Final Paper for Spring 2018 EE 396/496 2nd Generation Relay Node: Team Bumblebee

Authors: Kayla Amano, Alexander Moore, Rebecca Rupley, Alexander Wong

Date: April 28th, 2018

Abstract: The Smart Campus Energy Lab (SCEL) sensor node network wirelessly communicates collected meteorological data via an Xbee device to the lab in Holmes Hall. Current sensor nodes developed by SCEL have a limited range in densely populated areas such as the University of Hawai'i at Manoa campus. Therefore, the relay module was developed to extend the range of the sensor node network. The relay node utilizes an Xbee Pro S2B for wireless communication, an Atmega328P for its microprocessor, and a solar panel with a rechargeable battery for power. Along with these components a PCB, firmware, and housing needed to be developed to create a working relay communication node. Preliminary range testing with the module has shown that range is affected by not being in line of sight of another node. More testing will be conducted to test the exact limitations of the module and how to make a more robust network.

1 Introduction

The Smart Campus Energy Lab (SCEL) is one of many research laboratories within the Center for Renewable Energy and Island Sustainability (REIS). In 2012, the University of Hawaii at Manoa paid 35 million dollars in electricity bills, prompting the push to look at energy saving and renewable energy options. The objective of SCEL is to develop technologies and practices to promote sustainability and renewable energy usage. Part of the motivation behind the projects of the lab is to also keep the University of Hawaii at Manoa, and Hawaii in general, in line with the Renewable Portfolio Standard (RPS) goal of reaching 100% by 2045 (Securing).

Currently, the main project of the lab is the development of low-cost and reliable environmental sensor nodes meant to collect meteorological data, such as temperature, humidity, and solar irradiance. These sensor nodes are meant to be placed on rooftops across the University of Hawaii at Manoa campus. Team Bumblebee's objective was to create a relay to extend the range of our current sensor node network.

The Bumblebee box does not have any sensors and therefore does not collect any data. The ideal location for bumblebee boxes, would be between distant sensors and the lab gateway. Bumblebee is based off of the third generation of sensor nodes, Cranberry, and uses many of the same components, such as the Atmega328P MCU, the Xbee Pro S2B, and being powered by a solar panel and rechargeable battery. However, the Bumblebee box has a simplified circuit and its own PCB design and housing. The Bumblebee project is also interested in range testing and networking with the Xbees.

2 Bumblebee Relay node

According to the datasheet, the Xbee Pro S2B has an outdoor line of sight range of 2 miles. However, testing has proven that packets can be dropped at a much closer range, even less than one mile. As the ultimate goal for this project is to have sensor nodes on multiple roof across the University of Hawaii campus a device to relay the data from distant roof was necessary. Started in Spring 2017, the Bumblebee relay node is a second generation communications node designed to relay meteorological data collected by the other sensor nodes. The first generation relay node, Ant, was started in the Fall 2016 semester and was based off of the generation Apple sensor node. The main goal of team Bumblebee was to reimagine Ant to be compatible with the generation Cranberry sensor node.

The goals for Spring 2017 semester included designing and fabricating a circuit board, doing Xbee field tests, and creating a working relay node. By the end of Spring 2017 a completed prototype of Bumblebee, built on a breadboard was successfully completed. The firmware of the current sensor nodes was modified to relay packets and testing has shown that it is able to relay sample data packets. Some basic field tests testing the effects of distance and not being in line of sight were conducted. The goals for Fall 2017 included completing a schematic and PCB design, designing housing, and fabricating the PCB.

By the start of Spring 2018, the PCB design and a preliminary housing design were complete. With a new housing team dedicated to designing and fabricating the housings for all of the teams in the lab, the focus this semester was on having the PCB printed, populated, and programmed. By the end of the Spring 2018 semester, we expected to deploy two completed Bumblebee boxes.

2.1 Design

The design of Bumblebee was inspired by the design of Cranberry. Unlike Ant, which used an Arduino Uno board, Bumblebee uses only an Atmega328P as its microcontroller. This will help to reduce the size of the board and relay node as a whole. What differentiates Bumblebee from Cranberry and the other sensor nodes is that Bumblebee does not contain any sensors or collect any data. Bumblebee's only purpose is to relay data, increasing the range at which sensor nodes can be placed. Because of this, the only major components that the Bumblebee board has are the microcontroller, the Atmega328P, the solar charging chip, and the Xbee Pro S2B radio module.

2.2 Block Diagram

The block diagrams describe the overall design of both the power system and the communications of Bumblebee. Figure 1 below shows how the different sensor nodes will communicate wirelessly.

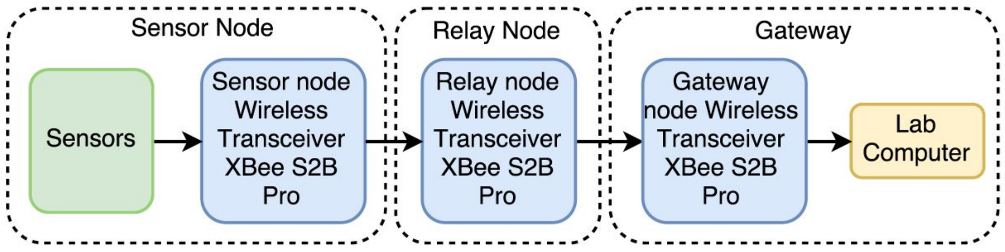

Sensor Node
Sensors
Sensor node Wireless Transceiver XBee S2B Pro
Relay Node
Relay node Wireless Transceiver XBee S2B Pro
Gateway
Gateway node Wireless Transceiver XBee S2B Pro
Lab Computer

Figure 1 Bumblebee Signal/Communication Block Diagram

The signal/communication block diagram describes the basic path that data will travel from the sensor node to the lab gateway computer. For example, Cranberry will collect data from its sensors, construct a packet with the data, and then send the packet to the Bumblebee relay node. Bumblebee will then receive the packet and forward it to the gateway Xbee, which will then be sent to the lab computer. The above block diagram is simplified because it is possible that in order to reach the lab gateway the packet will need to be passed along by multiple Bumblebee nodes. Currently, the block diagram only shows a single sensor node sending data to the Bumblebee relay node. In the future it may be possible to have multiple sensor nodes sending data to a single relay node. Or, depending on the location of the sensor and relay nodes, there may be multiple paths that converge or diverge.

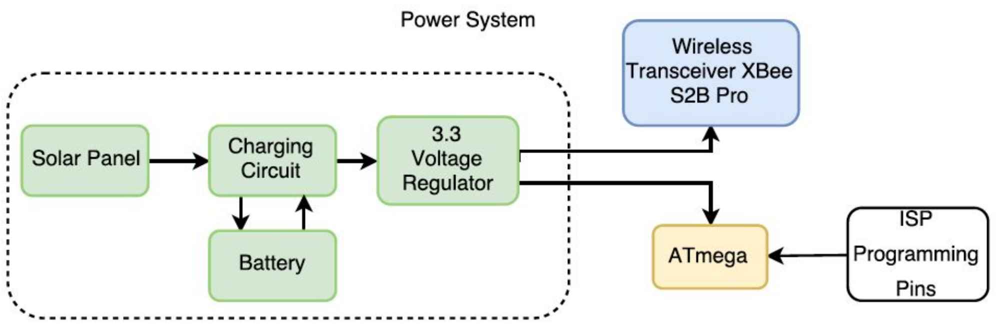


Figure 2 Bumblebee Power Block Diagram

The block diagram for the power system can be seen in Figure 2 above. The power block diagram describes how all of the hardware components are connected. Along with the circuit board, the Bumblebee relay node would incorporate a solar panel and charging circuit to help supply power to the board. The sensor node boxes use two 3.3 V voltage regulators, one for the Xbee and one for the Atmega. The sensor nodes also utilize a 5V boost converter to step the 3.3 V up to 5V for the sensors to operate. However, because Bumblebee does not use any sensors and the Atmega and Xbee can operate at 3.3V, a 5V boost converter is not needed. Two voltage regulators are used to supply enough current to the circuit, but because not as much current is needed for Bumblebee only one 3.3 voltage regulator is required. Eliminating the sensors also helps to decrease the power consumption and the amount of components needed. At this time the prototype of Bumblebee is on a breadboard and receives power from a laptop.

2.3 Bare Arduino Bumblebee

Figure 3 shows the circuit for the Bumblebee relay node. This circuit is referred to as the bare arduino because it does not include the complete arduino board, but only the Atmega328P microprocessor and a few necessary passive components. Without the sensors the Bumblebee box is very simple and does not have many parts or connections. Getting this prototype to work was the first step in creating a successful relay node.

