire the data and send it back to the home node.

Data acquired from the sender nodes will then be uploaded to a database using the Raspberry Pi 2. The Arduino IDE will be used to write scripts that will take the data from the home node and present it in a spreadsheet type format. This will make the data presentable for the user to understand and should have the ability to print this data if they choose.

The database in which the gas sensor data will be stored will only be accessible to those who have authorization. Those being granted authorization to the back end of the server will be Dr. Gerousis, Dr. Riedl, Ryan Kmetz, Jonathan Francis, Ian Anderson, and Connor Redmon. This was done by creating a login where an authorized user must sign into in order to gain access to the backend of the database.

5.1 Choice of Sensors:

For the project requirements to be fulfilled, the sensor network needs to measure CO_2, NO_2, O_3, and PM (Particulate Matter) clearly and reliably. A concern the team had about sensors was to make sure each is compatible with the microprocessor on the sender nodes (Arduino Uno). Since the sensors had to be compatible with the Arduino Uno, this limited our design alternatives as there is usually only a single compatible sensor. Addressing these concerns, the team found the MQ series of sensors. The MQ series is compatible with the Arduino Uno Rev3.

5.1.1 Sensor 1: MG-811 Carbon Dioxide Sensor

The first sensor the team found was the MG-811 Carbon Dioxide Sensor Module for Arduino [4]. This is a compact board that provides a fully functional Carbon Dioxide sensor on it. The sensor allows for both digital and analog output. This is perfect for the team because the Arduino Uno has many digital and analog ports. The board connects easily to the microprocessor with a quick plug in. This allows the client to be able to remove and possibly put a new sensor on the node.The sensors output raw voltage readings that are used to help determine the concentration of the gas being tested for. The output voltage of the sensor in clean air is typically 100-300 ppm CO2. This is the baseline voltage or V_0. If the concentration of CO_2 in the air rises then the voltage will decrease. When the concentration of CO2 is greater than V_0, the output voltage (Vs) is linear to the common logarithm of the CO2 concentration:

$$\text{Eq 1. } Vs = V_0 + \Delta Vs / (\log_{10}400 - \log_{10}1000) * (\log_{10}C_{CO2} - \log_{10}400)$$

5.1.2 Sensor 2: MQ-131 O_3 Sensor

For O_3 sensor the team decided to go with the MQ-131 sensor [5]. Like all the sensors, it is a small compact board with a working sensor on it. The board is directly compatible with the Arduino Uno Rev3. Unlike the MG-811, this sensor does not have easy connectable cables. The board allows for analog and TTL-level output. The MQ-131 works very similarly to the MG-811. It outputs a raw voltage reading that is used to find the concentration of O_3 in the air.

As the voltage gets higher the concentration will get higher as well (LOOK AT CODE TO DETERMINE). The voltage causes the concentration to get higher. Figure 5 is a picture of the MQ-131 sensor that is being used.

Figure 5: the MQ-131

5.1.3 Sensor 3: Sharp GP2Y1010AU0F Compact Optical Dust Sensor (PM)

The PM sensor the team decided on is the Sharp GP2Y1010AU0F Compact Optical Dust Sensor [6]. Originally this sensor can not able to directly attach to the Arduino Uno Rev3. However, recently the manufacturer realized the demand for a direct connect to the Arduino and produced a dust sensor adaptor. This allows for easy connect and disconnect from an Arduino. This sensor has very low power consumption. At max, the sensor uses 20mA. Usually though, the sensor stays somewhere around 11mA @ 7V. Like all of the sensors, this one uses an analog output for data. In the sensor there is an infrared emitting diode and a phototransistor that are diagonally arranged, allowing the sensor to detect the reflected light of dust in the air. Figure 6 is a picture of the Sharp Optical Dust Sensor that is being used.

Figure 6: Sharp GP2Y1010AU0F Compact Optical Dust Sensor

5.1.4 Sensor 4: Nitrogen Dioxide (NO2) Sensor, MiCS-2714

The last sensor the team decided to use is the Nitrogen Dioxide (NO2) Sensor, the MiCS-2714 [7]. This sensor has an operating voltage of 1.7V. It uses analog output to send sensor data to its microprocessor. Like all of the sensors, except the Sharp GP2Y1010AU0F Compact Optical Dust Sensor, it uses voltages to measure the concentration in the air. Figure 7 is a picture of the MiCS-2714 sensor that will be used.

Figure 7: MiCs-2714 Nitrogen Dioxide Sensor

5.2 Software Tools:

Understanding the software behind the hardware components was crucial to the project. Creating the programs for each component to perform the functions needed from establishing connections between the nodes to manipulating the data so understandable readings. IDE’s and easy to use languages were used to help build the functionality of the hardware and the overall network.

