An Internet of Things Drone Data Mule

Drew Cochrane, Nolan Evans, Michael Lane, Grant Woodbury, Xi Yin, Corey Zrobek, Marcia Friesen, Ken Ferens
Dept. of Electrical and Computer Engineering, University of Manitoba, Winnipeg, MB, Canada
Corresponding Contact: Ken.Ferens@umanitoba.ca

Abstract—This paper is the fourth in an annual series that reports on one of the approaches that an Electrical and Computer Engineering department at a Canadian University takes to improve, maintain, and ensure their accreditation in undergraduate programs. In particular, this paper reports on a capstone design project that gives computer engineering students the opportunity to demonstrate the skills (aka, attributes) that they have attained throughout their studies by designing, implementing and testing a solution to a medium complexity Engineering design problem. The skills these particular students demonstrated were the attributes of Knowledge, Design, Problem Analysis, Investigation, Teamwork, Communication, and Impact on Society, which are part of the Canadian Engineering Accreditation Board’s 12 Graduate Attributes.

Keywords—Capstone design project, accreditation, CEAB graduate attributes, Drones, Data Mule, Delay Tolerant Networks, Internet of Things, data networking, Internet of Things Architecture.

1. Introduction

The Department of Electrical and Computer Engineering (ECE) at the University of Manitoba, Canada, offers a “capstone” design course, ECE 4600 Group Design Project, which is normally taken by students in the final year of their undergraduate ECE program. The project work constitutes a significant design experience based on the knowledge and skills acquired throughout their undergraduate program and gives students an exposure to the concepts of engineering design in a team environment. Students are required to demonstrate, within a fixed time period (7 months, two terms), the ability to conceive a design of a solution to a medium size engineering problem, and to organize, conduct, test, and report on it. The requirements consists of five parts: a project proposal, the engineering log-book, written and oral progress reports, a formal engineering report, and a public oral presentation.

This paper is organized as follows: First, a brief overview is given on a chosen design project, followed by related work in the application of drone data mules in the Internet of Things (IoT) research and industry fields. Next, in depth description of the different subsystems, including the ground node, drone node, and server node is given. This is followed by a description of the unit and system testing experiments and analysis conducted to verify the system requirements. Finally, conclusions and recommendations for future work are given.

Overview of the Drone Data Mule IoT Device

The following is a brief overview of the drone data mule design project done by undergraduate students at the Department of ECE, U of M, 2019. This design project uses a drone to gather data from remote sensors and send the data up to a cloud for further processing and storage.

The primary goal of this project was to design a drone, which computably acted as a data mule, gathering data from remote regions and transferring to areas with established internet infrastructure. The motivation for this project comes from the desire to reduce risk, decrease expense, and increase accessibility to environmental or customer preferred data. The project is intended for regions that are hard to access with no internet connectivity and for data which is delay tolerant. Retrieval of such data tends to have high costs and places workers in unsafe situations.

This project was designed as a proof of concept and relies heavily on full system integration between the ground node (environmental sensor), the drone node (data mule), and the server node (internet connected infrastructure). The primary solution is a Printed Circuit Board (PCB) integrated onto an Unmanned Aerial Vehicle (Drone) which is capable of autonomously and wirelessly exchanging data. The PCB solution consists of a microcontroller unit, Bluetooth BLE module, SD card, and power systems. The PCB software was designed to interface and exchange information with both the ground and the server nodes. As a proof-of-concept, the ground node was developed to record, store, and transmit temperature data. The server node was developed to receive, and store data in a cloud based web server and database. This solution was developed at the University of...
2. Related Work

The Internet of Things (IoT) is the inter-networking of heterogeneous physical devices which rely on while also supporting networks of data-gathering sensors and cloud computing. IoT applications typically include a network of sensors, a data collection component or function, and a control or feedback component [1]. In the current work outlined in this paper, the IoT is envisioned as a distributed network of sensors collecting information (for example, water quality sensors on lakes; sensors on city infrastructure that may collect temperature or noise or emissions data). The drone, acting as a data mule, regularly travels past each sensor and collects (uploads) the data from that sensor via a Bluetooth connection between the drone and the sensor, and the drone stores these data in its on-board memory. Once the drone has completed its tour of visiting all sensor nodes, it travels back to a base station where the data are offloaded to Internet infrastructure for analysis.