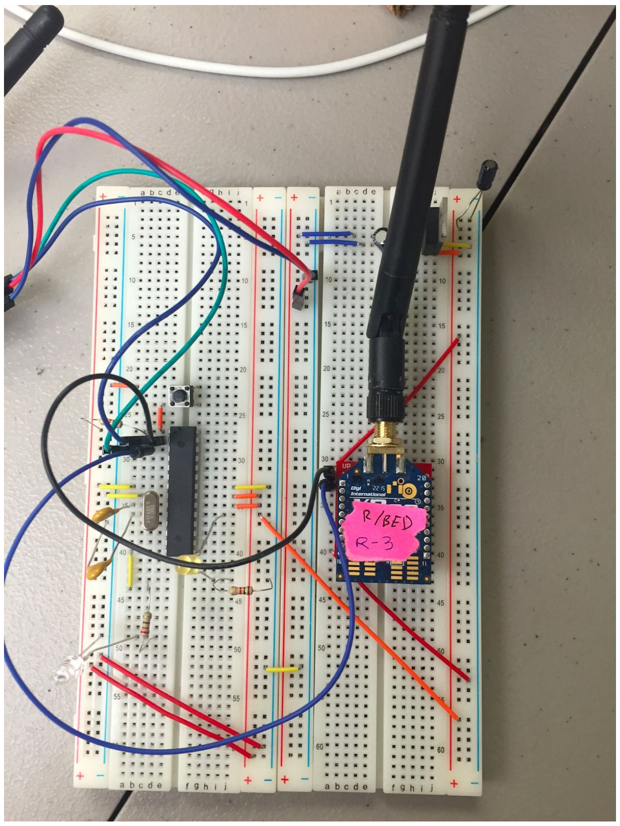


Figure 3 Bumblebee Bare Arduino on Breadboard

The bare arduino includes all components of the Bumblebee relay. These components includes: the Xbee radio module, the Atmega328p, the 3.3 voltage regulator, the FTDI programing circuit, the 8MHz external clock, and the necessary passive components.This prototype is fully functional, and in the end will be transferred into the Eagle schematic and board layout.

2.4 Eagle Schematic and Board Layout

In Fall 2017, we were able to complete an Eagle schematic and PCB layout for the Bumblebee relay node. This proved to be a challenge at first since no one on our team has had much experience using Eagle. Since our design is based off the Cranberry Weatherbox, we used Cranberry's schematic and our prototype to create the schematic for Bumblebee. It was decided that the design for our board will include all surface mount components except for one through-hole header for the FTDI. The schematic can be seen in Figure 4 below.

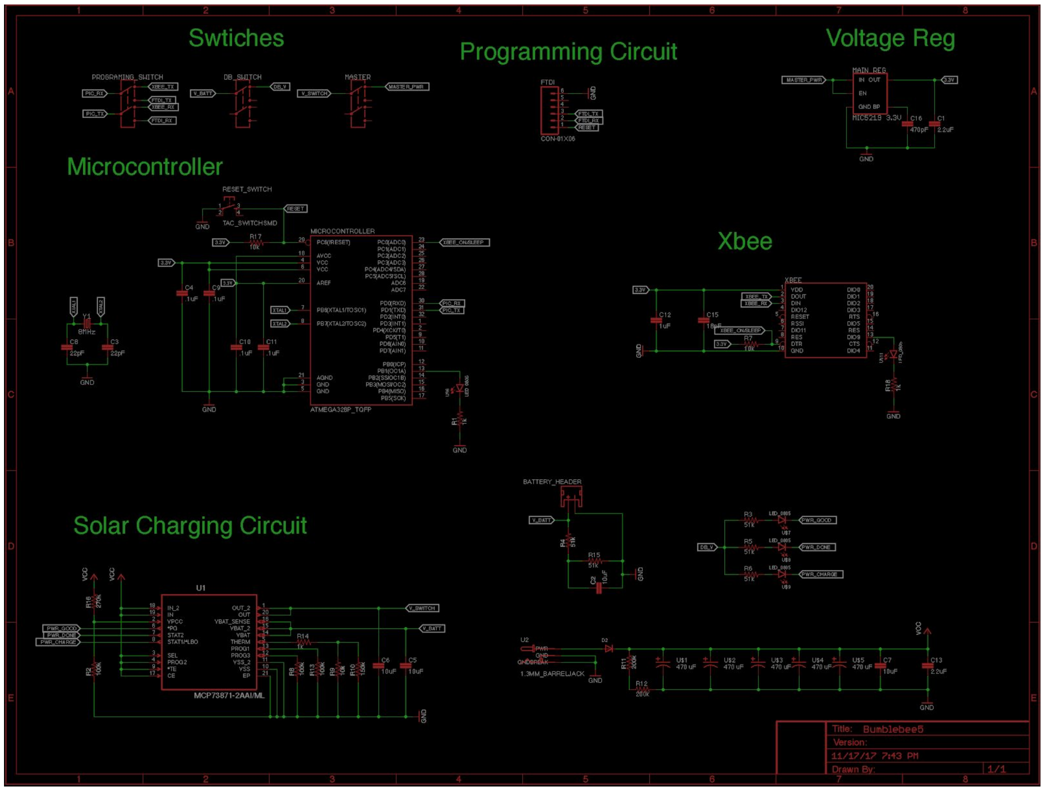


Figure 4 Bumblebee Eagle Schematic

There were a couple of additions that were implemented to our schematic that are not on our bare arduino. An Xbee sleep line was added to allow for the Xbee to be remotely put into sleep mode. The sleep mode of the Xbee is a feature that is yet to be implemented into any of the generations of sensor nodes. The ability to put the Xbee to sleep would lower the power consumption by putting the Xbee to sleep when data does not need to be transfered. This feature would most likely be implemented during the night, when there is no solar irradiance data to collect or send. Decreasing the power consumption would effectively increase the uptime of the sensor and relay nodes. With the help of the software team, the Xbee sleep mode can soon be implemented into the sensor node network.

Another component that was added would be a programming switch. In order to program the board, there must be nothing connected to the RX and TX pins of the Atmega. Currently the RX and TX are manually disconnected from the circuit when trying to program the board. The addition of a programming switch will help solve that problem. Flipping the switch would disconnect the RX and TX of the Atmega from the circuit. Flipping it back would connect the RX of the Atmega to the TX of the Xbee, and the TX of the Atmega to the RX of the Xbee. Instead of this programming switch, there is a software solution to getting the atmega to reset when programming. This was not implemented in this version of Bumblebee because we were pressed for time and decided to not to add another potential factor into an otherwise working prototype. This feature should be looked into and implemented in future versions of the Bumblebee relay node.

Instead of using the breakout board of the solar charging chip, we used the PPM 20 QFN IC microchip along with the necessary passive components. We followed Cranberry's schematic to implement the charging chip directly to our design. This chip will be used to help keep the relay node self sustaining. The solar panel will collect solar energy and use the solar charging chip to charge the battery. The solar charging chip also has three different colored LEDs used to tell the status of the charger. One LED for on, one for charging, and one for done charging. These LEDs are connected to a switch, so that we can turn off these LEDs along with any other debug LEDs to save power.

One of the more challenging parts of the project was designing the board layout. There was a learning curve to using Eagle. There are a couple of things to keep in mind when making a board layout. One tip would be not to have any right angles when routing. Right angles in the routes could cause a reflection of the signals that can cause problems with the functionality of the board. Another tip was to make sure traces don't come in at an angle into the components. All decoupling capacitors should be physically near the components they are connected to according to the schematic. Another important thing that was considered when making the design was making sure there are no metal parts near the antenna of the Xbee. Having metal on the board by the antenna of the Xbee could possibly cause interference with its communication signal. This could lead to possible unwanted data packet errors and dropped packets.The Xbee, battery header, and solar panel barrel jack were strategically placed on the edges of the board to allow for easy access when plugging in the devices. The antenna of the Xbee will not be interfering with the rest of the board. The placing of these three components are taken into account when making the 3D housing model for Bumblebee.