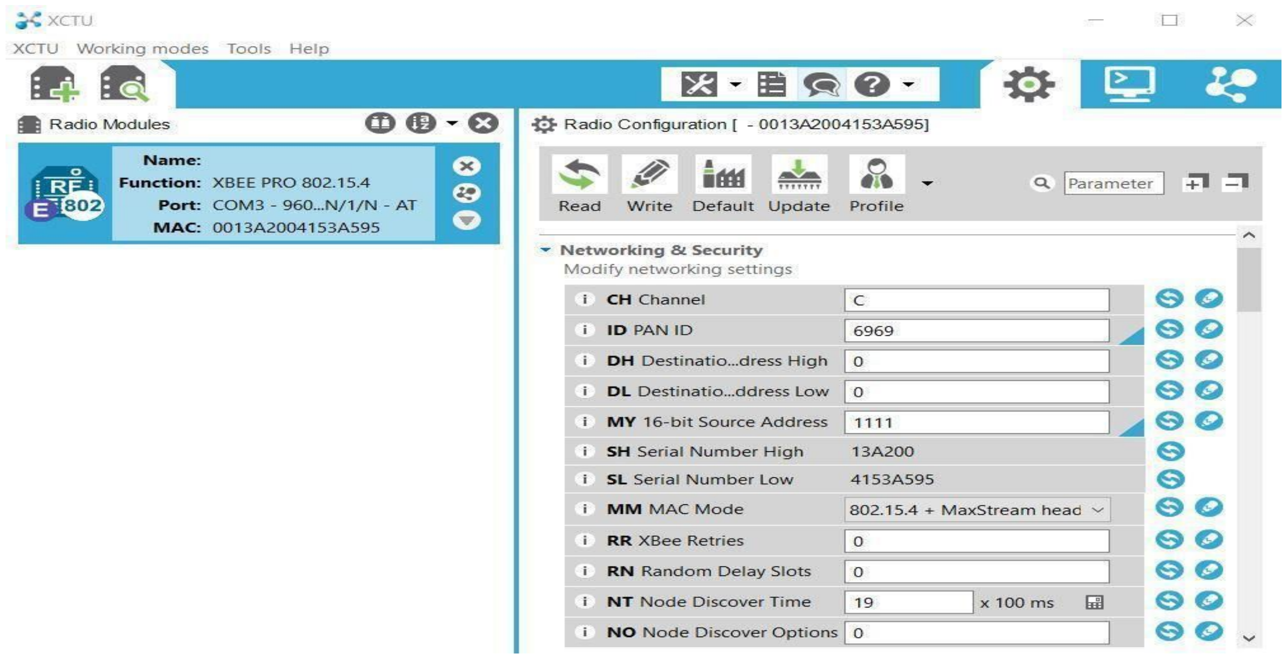
5.2.1 Arduino IDE:

The Arduino IDE is where most of the software development is taking place. The IDE is an open-source software. It allows the team to easily write and upload code directly to the board via a USB connection. The IDE runs on Windows, Mac OS X, and Linux. The code developed in the IDE is written in a set of C and C++, as seen in Figure 8 which is a portion of the Optical Dust Sensor code. It can be seen sending information through the XBee module is done through serial prints.

Figure 8: Arduino Code for Optical Dust Sensor

5.2.2 XCT-U:

XCT-U is the XBee software used in the project. This software allows the team to configure the XBee's to the same network and configure them as nodes or routers. XCT-U is available on Windows, Mac OS X, and Linux. Figure 9 shows an example of the XCT-U software. In this figure, it shows the configuration menu on the right side of the picture. This is where the team can establish the network and settings for each XBee. On the left side of the picture is where we can discover XBees that are connected to the computer.


XCTU
XCTU Working modes Tools Help
Radio Modules
Radio Configuration [- 0013A2004153A595]
Read Write Default Update Profile
Parameter
Networking & Security
Modify networking settings

Name:
Function: XBEE PRO 802.15.4
Port: COM3 - 960...N/1/N - AT
MAC: 0013A2004153A595

CH Channel	C
ID PAN ID	6969
DH Destinatio...dress High	0
DL Destinatio...ddress Low	0
MY 16-bit Source Address	1111
SH Serial Number High	13A200
SL Serial Number Low	4153A595
MM MAC Mode	802.15.4 + MaxStream head
RR XBee Retries	0
RN Random Delay Slots	0
NT Node Discover Time	19 x 100 ms
NO Node Discover Options	0

Figure 9: XCT-U software

5.2.3 Raspberry Pi 2 - Python

The Raspberry Pi 2 also comes preloaded with a python development environment to create runnable programs. Python is another easy to use programming language and is quickly beginning to rival Java as the best introductory programming language. What is crucial to the project is that Python has libraries that allow for quick and easy access to databases.

6. Implementation

6.1 Overview and Testing

- Configured Xbee's together and created a network
    - Started by connecting two Xbee's to the same network using the XCT-U software

    - Tested the connection using the serial terminal in XCT-U
    - Added a third Xbee to the network and tested the sending and receiving with more than one sender
    - Used the Xbee shields to connect Xbees to the Arduino Unos
- Integrated the Xbees with the Arduino Uno Rev3
- Tested interaction between Xbee and Arduino
    - Used the XCT-U send serial information to the Arduino with the Xbee
    - Tested interaction between the two without XCT-U
        - Sent information using the Arduino Serial
- Worked with sensors
    - Connected a sensor to an Arduino and tested the interactions
    - Wrote a simple sketch that reads in the sensor value
    - Sent sensor data collected to the home node via Xbee
        - Wrote a sender sketch that uses serial to send the value collected from the sensor
        - Wrote a receiver sketch that collects the information sent through serial
- Tested sleep function on sender nodes (to limit battery usage)
- Sleep was enabled on the router node using ISR
- Added router nodes to increase signal strength
- Found suitable weather resistant protection for sender nodes
- Wrote sketches that uploaded data collected from sensor nodes to the RPi2
- Used the RPi2 to upload data to the server database
- Found a power source
- Tested the final product
- Constructed nodes in weather resistant protection with the power sources for outdoor placement