An increasingly familiar arena of a growing range of IoT applications is the so-called Smarthome realm. This includes applications such as a refrigerator maintaining an inventory of contents, remote monitoring and control of home entry doors and garage doors, systems that detect and activate home lighting, and personal assistants that control electronic devices such as Google Home and Amazon Alexa. This scope of IoT applications continues to develop rapidly.

In industrial sectors, IoT is often part of a strategy to gain operational and financial efficiencies, and those efficiencies are gained through analysis of data captured through IoT applications. Two fields that are prime targets for IoT applications are transportation and environmental monitoring. In the transportation field, IoT applications vary from fleet management (e.g. vehicle location, optimized trip trajectories and multi-node tours, geo-fencing), complementary way-finding technology (signage, hazards alert, etc.) and fleet analytics.

IoT applications in environmental monitoring include but are not limited to air and water quality monitoring. A common example is monitoring surface water quality over time and area. One can envision the application described in this paper as highly suited to gathering surface water quality information from a sensor network in remote locations difficult to access efficiently by other means, for example, in a series of remote lakes. IoT applications in

3. System Architecture and Design

Ground Node Hardware

The purpose of the ground node is to act as a remote temperature sensor, storing data over time to be collected by the drone node. The ground node collects temperature measurements every second, storing these on an SD card. When a BLE connection is formed with the drone node, all temperature measurements collected on the SD card are transmitted wirelessly to the drone node. As shown in Fig. 1, the ground node consist of a microcontroller, temperature sensor, Bluetooth module, and SD card module.

The Arduino Uno was chosen to be used as the microcontroller for a variety of reasons. First, the Arduino Uno supports Inter-Integrated Circuit I2C and Serial Peripheral Interface (SPI) communication protocols. These protocols are needed for interfacing with peripheral devices to build the ground node [8]. Secondly, there is a large open source software community supporting the Arduino Uno. Each respective ground node device purchased, such as the TMP102 temperature sensor breakout board, already had wiring schematics and example software for use with the Arduino Uno. Open source support was integral given that the main focus of the project was the drone node; any time saved during ground node development was essential to project success.
To collect temperature sensor measurements, the TMP102 temperature sensor was chosen. The SparkFun Digital Temperature Sensor Breakout - TMP102 module was purchased due to the small size of the sensor, to alleviate soldering difficulties. The TMP102 breakout board has built in 4.7 k pull-up resistors to facilitate the I2C serial communication supported by the TMP102 temperature sensor. As well, SparkFun provides open source driver software to facilitate I2C communications with the breakout board [9]. This open source software facilitates changing temperature units, the bit accuracy of measurements, and the frequency of distinct temperature measurements on the TMP102 sensor.

After a temperature measurement was collected by the TMP102 temperature sensor, the measurement needed to be stored. The Adafruit Micro SD breakout board was chosen to store collected temperature measurements. The newest Micro SD cards are edge triggered and require very square transitions during SPI communication. Further, SD cards require a strict 3.3V supplied to the card; it is recommended level shifters are used to accomplish this strict voltage level [10]. Thus, after preliminary research it made sense to purchase an SD card circuit that was known to work, rather than building a circuit from scratch. The Adafruit Micro SD breakout board met the above requirements of having square transitions during serial communication, as it used a Texas Instruments CD74HC4050 level shifter that converted all interface logic to 3.3V [10]. The manual provided with the Adafruit Micro SD Breakout Board also describes in detail how to wire the breakout board to an Arduino Uno. As well, Adafruit provides details on how to use the built in SD card libraries supported by the open source Arduino Community [10]. Choosing a board that supports the use of open source SD card libraries was essential given the complexity associated with SD card interfacing.

Temperature measurements stored over time on an SD card is needed to be transmitted wirelessly to the drone node. BLE was chosen early in the project as the wireless protocol to be used to transmit temperature data from the ground node to the drone node. The original plan was to use the same BLE module in both the ground node and drone node. However, issues were encountered when originally attempting to interface the BM71 BLE module to the drone node; it was unclear whether the BM71 BLE module was going to work for the drone node. Thus, to facilitate continued development of the ground node, the Adafruit Bluefruit LE SPI Friend was chosen as the BLE module for the ground node. The Adafruit BLE module uses SPI communication, this was already proven to work with the Adafruit SD card module at this point in the project. Further, one of the lessons learned from the BM71 module was that extensive documentation and examples on how to use a BLE module saved time during development. The Adafruit BLE module has an extensive manual describing its use, as well as countless example software projects compatible with the Arduino Uno. Further, the module had an accompanying Android Application for testing the aforementioned example projects. As well, Adafruit provides driver software for interacting with the Adafruit BLE module [11]. Given successes using the Adafruit SD card module at this point in the project, as well as the above documentation and examples, the Adafruit BLE module seemed like the most logical choice for the ground node.