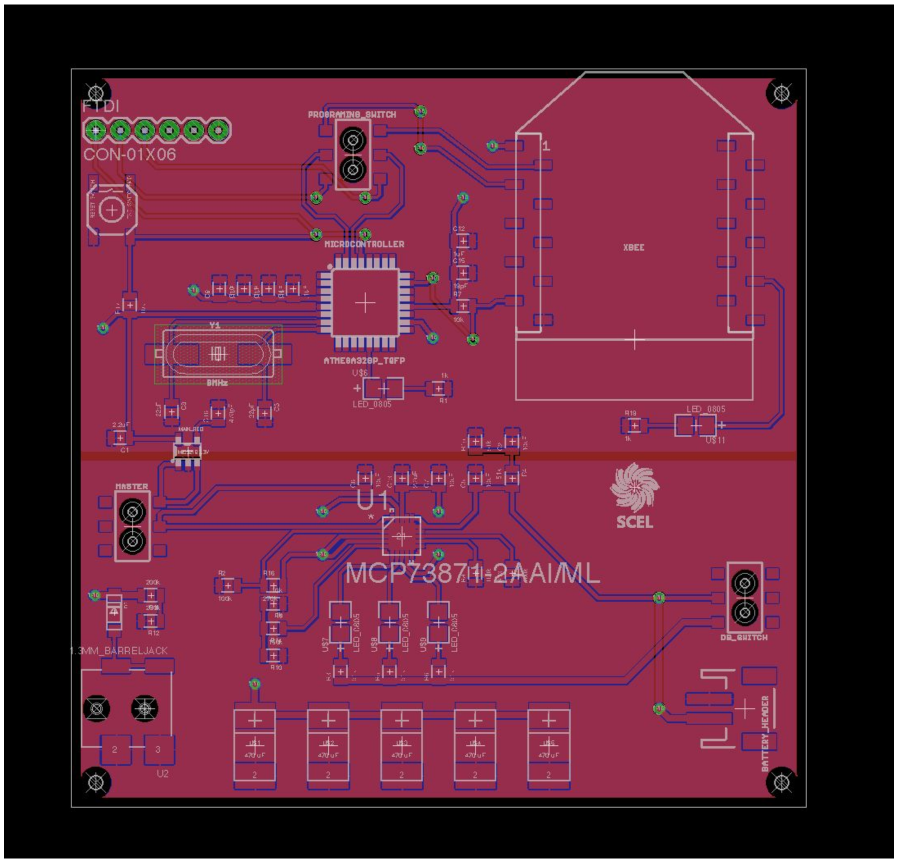


Figure 5 Bumblebee Board Layout

The board is 3 inches by 3 inches with a top layer and a split bottom layer. The top layer is ground while the bottom layer is split in half. The upper half of the bottom layer is 3.3V and the lower half is VCC. The bottom layer was split like this because the components on the upper half connect to 3.3V while the components on the lower half connect to VCC. This design choice allowed for less traces with the use of vias. Keep in mind when creating polygons, the line width should be set to a number larger than zero. When running the necessary CAM (Computer-Aided Manufacturing) files, you will run into a warning saying that there is “extremely large plot data.” The CAM process would attempt to fill in the polygon with the smallest size lines. If the polygon line width is set to zero, the CAM job will try to fill it with an infinitely small line which causes the large plot data warning. The CAM job will take an extremely long time to complete the otherwise less than 5 second process. The signal traces are set to 12 mils, while the power traces are set to 14 mils or larger. The larger power traces allow for more current to flow which is good for supplying power. The passive components are 0805 packages. All of the components are on the top of the board. The overall size of the board can possibly be decreased if components are placed on the bottom of the board as well.

2.5 Bootloading

Before the microcontroller can be programmed it must be bootloaded to operate at 3.3 V and 8 MHz. If using the DIP package of the Atmega328P the chip can be placed on a breadboard and then connected to the corresponding pins on an Arduino Uno board. The Arduino Uno board will be bootloading the chip. The connections needed will be listed from the Arduino Uno to the chip. The connections are digital pin 10 to the reset pin, digital pin 11 to the MOSI (pin 17), digital pin 12 to MISO (pin 18), and digital pin 13 to SCK (pin 19). The reset, MOSI, MISO, and SCK pins are collectively called the ISP programming pins. The chip should also be powered with 3.3 V and an external 8 MHz clock can be connected to the XTAL pins. Then, Nick Gammon's bootloading process was followed. The board detector sketch was uploaded to the bootloading Arduino Uno and then the program was ran. If the board has been previously bootloaded the board detector sketch will notify the user. Next, the Atmega board programmer sketch is uploaded to the Arduino Uno. When the sketch is ran the 8 MHz chip bootloader is selected and the chip will be bootloaded.

For the Atmega328P TQFP chip which is surface mounted onto the PCB the bootloading process is the same, but with a different pinout. On the TQFP chip the reset is on pin 19, MOSI is pin 15, MISO is pin 16, and SCK is pin 17. Also, in order to connect the pins to the bootloading Arduino Uno thin wrapping wire had to be soldered to the leads of the reset, MOSI, MISO, and SCK pins. However, these connections are not the most stable and can cause errors in the bootloading process. If the board is not detected connections should be checked and sometimes it will take multiple attempts to bootload successfully. In order to have better connections it is suggested to include a header on the PCB with traces from the header to the ISP programming pins. Once the board has been bootloaded the chip is ready to be programmed through the FTDI. The current PCB design includes a header for the FTDI.

2.6 Housing

This semester, the housing design was created by the housing team. The 3D model of the housing was created in SolidWorks. The purpose of the housing is to protect the circuitry from weather conditions. We wanted to take a different approach to the design of the housing than the other weatherboxes housing designs. The other designs are simple rectangles with a variation of covers and other components. Since our team name is Bumblebee, we requested that our housing be designed in a hexagon shape that would be similar to the honeycomb structure. We felt that this shape was strong enough structure to withstand the different weather conditions. The housing design can be seen in Figure 6 below.

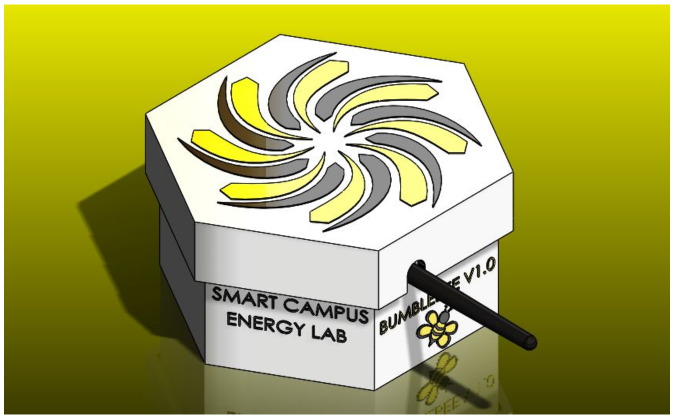


Figure 6 Bumblebee Housing Design

The design created by the housing team is a simple hexagonal box. Inside the box, there is a shelf for the PCB to rest on. The battery would lie underneath the shelf. The solar panel would then be secured to the box with zip ties. The entire box with the solar panel would then be zip tied to a clamp on the roof of Holmes Hall. One large hole is cut on one side of the box for the Xbee antenna to stick out of and another smaller hole is cut on another side for the solar panel jack to attach to the PCB inside.

Unfortunately, this housing design could not be implemented due to limitations of the 3D printers in the lab. As a result, we opted to use a mass produced plastic box seen in Figure 7 below.

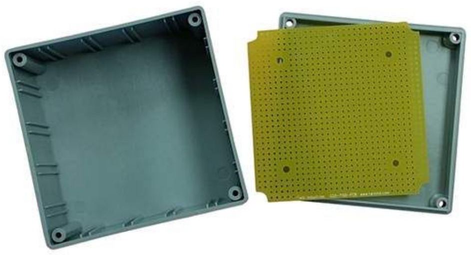


Figure 7 Plastic Box Used for Housing

The PCB corners needed to be shaved off in order for the board to fit inside the box. The battery was placed at the bottom of the box with the PCB resting on top of the battery. Holes were drilled into the box to accommodate for the Xbee antenna and the solar panel barrel jack. Additional holes were drilled close to the top of the box for zip ties to be fed through. Once the PCB and battery were properly positioned in the box and zip ties were inserted in the holes, the box was screwed shut. The box was then thoroughly wrapped in Gorilla Tape with extra tape being applied over the seams on the box and the holes in the box in order to ensure the box was water tight. The box and the solar panel were then secured together using the zip ties attached to the box and the entire set up was secured to two clamps on the top of Holmes hall using the same zip ties. Although we were not able to make an in house housing in the desired shape of a hexagon, we were able to create a low budget housing that met the housing specifications of the original design using a cheap plastic box.

2.7 Packet Relay Testing

When testing the Xbees' ability to send and receive packets the programs XCTU and Arduino IDE were used. XCTU was used to configure the Xbees, manage the network, and conduct communication testing. Xbees can be configured in either API or AT mode. AT mode is known as the transparent mode, where data is sent immediately to the destination and is not formatted into a packet or frame. API is Application Programming Interface and requires for the data to be formatted into a packet. Larger networks require Xbees to be in API mode because AT mode only works between two Xbees talking directly to each other. Xbees are also configured as either a coordinator, router, or end device. It is necessary for every network to have one coordinator that is responsible for maintaining the health of the network. Xbees are identified by their MAC addresses, which can be divided into two parts the serial high and the serial low components. The Xbees can also be given names, so for testing the Xbees have been named Bumblebee 1, Bumblebee 2, Bumblebee 3, and Bumblebee 4.