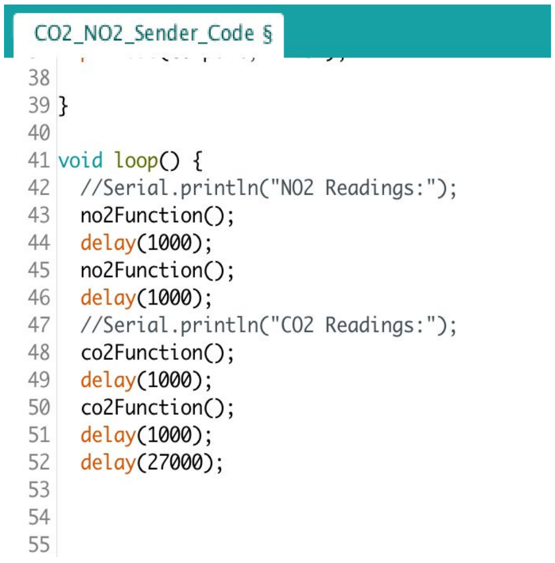
6.2 System Components and Tests

6.2.1 XBee Testing

The first phase of the implementation was to test the functionality of the XBee Pro S1 modules. Using the XCTU software, two XBee modules were configured with the same PAN ID. The serial monitor was opened for both modules and a text was written to a chosen console. On the other console this text would appear as the XBee module sent it through the network connection. The next round of testing was to see the networking capabilities of the XBee modules. This was done by adding a third XBee module into the network by using the same PAN ID. The Destination Low (DL) of two of the modules was set to the ID of the third module. This established two sender nodes and a receiver node. It was observed that the receiver module was receiving the data from both sender modules.

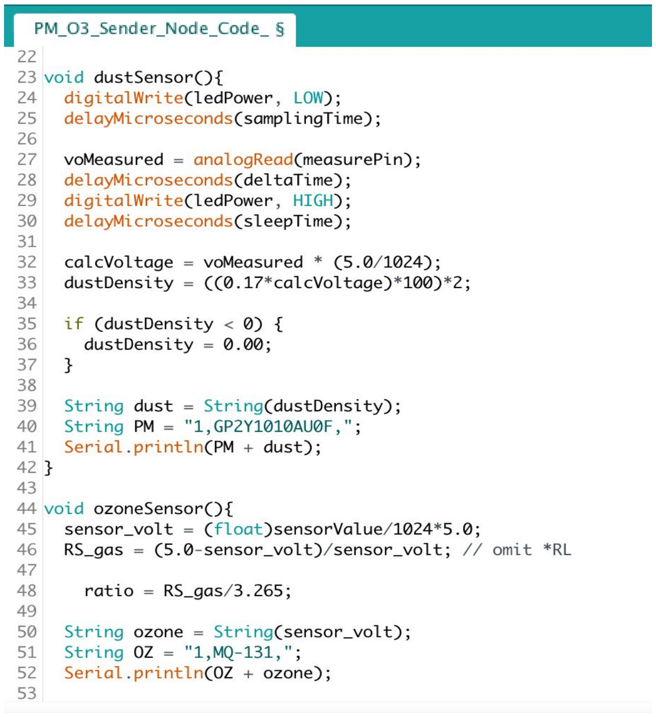
6.2.2 Arduino and Network Testing

The second phase of the implementation was to test the functionality of the XBee modules with the Arduino boards. The XBees were attached to the Arduino boards through the Arduino XBee Shields. To start, the network consisted only of two nodes and the functionalities of the XBee modules were tested to ensure the shields were giving the modules proper power. After it was found that the modules were still functioning correctly, one Arduino was uploaded with a simple code that alternatingly sent ‘H’ and ‘L’ to be the receiver node. The serial console of the receiving node was observed to ensure that the sender node was properly sending the data. A problem was discovered in that in order to upload a program to the Arduino board, the XBee shield had to be in USB mode and the XBee module had to be removed from the shield. Once in operation the XBee was returned to the shield and the shield mode was set back to UART or XBEE. After this problem was discovered and corrected, the network was working properly. The next step was to make the receiver node wireless. A code was developed so the receiver node could turn on and off an LED. The receiver would set pin 13 on the board to high, lighting the LED, when it received ‘H’ and would set pin 13 to low, turning off the LED, when it received ‘L’. This would visually confirm that the codes for both nodes were working properly. Both nodes were given power and the LED began to turn off and on. With that success, a third node was added into the network. Each sender node now had an LED to indicate which was sending to the receiving node. An Arduino code was developed for two sender nodes so that one node would send ‘H’ and ‘L’ for five seconds then wait for a duration of five seconds. During this time, sender node 2 would begin sending ‘H’ and ‘L’. This final test showed that multiple sender nodes could send data to the receiving node remotely and without help from an outside source such as a laptop and console. Figure 10 is a sample of the code from the CO2 and NO2 sensor node. Figure 11 is a sample of the code from the PM and O3 sensor node.



```

}

void loop() {
  //Serial.println("NO2 Readings:");
  no2Function();
  delay(1000);
  no2Function();
  delay(1000);
  //Serial.println("CO2 Readings:");
  co2Function();
  delay(1000);
  co2Function();
  delay(1000);
  delay(27000);