To create a functioning ground node, the sensors described above all needed to be connected to the Arduino microcontroller board. The TMP102 temperature sensor used the I2C communication protocol; it was connected to the hardware I2C pins of the Arduino Uno. However, both the Adafruit SD card module and Adafruit BLE module used the SPI communication protocol. The Arduino Uno supports two methods of SPI communication, hardware SPI and software SPI. Hardware SPI uses a dedicated SPI circuit to facilitate SPI communication, whereas software SPI manipulates general purpose input/output pins to enable SPI communication. Given the overhead of the SPI protocol using purely software, as well as greater inaccuracy of General Purpose Input/Output (GPIO) pins, software SPI is significantly slower than hardware SPI.
Both the SD card module and BLE module on the ground node require high communication speeds in order to ooad SD card data via BLE to the drone node as fast as possible. Consequently, a shared hardware SPI bus was used to ensure the fastest transfer rate of SD card data to the BLE. Originally software SPI was used for the BLE module, however this resulted in significantly slower throughput to a peripheral when transmitting data. With both the SD card module and the BLE module sharing a hardware SPI bus, a throughput of approximately 16 Kbps was achieved to a Samsung Galaxy S8+ BLE peripheral when transmitting temperature data. However, the aforementioned throughput was not attainable when sending temperature data to the drone node, as is described in detail later in Section 3 of the report.

Software

The ground node software uses the three hardware components, described in the ground node hardware section. These components collect temperature measurements over time, and then transmit these values to the drone node when a BLE connection is formed.

First, the ground node software initializes its connections with the three peripheral components, these being the I2C connection with the TMP102 temperature sensor, and the SPI connections with the SD card module and BLE module. The BLE module is then configured with a custom BLE service, which encapsulates custom defined BLE characteristics for the transmission and reception of data. At first, the intention was to have the drone node use BLE notifications on the transparent Universal Asynchronous Receive Transmit (UART) service of the BLE module. A transparent UART service forms a data tunnel between two BLE modules, masking the manipulation of BLE characteristics in the transmission of data between the two modules. Behind each transparent UART service are a transmission and reception characteristic defined within a BLE service; these are used to form the actual data tunnel between the two BLE modules. However, in order for the transparent UART service to work between two modules, the transmission and reception characteristics must have the same Universal Unique Identifiers (UUID) on each BLE module respectively. The drone node BLE module and ground node BLE module use different characteristics for their transparent UART services.

Since transparent UART services manipulate a set of characteristics with distinct UUID’s in order to facilitate communication between BLE modules, it then made sense for the drone node BLE module to notify on the ground node transparent UART transmission BLE characteristic. The BLE server was the BLE module on the ground node, and the BLE client was the drone node BLE module. A BLE client notifies on a BLE server characteristic in order to receive its data. Every time the value of the server characteristic changes the new characteristic value is sent to the BLE client, in this case the drone node. Thus, if the drone node RN4871 BLE module notifies on the ground node transmission characteristic, it would receive all transmission data from the ground node, as well as simplify Application Programming Interface (API) calls in ground node programming by utilizing the ground node transparent UART service which hides underlying ground node characteristic manipulation.

BLE characteristics have certain permissions that specify which operations can be performed on them by a peripheral BLE module. The drone node BLE module invoked notifications on the characteristic of a server by writing 0x0100 to the peripheral characteristic it wished to notify on. However, the transparent UART service transmission characteristic of the ground node did not have the write permission defined, and the request to notify was never received. This led to the custom definition of a BLE transmission characteristic on the drone node with the write permission allowed; the drone node BLE module was able to receive data whenever the value of this characteristic was changed after writing the notification message to the ground node transmission characteristic. An associated custom BLE receive characteristic was then defined within the same BLE custom service as the transmission characteristic. The drone node writes values to the receive characteristic to send data to the ground node. While defining custom characteristics did create more work, it cannot be expected that every ground node will share the same transparent UART characteristics as the drone node BLE module, nor allow writing to their transparent UART characteristics. Having the drone node notify on a custom transmission characteristic is a more general solution to communication between the ground and drone node and allows communication between BLE modules of any type.