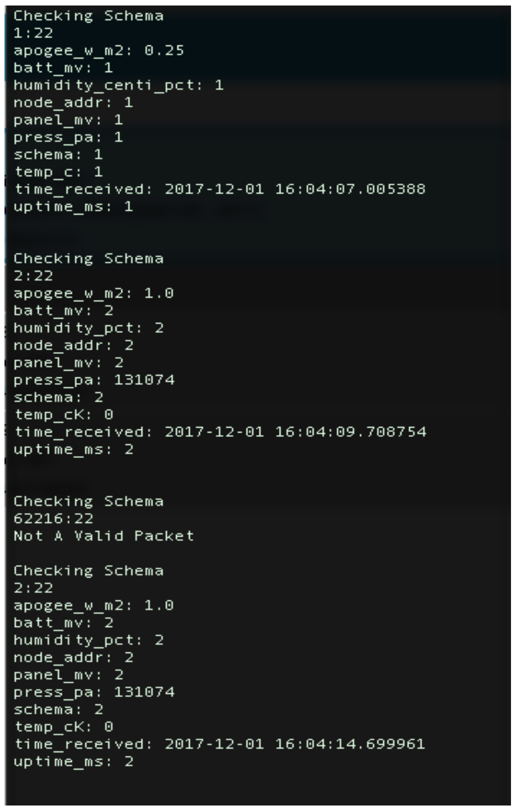
For packet testing the Xbees were set in API mode with one Xbee set as a coordinator and the other two set as routers. XCTU is able to generate and send API frames and is then able to decode the frame on the receiving end. Where the packet is sent from and the contents can be seen in the XCTU console. In order to communicate with each other the Xbees have to have the same PAN ID. If a specific address for a destination Xbee is not specified then the packet will be broadcast and sent to all operating Xbees with the same PAN ID. While testing the Xbees' ability to send and receive when connected to a laptop and XCTU, it was found that if the coordinator tries to send a frame it will receive its own frame back and not send it to the other Xbees. Therefore, the coordinator was used as the receiving Xbee and routers were used to send packets and relay packets. End node Xbees were not used.

As a second step, communication between an Xbee attached to an Arduino board and an Xbee connected to XCTU was tested. In order for the Xbee to send data packets at certain time intervals, a program had to be written and uploaded to the Arduino. There is an Xbee library by Andrew Rapp for the Arduino IDE that can be found online that has all the functions needed to write, send, receive, and read packets. There are also many other functions included in this library that were not needed for our purposes. Two separate programs were written, one for sending a packet and one for receiving a packet. The sending packet is configured as an 8 byte integer. Once the packet was received and read the packet was printed to the serial monitor to check that the contents were correct. The Xbee could be programmed to either send a packet to a specific Xbee or simply broadcast to all the Xbees; however, the focus for these tests was direct communication. Which Xbee the packet was sent to was determined by the serial high and serial low address input in the program's address section. Initial test packets sent contained a simple string, "Hi." Numbers and other special letters were also tested to be included in the packet. Throughout testing the maximum size of the packet was never reached, so no missing information was noticed. The Xbee connected to the Arduino could also receive packets sent by the Xbee connected to XCTU. The contents of the received packet could then be viewed on the serial monitor of Arduino.

Next, an Xbee connected to the bare Arduino board and an Xbee connected to XCTU was tested. Packets could be sent and received both ways. Then, a third Xbee connected to an Arduino board was added to the network. The Xbee connected to the Arduino would send a packet to the bare Arduino Xbee, which would then relay the packet to the XCTU Xbee. In order to relay, the code of the relay Xbee was modified to have both the receiving and sending code. The code required the received packet to be read, copied into a new packet, and then was sent to the gateway Xbee's address.

After the success of these tests it was attempted to use the gateway simulation and a sensor node test packet to test Bumblebee's ability to relay an actual packet. Up until this point only string packets were being used, however actual sensor node packets are set up as a struct with multiple variables inside. Because there were no sensors to generate data for the packet, values were hardcoded. In order for the packet to be recognized as a valid packet the size of the data packet needed to be 22. Also it was necessary to have a recognized schema number, so that the gateway knew how to decode the packet accordingly. The schema number is the part of the packet that tells the gateway which sensor node the data is coming from. With a different schema number, we will be able to differentiate which arduino the data is coming from. The test packets used the schema number 1, which should be for the first generation of sensor nodes, Apple. From testing, it was shown that the code used to forward a simple string packet could also forward a test sensor node packet.

Another test was done to see if the relay node was able to handle data packets coming from multiple sources. The same setup with the gateway simulation was used, expect with one more Xbee connected to an arduino board. This new Xbee was programed to send to a sample test packet, but with a different schema number. It used the schema number 2. An example of the output of the gateway simulation can be seen in Figure 8 below.



```
Checking Schema
1:22
apogee_w_m2: 0.25
batt_mv: 1
humidity_centi_pct: 1
node_addr: 1
panel_mv: 1
press_pa: 1
schema: 1
temp_c: 1
time_received: 2017-12-01 16:04:07.005388
uptime_ms: 1

Checking Schema
2:22
apogee_w_m2: 1.0
batt_mv: 2
humidity_pct: 2
node_addr: 2
panel_mv: 2
press_pa: 131074
schema: 2
temp_cK: 0
time_received: 2017-12-01 16:04:09.708754
uptime_ms: 2


Checking Schema
62216:22
Not A Valid Packet

Checking Schema
2:22
apogee_w_m2: 1.0
batt_mv: 2
humidity_pct: 2
node_addr: 2
panel_mv: 2
press_pa: 131074
schema: 2
temp_cK: 0
time_received: 2017-12-01 16:04:14.699961
uptime_ms: 2
```

Figure 8 Gateway Simulation Output

It can be seen that the relay node has the ability to receive and send data from two different sources to the gateway. This was achieved by offsetting the sending times of the two sending Xbees. Both of the sending sensor nodes sent data every 5 seconds, but with a delay of about 2 seconds between them. In the simulation output image it is shown that the first packet with schema number 1 arrives at time 16:04:07.005388 and the second packet with schema number 2 arrives at 16:04:09.788754. It is also seen in the output that the gateway returned “not a valid packet” when checking the schema for the third packet. The exact reason for this is unknown, but one possible explanation is that at some time points the relay node is receiving data from both sending nodes at the same time and getting confused. Another possible explanation is that once the sent packet is sent it is not cleared away to make room for the next packet and so the new packet is added to the last packet. Therefore, the length of the packet is no longer correct. More testing will have to be done to polish the relay node’s ability to relay data packets from multiple sources reliably.

The relay node was then tested with an actual Cranberry data packet. Originally the Cranberry node was configured to broadcast the data packet to all Xbees in the area with the same PAN ID. However, possibly because the coordinator was close enough to the Cranberry node, the relay node would not receive the data packet. In order to test the ability of the relay node the sending address in the code of Cranberry’s board was set to the relay node’s Xbee address. Then, the relay node’s sending address was set to the gateway Xbee’s address to pass the data packet to the gateway. Testing was successful and this was implemented into the final product. As it is any sensor node can set their sending address to the relay node’s and the relay node will be able to forward the data packet.

2.8 Range (Field) Testing

According to the datasheet, the Xbee Pro S2B, as seen in Figure 9 below, should have an outdoor RF line-of-sight range of up to 2 miles, and a indoor/urban range of up to 300 ft. However, obstacles and interference can greatly reduce the performance of the Xbee. In order to determine what the performance of the Xbee would be in the UH Manoa campus area one of the goals of Team Bumblebee was to conduct range testing. Range testing was not conducted in the Spring 2018 semester in order to focus on fabricating a complete Bumblebee relay node. However, in previous semesters Team Bumblebee conducted range testing and the results are shown below.

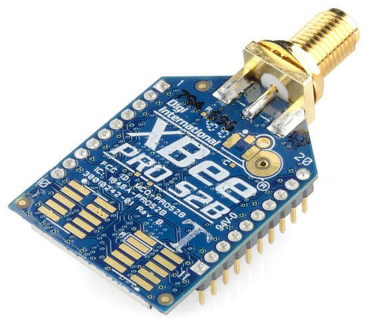


Figure 9 Xbee Pro S2B

Range testing was conducted, taking into account as many variables as possible. One of the first tests conducted was straight line-of-sight test. This was done on the 4th floor of Holmes Hall with one Xbee kept at one end of the building, while the other was moved away in increments of 30 ft all the while keeping line-of-sight. The built-in range test on the XCTU software was used for this testing. When running these test, the data values looked at included:

Local Strength, Remote Strength, Packets sent, Packets received, TX errors, Packets lost, and Percentage of packets received. Ideally, the signal strengths should have been around -32 dBm, no TX errors, and 100% packets received. For the line-of-sight test, the results show that at 390 ft the signal strengths were around -63 dBm and -65 dBm with 100% packets received. Figure 10 is a chart showing distance vs the received signal strength.