```

Figure 10: CO2 and NO2 sample code



```

void dustSensor(){
  digitalWrite(ledPower, LOW);
  delayMicroseconds(samplingTime);

  voMeasured = analogRead(measurePin);
  delayMicroseconds(deltaTime);
  digitalWrite(ledPower, HIGH);
  delayMicroseconds(sleepTime);

  calcVoltage = voMeasured * (5.0/1024);
  dustDensity = ((0.17*calcVoltage)*100)*2;

  if (dustDensity < 0) {
    dustDensity = 0.00;
  }

  String dust = String(dustDensity);
  String PM = "1,GP2Y1010AU0F,";
  Serial.println(PM + dust);
}

void ozoneSensor(){
  sensor_volt = (float)sensorValue/1024*5.0;
  RS_gas = (5.0-sensor_volt)/sensor_volt; // omit *RL

    ratio = RS_gas/3.265;

  String ozone = String(sensor_volt);
  String OZ = "1,MQ-131,";
  Serial.println(OZ + ozone);

```

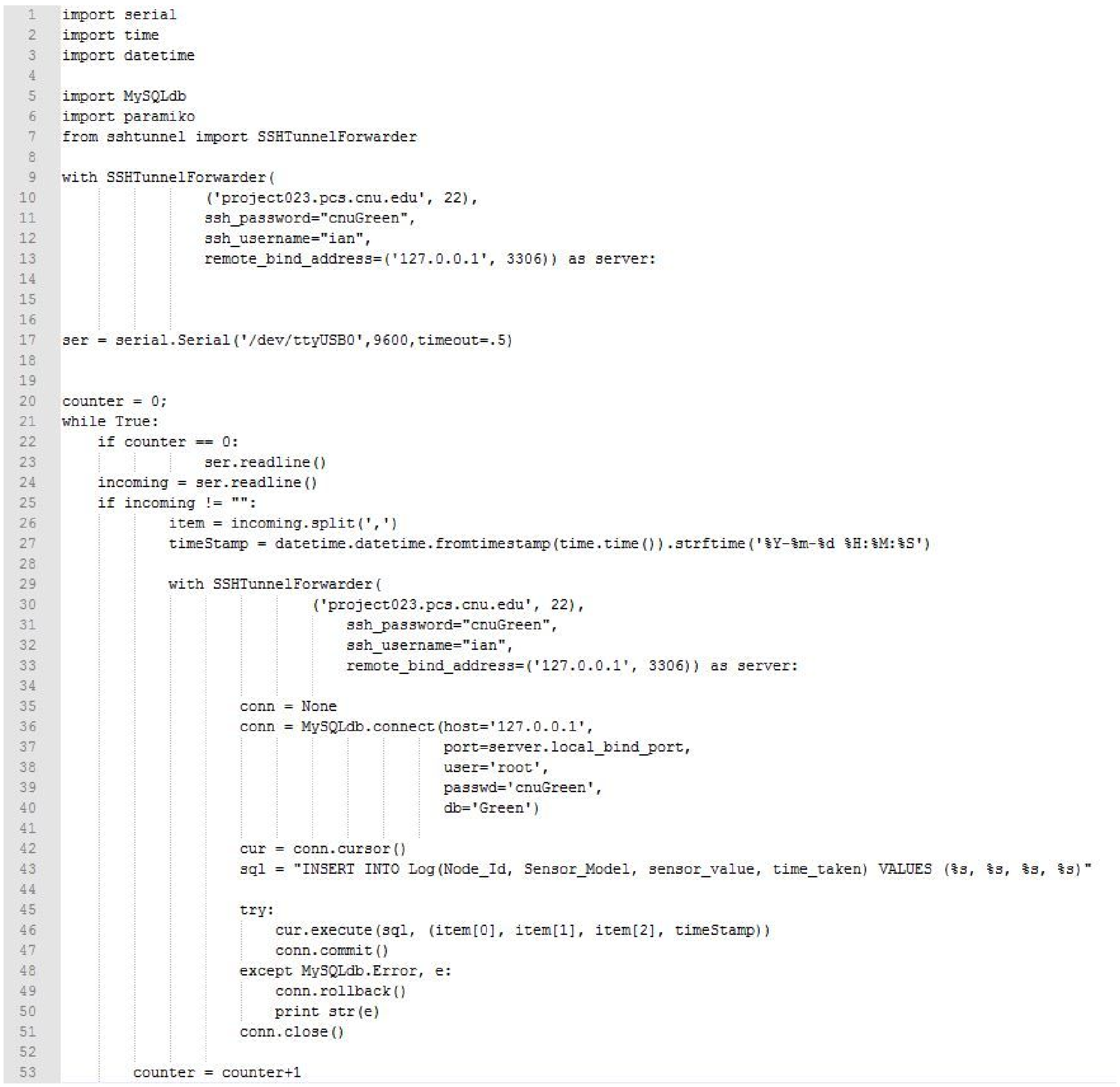
Figure 11: O3 and PM sample code

6.2.3 Sensor Testing

A very simple network was established and data was being properly sent and received. This moved to the next phase of the project which was testing the sensors. One setback was that the parts ordered for the project took longer than expected. Work was continued with the Sunfounder Arduino sensor kit. This includes similar sensors to the ones purchased, such as the combustible gas sensor, light sensor, and others. These various sensors were individually attached to a sender node and the data collected was viewed on the serial console of the receiving node. After observations were made for each sensor, the threshold factors were determined so that when these factors or data values appear, an LED light could be turned on. In these observations, it was noted that the data values coming into the serial console contained non-ASCII characters. Moving forward, the receiver node was made wireless again and the LED was attached. An LED was also attached to the sensor node to ensure the sensor was outputting the correct data. When both nodes were activated, the sensor node LED was activating when the sensor was detecting the values it was supposed to trigger on.