After the definition of custom BLE characteristics, the ground node software moves into a continuous loop. The ground node checks repeatedly whether a BLE connection has been formed by a client with the ground node BLE module. If a connection has not been formed, the ground node reads a temperature from the TMP102 temperature sensor using the I2C libraries of the Arduino Uno. This temperature value is then written via SPI to the SD card using the Adafruit SD card module using the Arduino SD libraries, preceded by the number of seconds from the start of Arduino Uno operation and delimited by a space. The
number of seconds represents a time associated with the temperature measurement. To get an actual time required the interfacing of a real time clock with the Arduino Uno, which was out of project scope. Distinct time and temperature measurement pairs were delimited by a comma. Comma delimiting was performed due to the fact that the Adafruit BLE Arduino libraries interpret newlines to mean the end of a string. A delay of one second occurs between each distinct temperature measurement.

If a BLE connection is formed with a peripheral (the drone node), the ground node waits for the drone node to write the value 0xFF to its custom defined BLE receive characteristic. The drone node must notify on the custom defined ground node transmit characteristic in order to receive the characteristic’s updated values; if the value of the ground node transmit characteristic was updated before notification was requested by the drone node this data would not be sent to the drone node. Once the code 0xEE is received from the drone node, the ground node begins reading chunks of logged temperature data from its SD card. The ground node updates the value of its transmission characteristic in 20-byte chunks (as this is the maximum size that can be written to a BLE characteristic at one time). The value of the transmit characteristic is updated 5 times, and 80 bytes of data are sent. The ground node then waits for the drone node to write 0xEE to its receive characteristic again. This signifies another 80 bytes of data can be sent to the drone node. Data must be sent in chunks due to the limited size of the drone node receive buffer. If notifications from the ground node are received too rapidly, the BLE receive buffer on the drone node overflows.

Drone Node Hardware

The function of the drone node addition to the actual drone (Fig. 2) is to wirelessly collect data from the ground node, store the collected data, and offload the stored data wirelessly to a server.

The microcontroller of the drone node facilitates the collection, storage, and offloading of temperature data through serial communication with peripherals, as well as software run on the microcontroller itself. Components of the drone node module are chosen to support or be controlled by the drone node microcontroller. The microcontroller chosen for this project is Microchip Technology’s PIC24FJ256GA702 (PIC24). This microcontroller serially interacts with the RN4871 Bluetooth module and SD card circuit of the drone node, which communicate and store temperature data respectively.

One of the most vital requirements of the drone module is that it had to be battery powered, and last for up to eight hours through the light of the drone carrying the module. While the PIC24 microcontroller is not the largest consumer of power, its low power consumption allows for a smaller battery and more configurability to be used in the overall design, aiding in meeting the weight requirement. As per the PIC24 data sheet, the maximum current output of any Vss pin is 300 mA, while the maximum output voltage is 4V. In terms of system throughput, with the 8 MHz internal oscillator being used serial communication throughput exceeds 10 Kbps. Exemplary of this is SPI communication with the SD card supported by the PIC24 with an SPI clock speed of 2 MHz. Lastly, the 16 KB of RAM on the PIC24 microcontroller met the memory requirement of storing 10 KB of temperature data, however the SD card circuit was added into the drone module to greatly enhance the amount of temperature data stored per drone light and storage on the PIC24 became irrelevant.

The acquisition of temperature data from ground nodes, as well as the offloading of collected data to the server node is done through Bluetooth BLE services. The RN4871 Bluetooth 4.2 Low Energy Module produced by Microchip Technologies was chosen to perform BLE communications with peripherals from the drone node. The RN4871 module uses UART for serial communication with the PIC24 microcontroller. As described in the RN4871 manual, an ASCII command API is used to configure different RN4871 settings as well as different modes of operation by sending different text strings from the PIC24 to the RN4871 serially. The RN4871 has two roles in the drone node implementation; it must retrieve data from a ground node acting as a BLE client, or send data to the server node as a BLE server. Given the RN4871 can switch between BLE client mode and server mode through the reception of ASCII commands, or through a hardware pin control, it was a valid choice for communication with the ground and server nodes.
The RN4871 was chosen for its small form factor, built-in ceramic chip antenna, support of UART, and interfaces at an operating range of 1.9 V to 3.6 V. A small form-factor is important to the overall design of the drone node because of strict weight and space restrictions, and the presence of a integrated antenna was necessary because of the challenging nature of antenna design.