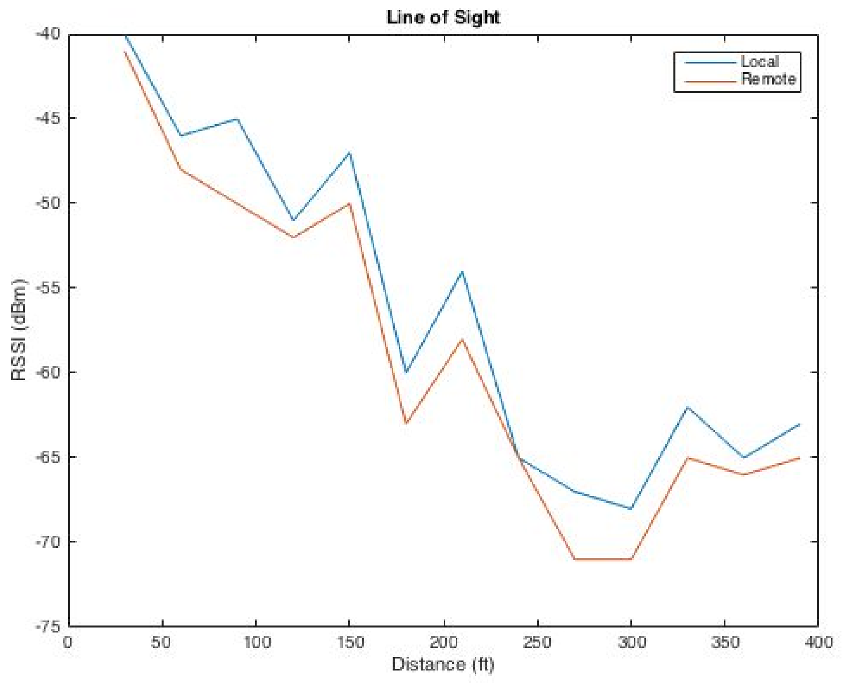


Figure 10 Distance vs. RSSI for Line of Sight

The next variable tested was not line-of-sight (through obstacles) and was also conducted on the 4th floor of Holmes Hall. Using the same setup, one Xbee was kept at one end of the building while the other moved into the next hallway being sure to stay out of sight. Immediately, at the 72 ft mark 6 packets were lost and Tx errors began to occur. As the distance between the two increased the number of Tx errors also increased until all packets were unable to be sent. Also at around the same distances the signal strengths were significantly lower. This data is shown in Figure 11 below.

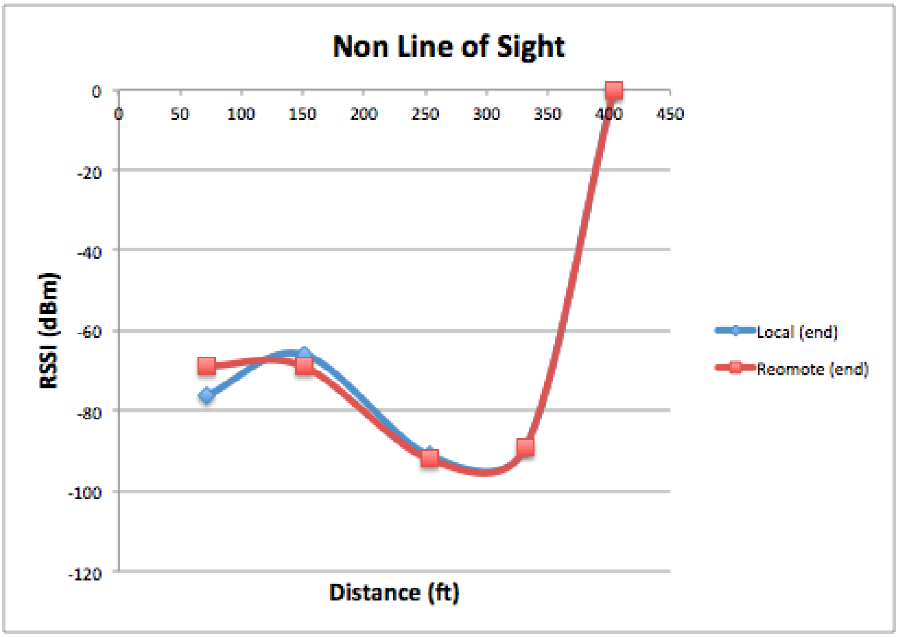
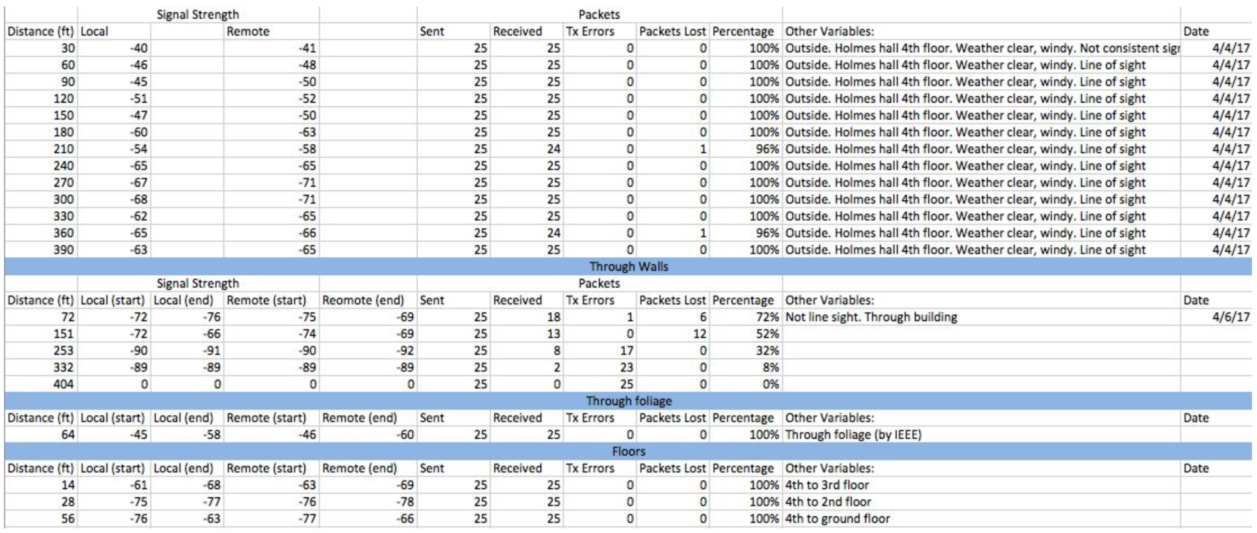


Figure 11 Distance vs RSSI for Non Line of Sight

Non line of sight testing between the floors of Holmes Hall was also conducted. From the fourth floor to the first floor, which is approximately 56 ft, there was 100% packets received and the signal strength was varying between -77 dBm and -66 dBm. As you can see, the signal strengths are a little weaker than the line-of-sight values, but we were still able to receive 100% of the packets.

There are still other useful variables to test; one of which include the effects of different weather conditions. Being in Hawaii, rain can be very prominent at times and it would be good to know if rain would affect the range or strength of the RF signals between the Xbees. All of the range testing data we gathered can be found in Table 1 below.