6.2.4 Raspberry Pi 2 Configuration

The Raspberry Pi 2 was configured with the New Out Of the Box operating system (NOOBS OS). This is a basic, easy to use operating system that is a derivative of Raspian which is a Linux based system. After the operating system was installed, the settings were adjusted to fit the functionality that was needed for the project.The Raspberry Pi runs a program written in Python to take the data it receives and upload it to the database as seen in Figure 12. The python program is run automatically on boot up. The program establishes a Secure Shell (SSH) tunnel to the server where is can then access the database when needed. The port is then established for which the serial data will be delivered by the XBee module. The while loop is started and if data is already present, the program skips it if the counter is zero as some of the data needed may have already been lost. The program then continues to run and if data is present from the serial port, it establishes a connection to the database and inserts the data. If the data does not match the proper format for the database, then an error is thrown and caught which causes the database to roll back and the program continues to run. Otherwise the program will go back into the loop and repeat the process.



```
import serial
import time
import datetime

import MySQLdb
import paramiko
from sshtunnel import SSHTunnelForwarder

with SSHTunnelForwarder(
                ('project023.pcs.cnu.edu', 22),
                ssh_password="cnuGreen",
                ssh_username="ian",
                remote_bind_address=('127.0.0.1', 3306)) as server:



ser = serial.Serial('/dev/ttyUSB0',9600,timeout=.5)


counter = 0;
while True:
    if counter == 0:
                ser.readline()
    incoming = ser.readline()
    if incoming != "":
            item = incoming.split(',')
            timeStamp = datetime.datetime.fromtimestamp(time.time()).strftime('%Y-%m-%d %H:%M:%S')

            with SSHTunnelForwarder(
                            ('project023.pcs.cnu.edu', 22),
                                ssh_password="cnuGreen",
                                ssh_username="ian",
                                remote_bind_address=('127.0.0.1', 3306)) as server:

                    conn = None
                    conn = MySQLdb.connect(host='127.0.0.1',
                                           port=server.local_bind_port,
                                           user='root',
                                           passwd='cnuGreen',
                                           db='Green')

                    cur = conn.cursor()
                    sql = "INSERT INTO Log(Node_Id, Sensor_Model, sensor_value, time_taken) VALUES (%s, %s, %s, %s)"

                    try:
                        cur.execute(sql, (item[0], item[1], item[2], timeStamp))
                        conn.commit()
                    except MySQLdb.Error, e:
                        conn.rollback()
                        print str(e)
                    conn.close()

        counter = counter+1
```

Figure 12: Code for Raspberry Pi 2 to upload to database

6.2.5 Network Construction

After all of the components had been configured and tested, the network was constructed. First it was constructed with one node that contained two sensors and the Raspberry Pi receiver. Data from both sensors was collected and transmitted to the receiver. The Raspberry Pi received the data and displayed it on its terminal. After the success of the basic single node to receiver network, a relay node was added. This was tested again and the data was successfully transmitted to the receiver terminal. The next functionality of the network to test was the addition of a second node. To ensure that the data would be uncompromised, a simple code was written to have simultaneous transmissions to the receiver. Upon receiving both transmissions, the data was displayed properly on the Raspberry Pi terminal. Sensors were then attached to the second node.

6.2.6 Database

A Ubuntu 16.04 server was established to host the database and user interface. A MySQL database was installed and configured on the server. The database has tables to store information on nodes, sensors, data logged, administrative user's login information, and administrative user generated reports. Figure 13 demonstrates the relationships of the tables in the database.