The RN4871 supports two different roles, BLE server mode and BLE client mode. Both roles are utilized by the RN4871 during flight. The RN4871 goes into BLE client mode to connect to the ground node, which acts as a BLE server, described in detail in section 2 of the report. Being in BLE client mode allows the RN4871 to connect to multiple ground nodes sequentially if their MAC addresses are known. Data is received from ground node BLE servers by notifying on their BLE transmission characteristics, and data is sent to ground nodes by writing to their receive characteristics. The transparent UART service could not be used for this portion of the project, as the Adafruit BLE module of the ground node does not have the requisite properties to allow notification from the drone node. Conversely, the RN4871 acts as a BLE server when connecting to the server node (which acts as a BLE client). The server node accesses data from the drone node by notifying on its transparent UART transmit characteristic. Data is then received from the server node by the drone node by reading from its transparent UART receive characteristic. Having both BLE client and server mode allows the drone node to follow two distinct programming paths, which reduces redundancy in the code and alleviates the necessity of checking which device the drone node is connected to, reducing the size of the state machine.

A method for storing temperature data received from the ground node needed to be put in place, as the PIC24 has limited on-board memory. While the PIC24 did have sufficient RAM to meet the memory requirements in the project proposal, more temperature measurement storage on the drone node would make the drone node more applicable in the real world. More than 10 KB of temperature data is likely to be collected by ground nodes between subsequent visits of the drone node. Due to weight constraints placed on the drone node PCB, a micro SD card was the most weight effective storage method. As well, through research, SD cards were a very prevalent method of data storage in embedded data logging applications; there were examples on the Internet showcasing data logging on SD cards with PIC microcontrollers.

The drone module power supply system consists of a lithium polymer (LiPo) battery, voltage regulator, and charging circuit. The LiPo battery has a capacity of 900 mAh, and can power the drone module for over 8 hours. The voltage regulator takes the 4.2V provided by the LiPo battery and converts it to 3.3V that is required by the rest of the components on the PCB. The charging circuit is used to charge the drone module through a USB port and consists of a microUSB input and a charging controller, and utilizes the 3.3V voltage regulator to provide power to the rest of the drone module circuit when charging. When dealing with small, sensitive electronic components such as in this design it is important to have a stable supplied power that is within the range specified by the manufacturer for the components and has no risk of surging or burning out equipment.

**Drone Node Software**

Software development on the drone node was performed using the MPLABX IDE with an XC16 compiler. MPLABX IDE provides on board configuration support for Microchip Technology PIC24 microcontrollers in the form of the MPLAB Code Configurator, as mentioned in the microcontroller section above. The PIC firmware can be divided up into three main subroutines: initialization, ground node communication, and server node communication.

It is important that any data stored on the drone node SD card cannot be recovered by adversary entities in the event that the drone is lost or stolen. To achieve this, data will be encrypted using the encryption standard (Advanced Encryption Standard) AES-128. AES-128 is deprecated in many large corporations due to a stronger encryption standard AES-256, which has a larger key size and is therefore more robust to brute force attacks. The microcontroller chosen for this project does not have the memory or processing power to effectively implement AES-256; as such AES-128 is used. This comment should not take away from the security granted by an AES-128 implementation. AES-128 continues to be an extremely secure protocol with a negligible probability of a successful brute force attack. According to the best estimate calculations from 2012, it would take about one quintillion years to execute a brute force attack on AES-128.

**Drone Node PCB**

Due to limitations in weight and size, the initial breadboard prototype could not be mounted to the drone. Instead, a PCB was designed and manufactured. The design of the PCB went through several iterations and needed to follow strict design rules to ensure crosstalk and interference would not be a factor. The final PCB ended up being a two-layered board with a thickness of 1.6 mm, width of 55.88 mm, and length of 73.66 mm.

In terms of layout, the most important design rule is to keep all data lines short and isolated. To do this, the SD
card mount and RN4871 were kept on opposite sides of the board with the PIC24 microcontroller placed at an approximately equal distance between them. This ensured that these data lines would not be put in a position to interfere with each other, potentially causing errors. As well, a minimum six mil spacing between traces was used to ensure that there would be no interference between lines running in parallel to and from the PIC24.

In order to design the PCB, EasyEDA was used to generate the footprints and draw the traces between the components. Gerber files were then generated of the EasyEDA design and sent to the manufacturer through PCBway. Once the ordered PCBs arrived, the components were soldered to the board using a standard soldering iron and 27 AWG solder.