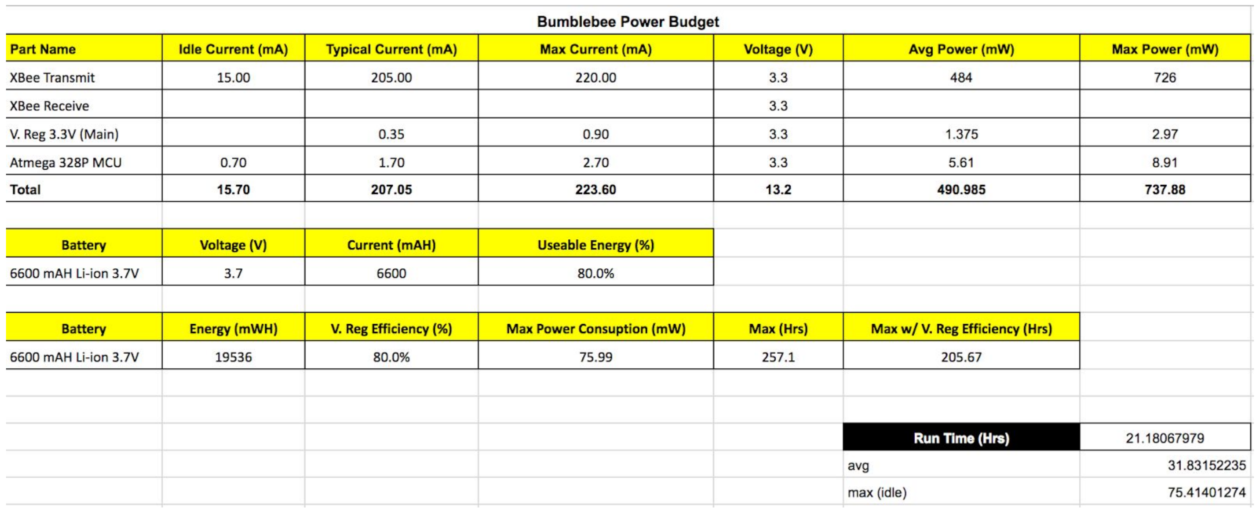


		Signal Strength					Packets				
Distance (ft)	Local		Remote		Sent	Received	Tx Errors	Packets Lost	Percentage	Other Variables:	Date
30	-40		-41		25	25	0	0	100%	Outside. Holmes hall 4th floor. Weather clear, windy. Not consistent sigr	4/4/17
60	-46		-48		25	25	0	0	100%	Outside. Holmes hall 4th floor. Weather clear, windy. Line of sight	4/4/17
90	-45		-50		25	25	0	0	100%	Outside. Holmes hall 4th floor. Weather clear, windy. Line of sight	4/4/17
120	-51		-52		25	25	0	0	100%	Outside. Holmes hall 4th floor. Weather clear, windy. Line of sight	4/4/17
150	-47		-50		25	25	0	0	100%	Outside. Holmes hall 4th floor. Weather clear, windy. Line of sight	4/4/17
180	-60		-63		25	25	0	0	100%	Outside. Holmes hall 4th floor. Weather clear, windy. Line of sight	4/4/17
210	-54		-58		25	24	0	1	96%	Outside. Holmes hall 4th floor. Weather clear, windy. Line of sight	4/4/17
240	-65		-65		25	25	0	0	100%	Outside. Holmes hall 4th floor. Weather clear, windy. Line of sight	4/4/17
270	-67		-71		25	25	0	0	100%	Outside. Holmes hall 4th floor. Weather clear, windy. Line of sight	4/4/17
300	-68		-71		25	25	0	0	100%	Outside. Holmes hall 4th floor. Weather clear, windy. Line of sight	4/4/17
330	-62		-65		25	25	0	0	100%	Outside. Holmes hall 4th floor. Weather clear, windy. Line of sight	4/4/17
360	-65		-66		25	24	0	1	96%	Outside. Holmes hall 4th floor. Weather clear, windy. Line of sight	4/4/17
390	-63		-65		25	25	0	0	100%	Outside. Holmes hall 4th floor. Weather clear, windy. Line of sight	4/4/17
							Through Walls				
		Signal Strength					Packets				
Distance (ft)	Local (start)	Local (end)	Remote (start)	Reomote (end)	Sent	Received	Tx Errors	Packets Lost	Percentage	Other Variables:	Date
72	-72	-76	-75	-69	25	18	1	6	72%	Not line sight. Through building	4/6/17
151	-72	-66	-74	-69	25	13	0	12	52%		
253	-90	-91	-90	-92	25	8	17	0	32%		
332	-89	-89	-89	-89	25	2	23	0	8%		
404	0	0	0	0	25	0	25	0	0%		
							Through foliage				
Distance (ft)	Local (start)	Local (end)	Remote (start)	Remote (end)	Sent	Received	Tx Errors	Packets Lost	Percentage	Other Variables:	Date
64	-45	-58	-46	-60	25	25	0	0	100%	Through foliage (by IEEE)	
							Floors				
Distance (ft)	Local (start)	Local (end)	Remote (start)	Remote (end)	Sent	Received	Tx Errors	Packets Lost	Percentage	Other Variables:	Date
14	-61	-68	-63	-69	25	25	0	0	100%	4th to 3rd floor	
28	-75	-77	-76	-78	25	25	0	0	100%	4th to 2nd floor	
56	-76	-63	-77	-66	25	25	0	0	100%	4th to ground floor	

Table 1: Xbee Range Test

2.9 Power Budget

As this project is a relay communication node, we will not include sensors to test for temperature, humidity, and pressure as the other nodes do. This significantly lowers our power budget as most of the power drain will be caused by the Atmega328P microprocessor and the 3.3V voltage regulator. As you may see in Table 2 below, the maximum power drain is about 738mW while the average power drain is about 491mW. By taking the battery details into consideration, we find that our minimum run time (with maximum power drain) is nearly one full day at about 21 hours, while the maximum run time, assuming the Xbee is idle for the entire time, is estimated at 75 hours of run time which is a little more than three full days. Using the average power consumption, which averages the current draw of the Xbee when it is idle and transmitting, the run time was calculated to be about 32 hours.



Bumblebee Power Budget

Part Name	Idle Current (mA)	Typical Current (mA)	Max Current (mA)	Voltage (V)	Avg Power (mW)	Max Power (mW)
XBee Transmit	15.00	205.00	220.00	3.3	484	726
XBee Receive				3.3		
V. Reg 3.3V (Main)		0.35	0.90	3.3	1.375	2.97
Atmega 328P MCU	0.70	1.70	2.70	3.3	5.61	8.91
Total	**15.70**	**207.05**	**223.60**	**13.2**	**490.985**	**737.88**

Battery	Voltage (V)	Current (mAH)	Useable Energy (%)
6600 mAH Li-ion 3.7V	3.7	6600	80.0%

Battery	Energy (mWH)	V. Reg Efficiency (%)	Max Power Consuption (mW)	Max (Hrs)	Max w/ V. Reg Efficiency (Hrs)
6600 mAH Li-ion 3.7V	19536	80.0%	75.99	257.1	205.67

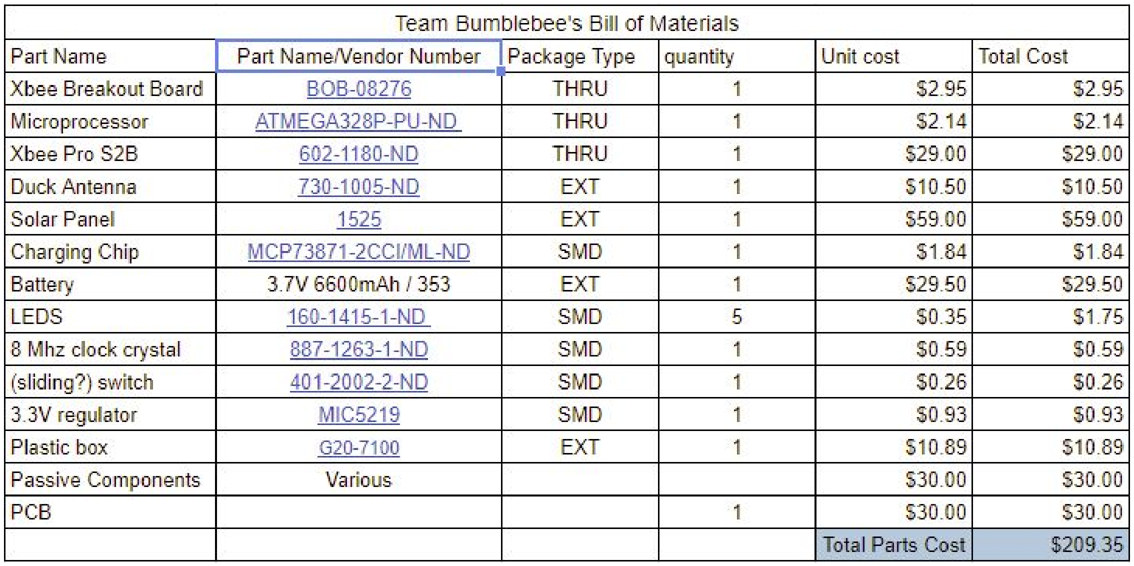
Run Time (Hrs)	21.18067979
avg	31.83152235
max (idle)	75.41401274

Table 2: Power Budget

After completely populating and programming the PCB, the board was left on in the lab with a full battery in order to test run time. The board was programmed to relay a test packet at five second intervals to the gateway server. By comparing the time stamp on the first packet received from Bumblebee by the gateway server with the time stamp on the last packet received from Bumblebee by the gateway server, it was computed that the board was able to last two full days on a full battery charge. The reason this is significantly longer than the theoretical run time is the theoretical run time assumes that the Xbee is always transmitting when in reality, the Xbee is set to transmit on five second intervals, thus drastically reducing power consumption.

2.10 Bill of Materials

Including the cost of all of the components, the manufacturing cost of the PCB, and the cost of the plastic box used for the housing, the total price of the Bumblebee box comes out to $209.35. A full list of the parts used and the price for each part can be found in Table 3.



Team Bumblebee's Bill of Materials

Part Name	Part Name/Vendor Number	Package Type	quantity	Unit cost	Total Cost
Xbee Breakout Board	BOB-08276	THRU	1	$2.95	$2.95
Microprocessor	ATMEGA328P-PU-ND	THRU	1	$2.14	$2.14
Xbee Pro S2B	602-1180-ND	THRU	1	$29.00	$29.00
Duck Antenna	730-1005-ND	EXT	1	$10.50	$10.50
Solar Panel	1525	EXT	1	$59.00	$59.00
Charging Chip	MCP73871-2CCI/ML-ND	SMD	1	$1.84	$1.84
Battery	3.7V 6600mAh / 353	EXT	1	$29.50	$29.50
LEDS	160-1415-1-ND	SMD	5	$0.35	$1.75
8 Mhz clock crystal	887-1263-1-ND	SMD	1	$0.59	$0.59
(sliding?) switch	401-2002-2-ND	SMD	1	$0.26	$0.26
3.3V regulator	MIC5219	SMD	1	$0.93	$0.93
Plastic box	G20-7100	EXT	1	$10.89	$10.89
Passive Components	Various			$30.00	$30.00
PCB			1	$30.00	$30.00
				Total Parts Cost	$209.35

Table 3: Bill of Materials

3 Problems Encountered and Solutions

Firstly it is important to note that although this was the third semester developing Bumblebee, this Spring, two new members were added to the team. When planning our Gantt Chart we were slightly limited as it took some time to explain SCEL as a whole, the goals of Bumblebee, as well as catch the new member up to speed regarding all design procedures and processes. All in all, it took about two weeks (nearly six group sessions) in order to have the member completely and trained. Moving on to the technical issues encountered, the first problem included trying to upload programs to the microcontroller. Because the microcontroller was not taken from an Arduino, and therefore had to be bootloaded, the problem was originally thought to be the bootloader. However, from research it was discovered that the microcontroller needs to be reset during the uploading process to successfully upload. The Arduino board does this automatically each time a program is uploaded. In order to replicate this on the bare arduino the reset button had to be pushed after uploading began. Resetting at the correct time took trial and error and having verbose mode on helped to get the timing correct. After the Preliminary Design Review another team who had also had this problem suggested using a capacitor in place of the reset button to automatically reset the microcontroller. This solution worked for the first few attempts at uploading, but upon further testing proved to be inconsistent. The reset button was replaced and has continued to be used when uploading.