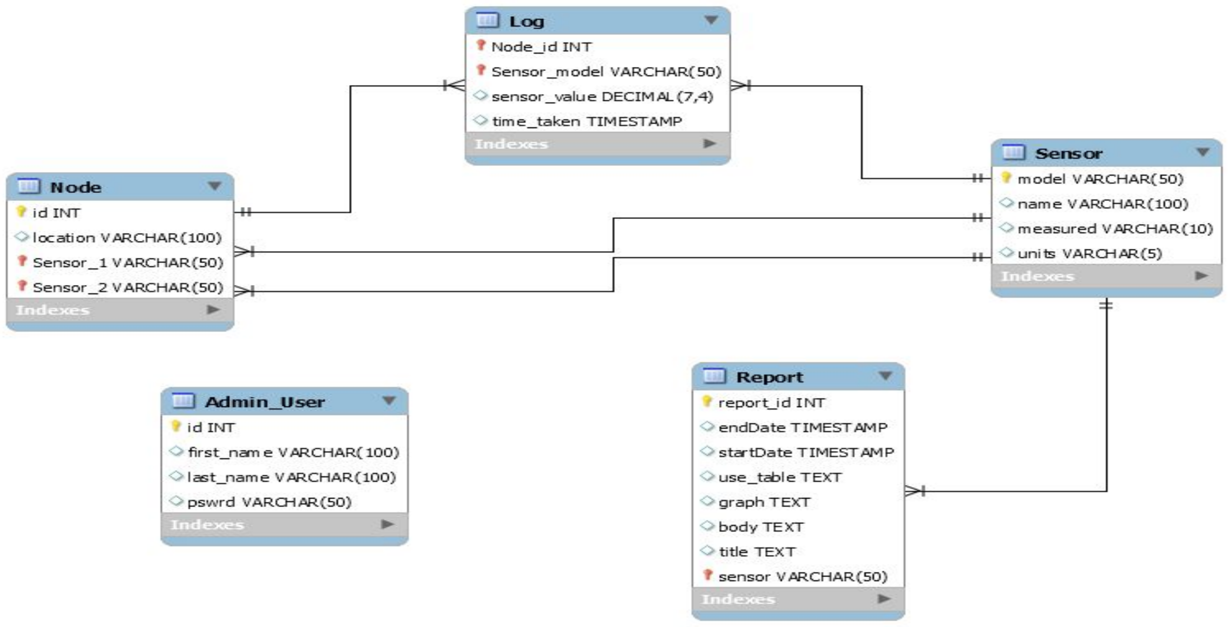


Figure 13: Entity Relationship Diagram for Database

6.2.7 Interface

The server was configured with Apache Tomcat8 so that the Java Servlet Pages (jsp) could be rendered. The front end user view of the interface was created with HTML and CSS. On the backend, Java Web Servlets are used to make calls to the database to pull needed data and manipulate it as needed. The linking between the back end and front end is done with Javascript and JQuery/Ajax calls. User input is pulled from fields on the front end and passed to the proper Java Web Servlet. The web servlet returns Json objects to the JQuery/Ajax call for further processing. This is where the graphs, charts, and tables are created before they are injected in the DOM on the front end. Figure 14 shows the user interface that all users will be able to see showing the current air quality data gathered from the nodes plotted out in graphs. Authorized users can log in to the admin portion of the user interface where they can view the current data and also create tables, graphs, and reports to be put on the public side of the interface or downloaded for other uses. Figure 15 shows the report, graph, and table creation page of the interface.

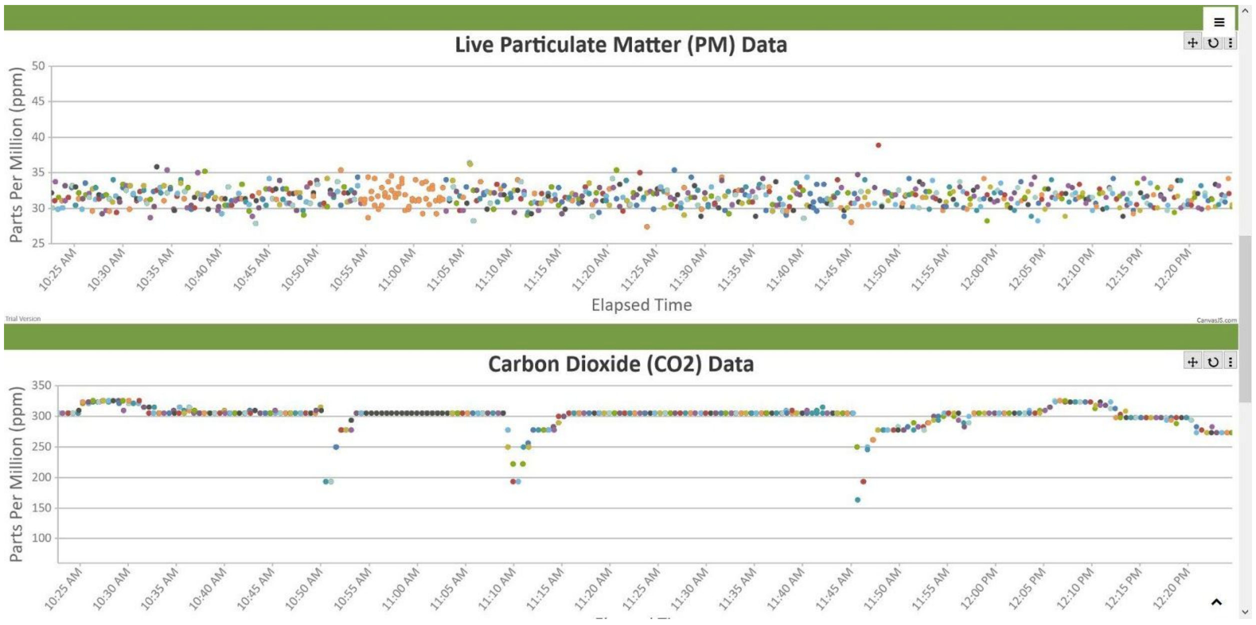
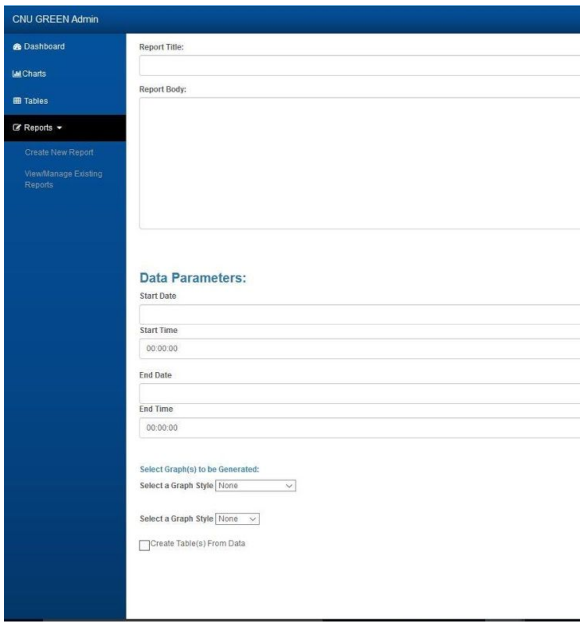