**Server Node**

The purpose of the server node is to receive stored temperature data from the drone node. The server architecture can be broken into three primary components. The edge server, web server and cloud database. These components form the basis of the fog computing architecture. Fog computing describes a hierarchical network designed to store and analyze internet of things data. The most attractive feature of the fog architecture is its scalability. If more data is to be offloaded to the server node in the future, the server node would be capable of scaling to the larger data and computational demands. This criteria made Amazon Web Services (AWS) a perfect solution to host the server node. The general architecture is shown in Fig. 3.

![Server node architecture](image)

**Fig. 3** Server node architecture.

The cloud component of the server node was created using Amazon Web Services (AWS) and provides a location to store all data retrieved from the server node; theoretically many server nodes. AWS contains a suite of linked product offerings specifically designed to be quick to launch, easy to prototype, and simple to scale.

Amazon Relational Database (RDS) service provides a relational database in the cloud with many configuration options, capital free hardware resources, security, and near perfect availability. RDS and the open source Structured Query Language (MySQL) software together form the 20 Gibibyte (GiB) database used for this project. This 20 GiB database falls into the free tier of Amazon’s offerings, allowing creation of a cloud computing database at no cost for the duration of the project. Security was also paramount in deciding which database to use. Despite the relative insignificance of test data collected in this project, security online is one of the most important features of a project to consumers and corporations. RDS provides a very secure architecture as well as the ability to customize access to the database not only by user but also by Internet Protocol (IP) address. Individual user permissions are controlled by RDS as well as from within the MySQL database implemented in this project. This means that to create a user with database edit permissions you must not only have access to the MySQL database command line but also root access to the database through Amazon RDS. Security of data between the Web server and RDS is handled by AWS using the Virtual Private Cloud (VPC) service. The VPC service enables data to be transferred securely between the database and AWS web server preventing eavesdroppers from sniffing unencrypted data packets.

The web server has three important functions. The web server must be able to accept data from the edge server, upload data to the cloud, and securely present the data to the user through a website. These features were achieved using the AWS Elastic Compute Cloud (EC2). EC2 provides access to Linux server instances which are completely configurable. The Linux server was set up as a t2.micro instance which means it has a single Central Processing Unit (CPU) and 1 GiB Ram. This instance type was chosen for its sustainable CPU performance and solid state drive speeds. The server is configured in the same VPC as the database allowing secure communication between the two. The open source Apache HyperText Transmit Protocol (HTTP) server software was installed on the instance to serve the web page to viewers. Port 80 was opened to all IP addresses and Secure Shell File Transfer Protocol (SFTP) port 22 is designed to be open only to specific edge node IP addresses. A screen capture of the web site is shown in Fig. 4.

![Web site home page](image)

**Fig. 4** Web site home page.
When navigating to the website you will be presented with a login page. The login form was sourced online [12] for the visual appeal only. Behind the scenes is a login and registration system adapted from a secure login tutorial [13]. The decision to utilize a ready-made system mirrors the justification for using public encryption algorithms. This decision addresses the concern of vulnerabilities that arise in non-professionally written security systems. The login system utilizes session variables, JavaScript, PHP, and MySQL to create a secure interface. Passwords are hashed using Secure Hash algorithm 512 in a JavaScript implementation [14]. This means that all passwords are sent, stored, and encrypted in the MySQL database and cannot be compromised by an eavesdropping adversary. The security of the login system could be improved to prevent brute force attacks using Google’s free reCAPTCHA, however this was omitted due to limited project scope. Fig. 5 is a sitemap showing the redirects and PHP decision tree for the login page.

On successful login, the user will be redirected to the website home page. The home page plots a sampled set of data from the database. The chart is created using Google Charts. Google charts is programmed in JavaScript and the data is provided by an Ajax call. The Ajax call directs to a PHP script where the database is queried. The returned data is formatted in a proprietary JavaScript Object Notation (JSON) format set by Google Charts.

In order to programmatically get data into the database it must be routed through the AWS web server. A directory was set up to temporarily hold text files before the data is inserted into the database. The insertion process is handled by a PHP script which is automatically run by a Cronjob every 15 minutes. The script works simply by searching the directory for new files, parsing any new files, submitting the data to the database with an INSERT SQL statement, and finally zips the text file to conserve disk space. The database is currently configured with four tables. The DatamuleDB schema contains a ‘temperature’ table and a ‘nodes’ table in which respective temperature data and node metadata are stored. The ‘node’ table contains the information for each ground node, including the node ID, name, and latitude-longitude. The ‘temperatures’ and ‘nodes’ tables are designed to be joined on node id. To enforce this a foreign key constraint was added on the node id in the temperatures table. A second schema called secure login contains two tables. ‘Members’, stores the users and hashed passwords for the website login and login attempts stores the number of attempts made by each user. Fig. 6 shows the tables in the database.