The microcontroller was originally configured to run on 5V with an external 16MHz clock. However, because the Xbee runs on 3.3V, multiple voltage regulators and a 5V boost converter would have been necessary to supply both components with the appropriate voltages. Based on the design of Cranberry it was known that the microcontroller could run on 3.3V, but the microcontroller needed to be reconfigured. In an attempt to run at 3.3V and eliminate the need for the external clock the microcontroller was bootloaded with a program that was supposed to allow both. Although the microcontroller was bootloaded successfully, no program could be uploaded to the microcontroller thereafter. The problem was determined to be the bootloader program. After asking for help it was recommended to not use the internal clock and instead use an 8 MHz external clock and the Arduino Pro (3.3V and 8MHz) bootloader. This was successful. Because it was important for the Atmega to run on 3.3V, the design of the PCB was held off until this problem was resolved. Ultimately, it took so long to resolve this problem that the focus for the remainder of the spring 2017 semester was turned to range and relay testing.

Progress continued and it was possible to send simple string packets to the Xbee connected to the bare arduino and have it forward the packets to another Xbee. When it was attempted to relay sensor node test packets, the relay Xbee, which was connected to the bare arduino circuit, was unable to correctly receive or read the packets and could therefore not forward them. At first the problem was thought to be the code because changes had been made to send and receive the sensor node test packets. Original versions of the code, which only sent simple string packets, were eventually found and tested. The relay Xbee still did not work. Next, the voltages of the circuit were tested. It was found that the LM3940 3.3 voltage regulator was only outputting 2.7V. Looking at datasheet for the voltage regulator, it was observed that the capacitor values being used were incorrect. However, the required capacitance values were not available, so the LM3940 was switched out with the LM1086 3.3 voltage regulator. Following this replacement 3.3V was being supplied to the board. Relay tests were also then successful. This also served as a lesson to upload any code to Github, so that all changes to the code could be tracked.

With a working prototype of the Bumblebee circuit focus was shifted to designing the PCB. The greatest problem for the Fall 2017 semester was learning EAGLE and overcoming problems associated with designing a PCB. When converting the breadboard design to a PCB design considerations had to be made as to what parts would now be surface mounted instead of through hole and how to account for differences in pinouts or design. Additional capacitors were added as decoupling capacitors for the microcontroller and Xbee. When designing the schematic and board layout in EAGLE the correct packages for the different components was needed. The first problem encountered was that it was difficult to find all the packages. Sometimes even after a package was found the reliability of the package was questioned. Searching for the correct packages also slowed down the design process of the PCB. After talking to team Cranberry about the difficulties, Cranberry shared their EAGLE library with all the correct packages. With the correct packages the schematic could be completed.

Once the schematic was completed the board layout could begin. When designing a PCB there are many practices that should be followed that as first timers were unknown to the team. Asking for help from others more experienced with PCB design helped to solve potential problems with the PCB design. These checks from others will hopefully result in a successful board when it is fabricated and populated. The delay in ordering the PCBs was an unfortunate problem that caused the board not to be fabricated or populated by the end of Fall 2017.

At the start of the Spring 2018 semester, the PCB was ordered and could then be populated. During the bootloading process it was realized that connections were needed from the ISP programming pins on the Atmega328P TQFP chip to an Arduino Uno. In order to create connections thin wrapping wire was soldered to the required pin leads of the Atmega328P TQFP chip. Then a thicker wire was soldered to the wrapping wire and connected to the Arduino Uno. These connections were not the best, so multiple attempts at connecting the wires and trying to bootload the board had to be made before bootloading was successful. An important improvement to make on the next iteration is to include a header for the ISP programming pins so that wires do not have to be soldered onto the leads of the microcontroller.

The next problem arose when trying to program the microcontroller. As mentioned earlier the Arduino boards will automatically reset in order to upload a program. The earlier fix on the breadboard was to use a reset button. However, this was not effective on the PCB. Instead the reset pin on the microcontroller had to be connected to the FTDI through a 0.1uF capacitor on a breadboard. Unlike earlier the capacitor was effective in uploading the program to the microcontroller.

Once the program was uploaded successfully transmission and receival of a data packet was tested. At first the board could neither send or receive data packets. Upon observation of an LED connected to the Xbee sleep pin blinking, it was considered that the problem may be that the Xbee was not staying awake. After using XCTU to configure the Xbee to never sleep the system was tested again, but still did not work. After checking voltages and connections to the Xbee it was discovered that the pin of the header connected to ground of the Xbee had a bad connection. After resoldering the pin of the header the board was able to send, receive, and relay data packets.

With the board able to relay packets the relay node was deployed with Cranberry in a test deployment to the roof of Andrew's house. However, once the relay node was placed in the sun with the solar panel connected the relay node would stop transmitting and produce a high pitched whining sound. At first the problem was thought to be a current problem. In the original design of Cranberry, which Bumblebee is based off of, there were two 3.3 V regulators, one for the Xbee and one for everything else. However, because Bumblebee does not have any sensors it was thought that there should be enough current from one 0.5 A 3.3 V regulator for the whole board. While considering how to include a second voltage regulator onto the already fabricated PCB tests with the solar panel with the solar light were conducted. The whining sound was theorized to be from inconsistent voltage being fed to board from the solar panel causing a capacitor to oscillate and produce the sound. After further testing it was found that the barrel jack connector on the solar panel that Bumblebee was deployed with had a faulty connection. Once the solar panel was replaced the relay node began to work as expected under the solar light.

4 Future Work

Although it is now in its third semester, there is still much that can be done to improve the quality, reliability, and efficiency of the Bumblebee sensor node. The first and perhaps most important change to make would be to update the PCB design. Most of the problems encountered this semester were caused by the omission of ISP Programming Pins and the 0.1uF capacitor connecting the reset pin on the microcontroller to the FTDI. The next iteration of the Bumblebee PCB design should include both of these elements to accomplish bootloading and programming the microcontroller through the FTDI, rather than by soldering wires onto the pins of the microcontroller. Additionally, the layout of parts on the PCB can also be updated to make better use of space and reduce overall size. Bumblebee will also soon require a personalized housing. The set-up used this semester was not optimal, and was created at the last minute out of necessity. A custom housing will ensure that the PCB is waterproofed and tailored to meet the mounting requirements to be deployed on the roof of Holmes Hall. Ultimately, the long term goal for the future of Bumblebee is to be able to develop a multiplexing scheme that allows a single node to receive data from multiple weatherboxes. This would most likely be accomplished through time division multiplexing, which would stagger the reception of packets from deployed weatherboxes and the relaying of those packets to the gateway. This idea has yet to be attacked, but is definitely an important step to make for the future of Bumblebee.

4.1 Building a Larger and More Robust Sensor Network

Another future goal for team Bumblebee will be working on creating a larger network mesh for the sensor nodes. As of right now each data collecting sensor nodes send their data packet straight to the gateway Xbee. With the continuation of development of data collecting sensor nodes and the addition of Bumblebee, it will hopefully be possible to expand the network. This goal also includes potentially allowing data collecting sensor nodes to relay packets and the sensor nodes being able to find the shortest routes to the gateway. The overall goal for SCEL is to get sensor nodes on roofs around the whole UHM campus. With the range capabilities of the Xbee this goal is only possible with the use of relay nodes. This can be done in multiple different ways. The first way would be using one Bumblebee node in between one data sensor node and the gateway node. This is an effective way at the moment because we have this process already tested and working with our prototype. The next way would be to use one Bumblebee node to relay data for multiple sensor nodes. We have started testing the capabilities of this method this semester as described earlier in this paper. More work can be done by testing different methods of relaying data from multiple sources. At the moment, there are a few errors that occur when just offsetting the sending times of the sensor nodes. The gateway can receive the different packets; however, the gateway sometimes get an invalid schema error. A possible solution to this could be figuring out if the relay node can handle the data using some sort of queue. A queue will help in the fact that the relay node will be able to keep receiving data from the different nodes while sending out the data it has already received.