Figure 14: Current Data View Page of Interface



Report Title:

Report Body:

Data Parameters:

Start Date

Start Time
00:00:00

End Date

End Time
00:00:00

Select Graph(s) to be Generated:

Select a Graph Style None

Select a Graph Style None

☐ Create Table(s) From Data

Figure 15: Report Generate Page of User Interface

6.2.8 Final Product Testing

Once our database and interface were both set up, we were able to begin testing our final product. Firstly, we began by testing our network indoors so we could double check that everything was running smoothly. After we confirmed the network was working properly we took the router and sensor nodes outside and set up the home node in the Game Room located in Luter Hall. We placed the router node so that it would have the best line of sight corresponding to the home node. We found that in order for the router node to keep a stable signal to the home node, it need to be placed in the brush in front of Warwick River Hall, on University Place. From there we began to branch off the two sender nodes. We placed one node by the Freeman Center and the other node by the Greek Mansions. Overall, this network covered approximately 800ft from the home node to each sensor node.

6.2.9 Cost and Safety

Once was the final product was assembled, the total spent in parts, for the project was $600. This high cost cost was due to the multiple microprocessors and XBee modules needed to assemble the network in our original requirements for a final product. If our updated requirements were the original ones, the team would not have had to send as much money on the XBee modules and shields, lowering the total cost significantly. The limited availability of sensors also contributed to the high price. If the team was granted more money, we would replace the current sensors on the final product with higher quality, more accurate sensors. We would also like to increase the size of our network by creating more sensor and router nodes to spread across the campus. If the budget was smaller in the beginning of the design process, the team would have have opted for a smaller network that could access CNU's local network. Each sensor node was $175, the router node was $100, and the home node was $100. The extra $50 on our total expenses comes from the XBee explorer boards, which allowed us to configure the Xbee modules, the tupperware containments, and the silicon gel packets.

Safety was a huge concern for the team. We needed to make sure that a remote node that remained unattended for multiple hours, even days, would keep the surrounding environment safe from fire. To do this, all of the power banks attached to the nodes have a "smart" technology that only draw as much current and voltage as the nodes need. This "smart" technology prevents excessive amount of current being put into the nodes, preventing a possibility of a fire to start.

6.3 Engineering Standards

6.3.1 XBee ZigBee Standard

The ZigBee standard operates on the IEEE 802.15.4 physical radio specification and operates in unlicensed bands including 2.4 GHz, 900 MHz and 868 MHz [11]. The XBee Series 1 Pro modules used in this project ran on 2.4GHz frequency. This frequency was used across all the nodes in the network so communication could be completed successfully. Using these 2.4 GHz RF modules along with the XCT-U software, the XBee's were setup in a manner in that would provide a safe, secure and fast transmission link amongst the nodes. The ZigBee protocol was designed to provide an easy-to-use wireless data solution characterized by secure, reliable wireless network architectures [11].

6.3.2 Arduino Standard (C Standard)

The standard used to maintain the legitimacy of the Arduino scripts written was the C11, ISO/IEC 9899:2011 standard which is the current, most up-to-date standard used in the C language environment. This ensured the Arduino scripts were written with the proper specifications and functionality as given in the API.

6.3.3 Java Coding Standard

The Java Coding standard needed to be used as a guideline when developing the backend data processing pages for the user interface. This allowed for the development of terse, readable code that gave the proper functionality desired to the interface.

6.3.4 Linux Standard Base

The Linux standard base had to be followed when setting up the raspberry pi 2 as well as the Ubuntu server. This standard layouts a common setup and organizational structure for all Linux based applications and devices that use Linux.

6.3.5 Python Coding Standard

The Python coding standard outlined procedures for Secure Shell into the Linux server and JDBC into the database. This allowed for quick and easy access to the server and database with the ability to prevent and catch an errors encountered.

6.3.6 MySQL Standard

The standards that needed to be followed for MySQL helped in creating tables in the database to keep data within the required constraints and keep with referential integrity.

7. Final Product

For our final product, we produced a network with two sensor nodes, one router node, and one coordinator/home node. The two sensor nodes and the router node uses the Arduino Uno Rev3 as a microcontroller. One of the sensor nodes contains the MG-811 ($CO2$) and MiCS-2714 ($NO2$) gas sensors and the other node contains the MQ-131 ($O3$) and the GP2Y1010AU0F (PM) sensors. Each of these nodes use the Xbee Series 1 Pro to send data readings wirelessly via serial to the router node. Once sent to the router node, the information gathered is then relayed, using the Series 1 Xbee, to the coordinator node. This node is operated by the Raspberry Pi 2 microcontroller. This node gathers the data sent from the router node and uploads it to our database server. From there the data is represented in the form of graphs. Figure 16 shows an image of the final product with the nodes in their weather resistant containment.