The Edge Server has two primary jobs, to connect and receive data from the drone node and to transfer the data to the web server. For testing purposes an Apple Macbook Pro was used as the Edge server. Dedicated hardware would be preferred in a production environment. With that said, the macOS app created to function as the edge server is fully functional. Initially a JavaScript (JS) version of the edge server was created. The JS version relied on open source software which proved to be complicated and inefficient. The decision to change to a macOS app was made to gain more reliable control over the system’s Bluetooth hardware as well as a very large throughput increase. To get started with Apple’s Core Bluetooth
framework, a sample program [15] was used as a starting point. This sample code supplied the functions for discovering BLE peripherals (the drone node), connecting to BLE peripherals, discovering BLE services, and notifying on BLE characteristics.

4. Unit and System Testing and Results

The PCB was manufactured using a standard soldering iron and 27 American Wire Guage (AWG) solder. By using solder this thin, the small form factor components were able to be reliably soldered to the PCB. The soldering process was done incrementally, with each subsystem being added and tested.

The first subsystem implemented to the device was the system power. This included the battery connector, power switch, power LED, 3.3V regulator, charging LED, charging regulator, micro-USB connector, and all corresponding resistors and capacitors. After each of these were soldered onto the board, short circuit tests, connectivity checks, and voltage readings were done. All of these resulted in positive tests that indicated that the subsystem was working properly. However, after introducing a 5 V source via the USB, the charging integrated circuit was found to be drawing too much current to safely charge the battery. Fig. 7 shows the relationship between the resistor used in the programming pin of the charger and the current draw. It was determined that the 2 k resistor used was not accurate enough given the sensitivity of the charger. To remedy this, a much larger 4.7 k resistor was used in its place. This allows the battery to be safely charged, albeit at a slower rate.

![Charging Current Draw](image)

**Fig. 7** Charging current draw.

The next subsystem to be integrated onto the board was the PIC24 microcontroller. This was done so that there would be no issues regarding the test of the microcontroller affecting other components by setting pins high and low. The microcontroller also proved to be the most difficult component to solder due to the small spacing between the pins for the Surface Mount Device (SMD) form factor. Once it was soldered onto the board, the same tests were conducted to ensure proper connection. As well, basic sample code was programmed to the PIC24 through the PICkit 3 debugger in order to check that the pins were functioning properly by going high and low in a loop. This was verified by using a multimeter on the pins to see the 3.3 V drop to 0 V and back again.

The USB-to-UART converter was the next subsystem to be integrated. This includes the MCP2200, 12 MHz resonator, and corresponding resistors/capacitors. Once soldered, all connectivity checks were made, as well as a test to ensure data is being received by the UART. After plugging in the USB, the computer was able to recognize the device and connect to it through CoolTerm. Test data was then sent on monitored lines to ensure proper function. The next section required to be integrated was the RN4871 BLE module. This system includes the RN4871, all status indicator LED’s, the reset button, the mode switch, and corresponding resistors/capacitors. The soldering of this device went smoothly as well, with it passing all connectivity checks on the first pass. It was then tested by using the USB-to-UART converter to update the firmware of the RN4871. This was done using the isupdate software tool provided by Microchip. Through this testing the various indicators lights were monitored with no issues. Once updated, CoolTerm was used to send test commands to the RN4871 in order to see if it responds correctly. This was done by sending "$\$\$" to the RN4871, expecting to see CMD> be returned. Once that was verified, the PIC24 to RN4871 connection was tested in a similar manner to ensure that commands from the PIC24 were being sent and received properly. Once again, the CMD> was returned showing proper functionality. The final subsystem to be integrated was the drone’s SD card module. This included the SD card mount, 4050 buffer, SD power LED, and corresponding resistors. Once it was fully soldered, connectivity checks were conducted, followed by tests using sample code to write simple files to an SD card. After some issues with mounting, it was determined that the SD card being used was faulty, causing it to be replaced. After this, the SD card was able to mount, read, and write with the PIC24 successfully. As a final check, files from the ground node were transferred to the RN4871, then to the PIC, then to the SD card, then back to the PIC, then to the Cloud server. This was monitored using the MCP2200 showing that the system was successfully integrated.