There is a more advanced approach to the sensor node network that will take a lot of research but will be worth it. This approach involves a little bit of networking knowledge. Routers and switches in a network send out a broadcast signal to find out who is on the network and keeps track of how many hops it takes to get to a certain endpoint. With the use of certain internet protocols, the network can determine the shortest path for data to travel. We believe something like this could be implemented into SCEL's sensor node network. To make this possible and the network more robust, the sensor nodes themselves should be able to relay data packets as well as collect and send their own data. This might be easily implemented because of the way the code works for the relay node. The relay code simple waits to receive a data packet, copies the contents, repackages the data into a new packet, and sends it to the next address. The sensor nodes should be able to handle this task.

4.2 More Range Testing

For future range testing, longer distances for line of sight should be tested. Also, there are a variety of variables that could be tested. Some variables that have been identified for possible future testing are weather, through different materials, elevation, and location. Other types of antennas could also be tested to see if there any benefits to using different antennas or different frequencies for communication. More extensive range testing will ensure that optimal communication is happening within the sensor node network. The data collected during these test will help with determining the best locations for the sensor nodes and the relay nodes.

5 Engineering Standards

Our environmental sensor network node was based off of environmental and sustainable engineering standards while accounting for practical constraints. To restate the Smart Campus

Energy Lab mission: we foremost seek to forecast weather patterns pertaining to the University so that we may be able to evaluate the best places on campus to place renewable, power-generating resources. The relay nodes described in this project look to extend the range of such forecasting nodes and all the while considering our environmental impact not only through our calculated end goal, but also through each design choice. According to our Bill of Materials (see Table 3), our low cost makes the manufacturing of each relay node quite plausible. Additionally, with the use of 3D printing and a solar, self-sustaining circuit, we look to constrain our costs even further. Though we do not believe the designing and fabrication of this project violates any ethical codes, we do see the possibility of doing so if the University fails to share such information (after completion) to students and stockholders, understandably decreasing tuition and fees through the decreased need for external power. We also believe that we it would not be right to produce a product that did not support the goals of the Smart Campus Energy Lab. Therefore, we made sure to have our board designed to sustain itself without relying on traditional power methods meaning that we plan to attach two solar panels (connected in series) to feed into our battery. As talked about in our design portion of the paper, we have also sized down our voltage use for a greener and more practical purpose. Since we plan to deploy these boxes on rooftops around campus, it will be agreeable to have a circuit that does not require much from the battery. This design also answers the question on what to do if the weather does not generate enough solar energy, most likely during winter and rainy seasons. By including the voltage regulator stepping down 5V to 3.3V, the circuit saves power for itself and slims down required maintenance. In regards to health and safety, the Smart Campus Energy Lab is currently pushing most of its focus towards solar energy. Though wind energy seems ideal for Hawaii, due to rational health and safety regulations, we are not allowed to place moving parts upon the rooftops at the University. This is why our focus has shifted towards solar panels rather than wind turbines. However, after future weather forecasting, we do hope to find a safe alternative to incorporating wind energy into the system. It is also important to note that through testing and deployment, students in the lab do not have access to rooftops except when planned, coordinated, and accompanied by Dr. Garmire, a fellow University professor. Furthermore, we believe that this project holds a positive social impact. Additionally, Bumblebee's design is based off of another team's design and each group in the lab uses similar parts including the Xbee and Atmega. By incorporating parts that others use as well, everyone in lab has some understanding of how each circuit is laid out to work, which is another health and safety feature. As mentioned beforehand, the Smart Campus Energy Lab keeps in line with Hawaii's Renewable Portfolio Standard (RPS) of reaching 100% by 2045. Statewide, it encourages students and others in the community to accept renewable energy and commit to relying on it. Moreover, by running towards the RPS goal, Hawaii is setting a standard worldwide as all eyes are focused towards the capability and strategies to attain such goal. Economically and politically (in the long run), we expect to see Hawaii's dependence on foreign oil drop and have currency remain circulating within the state.

6 Conclusion

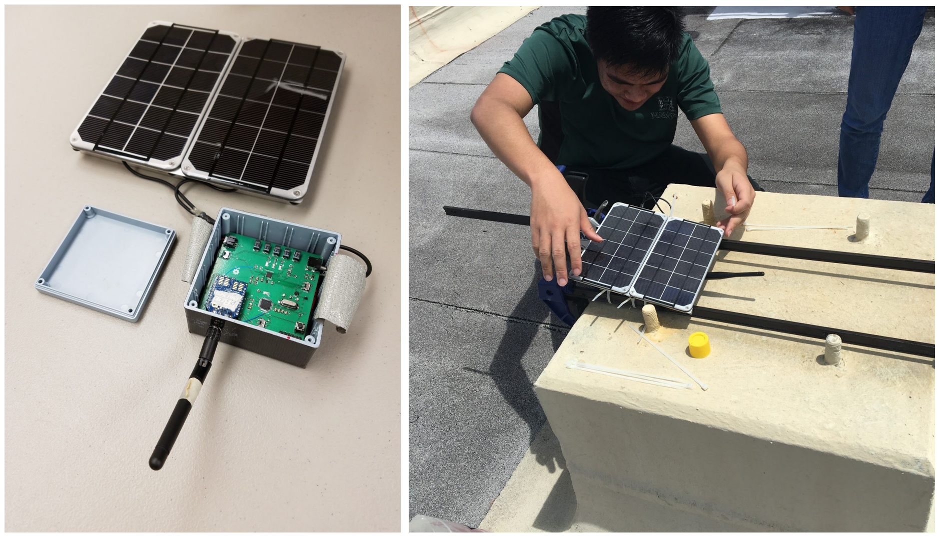


Figure 12 Final Product and Deployment

At the beginning of the Spring 2018 semester, it was anticipated that the route to deploying Bumblebee would be relatively straightforward. However, after populating our PCB, it was quickly realized that this was not the case. With missing components and no way of bootloading and programming the microcontroller, a workaround was needed very quickly. In the process of fixing these issues, more had arisen, such as the inability to obtain a custom housing, the tempermentalness of multiple XBees, and an improperly functioning solar panel. However, through the collective efforts of all members and help from former members, solutions were found and two deployable Bumblebees were produced. One currently sits on the roof of Holmes Hall capable of relaying data to the gateway in the lab, and the other is waiting for a functional weatherbox to be paired with.

In terms of packet relay ability, the Bumblebee prototype is able to receive a sensor node packet, read it, repackage it, and send it to the gateway. As of right now the data gathering sensor node is hardcoded to send directly to the address of the relay, and the relay is hardcoded to directly send to the address of the gateway. It would be ideal if every sensor node could dynamically find the shortest path to the gateway. Also, Bumblebee has only been tested to handle relaying from one source at a time. In the future more research and testing will need to be done to accommodate more sensor node nodes into the network. Some basic range tests for line of sight and non line of sight was conducted. For line of sight there were no errors for the length of Holmes Hall. However, with just a short distance obstacles caused a large amount of errors and a drop in signal strength. In the future, more thorough range testing should be conducted. Even other variables not being tested such as time of day and temperature should be recorded.

This project has been a great learning experience. We've gained more knowledge about PCB design, networking, and skills that have we have not learned in any class, such as soldering. In addition, this project has allowed us to apply topics learned from our courses. The firmware for this project uses Arduino and its libraries, which are based on C and object oriented C++. EE160 introduces students to C and EE205 introduces students to object oriented programming. These courses have helped in understanding the firmware already being used for the sensor nodes and how to make changes to achieve a relay node. For example, the packets of data that sensor nodes send are made with C++ and are constructed as structures, different data grouped together under one name. With the understanding of how packets were constructed, the firmware for the relay node was modified to read and then repackage the packet to be sent to the gateway. Another course related to this project is EE 438, renewable energy. EE 438 teaches the fundamentals of power, electric power grids,wind and solar power systems, and photovoltaic materials and systems. This course is especially relevant to the work of this project because of the incorporation of solar panels into the relay node's design.

References

1. B. Amano and K.P. Castro. “WIP: Wireless Environmental Sensor node Generation 3” University of Hawaiʻi at Mānoa. Dec. 2015, revised May 2016.
2. Gammon, Nick “How to make an Arduino-compatible minimal board” May 2012, http://www.gammon.com.au/forum/?id=11637
3. Rapp, Andrew “Arduino library for communicating with Xbee radios in API mode” Dec. 2016, https://github.com/andrewrapp/Xbee-arduino
4. S. Saepoo. “Self Sufficient Routing node for Mesh Sensor Network” University of Hawai ʻi at Mānoa. Dec. 2016.
5. “Securing the Renewable Future” Jan. 26, http://energy.hawaii.gov/renewable-energy
6. “Xbee / Xbee-PRO ZigBee RF nodes” User Guide, Digi International Inc., Apr. 2008, revised July 2016, http://www.digi.com/resources/documentation/digidocs/PDFs /90000976.pdf.