Figure 16: Image of Final Product

7.1 Meeting Requirements

Looking at our final product and the requirements given to us, our product succeeds by meeting just about every one. The final product is a working ambient air quality network that can be deployed across CNU’s campus to provide accurate measurements of air to the sustainability coordinator, Ryan Kmetz. The network has four nodes, two sensor nodes, one router node, and one home node. The data being collected continues to send until the battery powering the node is dead. The data has the ability to be viewed in real time via the remote database. As stated before, real time is the current values in the database at the time of when the user logs on or refreshes the page. It could have a slight delay due to the delay programed into the sensor nodes. Once the data arrives to the home node, it is sent straight to our database server. From there the sensor data is displayed in readable graphs that the user can manipulate. The backend of our server is only accessible to authorized user. These authorized users can create reports, graphs, and tables.

Due to budget cuts, we were unable to to fulfill our original requirements of five nodes; four sensor nodes and one home node. Also, we were unable to meet our original requirements for our battery life. Originally, we wanted out batteries to supply our nodes with 3-4 weeks of power. Unfortunately, due to the high cost of these batteries, we were forced to settle with an alternative that only lasted 65 hours.

8. Outlook/Future Implementation

Future tasks that could be completed for the project would include better power sources for the sensor nodes, better weather containment devices and possibly stronger detecting gas sensors. This could all be done with a higher budget in mind as the better the gas sensor, the more expensive the price. Another future advancement in the project could be to enhance the type of XBee modules being used or even considering WiFi as a communication method. With the changes made to the project in February, WiFi potentially could have suited our project needs but with already purchasing the XBee and XBee shields, we continued on as planned. The last thing that could be done in the future to enhance the project would be the database server. Right now the server receives the data from the coordinator node and adds it to the database which stores all received data. That data is used to create graphs to aid in visual representations of the data but maybe a spreadsheet type format could be created from the data within the database and made printable for viewing pleasure.

9. Conclusion and Teamwork:

Overall, the project was a great learning experience. The project was challenging and vigorous and tested the team in ways they have not been tested before in terms of hardware, software and networking applications. The team learned so much in the previous academic year regarding microprocessors, software engineering and networking. No team member had any previous knowledge or educational training in most areas of the project. This project helped round out the learning experiences of the team gained here at CNU.

Things the team would have done differently:

- Not spend so much time trying to fix minor issues
- Focusing more on power sources and rendering sleep mode for sensor nodes
- Making the graphs more visually enticing on the user interface
- Asking more professors/advisors for help resolving issues faster
- Finding weather proof containment for the nodes

As for teamwork, the team found there were little to no problems at all. The team was able to meet consistently throughout the year and everyone made reasonable contributions to the project. Each team member handled their responsibility as expected and performed to the highest level at all times.

Bibliography

[1] World Health Organization. "Ambient Air Pollution: A Global Assessment of Exposure and Burden of Disease." *World Health Organization*. WHO Document Production Services, n.d. Web. 1 Dec. 2016. http://apps.who.int/iris/bitstream/10665/250141/1/9789241511353-eng.pdf?ua.

[2] "WHAT IS ARDUINO?" *Arduino - Home*. N.p., n.d. Web. 02 Dec. 2016.

[3] @Raspberry_Pi. "Raspberry Pi 2 Model B." Raspberry Pi. N.p., n.d. Web. 06 Dec. 2016.

[4] Sandbox Electronics. N.p., n.d. Web. 02 Dec. 2016.

[5] "SainSmart MQ131 Gas Sensor Ozone Module For Arduino UNO Mega2560 R3 Raspberry Pi." *SainSmart MQ131 Gas Sensor Ozone Module For Arduino UNO Mega2560 R3 Raspberry Pi 3D Printing, Arduino, Robotics | Sainsmart*. N.p., n.d. Web. 02 Dec. 2016.

[6] #599397, Member, and Member #401984. "Optical Dust Sensor - GP2Y1010AU0F." COM-09689 - SparkFun Electronics. N.p., n.d. Web. 06 Dec. 2016.

[7] "MICS-2714." *MICS-2714, Gas Sensors, SGX Sensortech Limited (formerly E2v)*. N.p., n.d. Web. 02 Dec. 2016.

[8] "Bluetooth Basics." Bluetooth Basics. Spark Fun, n.d. Web. 06 Dec. 2016. https://learn.sparkfun.com/tutorials/bluetooth-basics.

[9] #591147, Member, Member #619622, Member #127179, Member #855741, Member #1373, Member #170380, Budhabar, and Member #724783. "XBee Pro 60mW Wire Antenna - Series 1 (802.15.4)." *WRL-08742 - SparkFun Electronics*. N.p., n.d. Web. 02 Dec. 2016.

[10] "List of Digests: NO2, O3, & PM." GreenFacts - List of Digests. WHO Document Production Services, n.d. Web. 06 Dec. 2016. http://www.greenfacts.org/en/digests/index.htm.

[11] "ZigBee® Wireless Standard." ZigBee® Wireless Standard - Digi International. Digi International, n.d. Web. 24 Apr. 2017. https://www.digi.com/resources/standards-and-technologies/rfmodems/zigbee-wireless-standard.