**Functional Integration**

Connectivity between the ground node and the drone node was fully established using the connection procedure defined in the code, whereby the drone node connected directly to the ground node’s Bluetooth MAC address. A topographic survey of connectivity was performed.
between the ground node and drone node. Connectivity between the modules was established in a semi-circle pattern with 45 degree resolution. As shown in Fig. 8, connectivity was tested to 31 feet at each of the desired positions. In these figures the different colours represent the different altitude planes where connectivity was established. Each connection point was validated by performing a full rotation of the drone node around the z-axis to simulate potential approaches to the ground node by the drone itself. Each rotation was also performed in increments of 45 degrees, by connecting and waiting for a disconnect between the ground node and drone node. Connecting to the ground node was reliable with no ambiguity as to the connection status. No disconnections were observed. Further testing was performed past a 31 foot radius from the ground node as the connectivity exceeds the performance metric of 5 meters. This connectivity is supported by Fig. 9 demonstrating the antenna characteristics of the RN4871. We can see that the module radiates strongly in most orientations. Based on our integrated design, connections would be coming from what is labelled as the xy-axis in Fig. 9, where the radiation pattern has high gain of 0 dB to -5 dBi.

5. Conclusions and Future Work

The object of this project was achieved: to develop a proof of concept for a method in which to gather data from sensors located in dangerous or remote environments through use of a drone. A drone was outfitted with a PCB having an onboard microcontroller, communications module, and non-volatile memory, which when hovered within range of either a ground node or a server node, functionally uploaded and offloaded data, respectively, thus acting as a fully functional data mule.

The design of this project has met the performance characteristics pertaining to battery life, range, memory, and security. For instance, a battery life of greater than 8.5 hours was achieved, a connectivity of 9.4 meters was measured, an expandable memory of 16 GB was added to the node, and basic security measures were implemented on the server.

The total implementation cost of a single drone node PCB was $31 CAD, meaning that this design represents a relatively inexpensive method of transferring data through use of a data mule.

The following describes future potential improvements for each component the project.

Ground Node

The Arduino which the ground node is based on does not have a real time clock. In a real world scenario, not having a date stamp would render any data from the sensor useless. For this reason we have ordered and received a real time clock I2C module that should be implemented prior to demonstration. There is a risk that the Arduino may not have enough available memory for the real time clock implementation which may prevent us from completing this in time. Because the ground node was more of an accessory to our main scope than within, we took the simplest, highest probability of success route by choosing the Arduino. For a production environment we recognize that the Arduino is not a viable solution; a customized PCB would be more efficient. Finally, some stretch goals may include a solar power system, integrating more features such as wind speed and direction detection, and protection from the environment.

Drone Node

Given the drone node was our primary focus, the potential improvements lie primarily in the software and would bring us closer to or surpass our performance metrics. The encryption function of the drone node is currently incomplete. There is a formalized plan to implement encryption as described in the Drone Node Encryption section prior to demonstration. We believe this is a very achievable goal with less risk compared to the real
time clock on the ground node. An important performance metric which was not achieved was the throughput speeds between the ground node and drone node. Falling short of the 10 kbps goal at about 1.2 kbps, there is much room for improvement. The primary factor affecting this speed is the interactions between the ground and drone node’s BLE modules. As they are two different manufacturers they provide two unique transparent UART services. The transparent UART mode is the most efficient method for data transfer and was successful for the server to drone connection at roughly 18 kbps. Choosing these modules to be of the same make would enable these transparent services and speed up the exchange of information.

As well, further integration with the drone controls would allow for more autonomy in the usage of this project. Letting the drone node communicate to the drone when files are done transferring would allow for more efficient travel times.

Server Node

The Server node, much like the ground node did not receive as much attention as the drone node. This leaves it in a state with many potential improvements especially because the scope of the server node is very large. Across the entire server node, the GUI requires an overhaul and addition of end user features. More interestingly, the potential of AWS has hardly been utilized. As the project stands, we are simply storing data in the cloud. A more dedicated machine learning project could built off of the architecture created by this project. The potential use cases for machine learning with large data in the cloud are unlimited. The edge node server is currently written exclusively for macOS. Given the remote nature of the problem, future iterations should include more platforms and most importantly mobile platforms. Mobility devices tend to have the furthest reach and therefore would make a more ideal server node.

References


