EE 496 Final Paper for Fall 2021
2nd Generation Relay Node: Team Bumblebee

Authors: Brian Griswold, Yin Nyein Aye, Thant Thiri

Department of Electrical Engineering

University of Hawaii at Manoa

Renewable Energy and Island Sustainability Program

Smart Campus Energy Lab

20 December 2021

Abstract: ‘Bumblebee’ is Smart Campus Energy Lab’s (SCEL) 2nd generation communications module designed to relay meteorological data collected from weather boxes on Holmes Hall. The data that is collected from the other hardware teams’ weather boxes is sent wirelessly using an XBee or the relay module to the gateway computer in Holmes Hall. The ‘Bumblebee’ module was developed to extend the range of the weather boxes to send packets to the gateway computer.

Table of Contents

I. Introduction

In 2008, the state of Hawaii and the Department of Energy decided to reduce the dependence on imported fossil fuels. The goal of the Hawaii Clean Energy Initiative was to become 100 percent clean energy by 2045[1]. Smart Campus Energy Lab (SCEL) was created under the University of Hawaii's Renewable Energy and Island Sustainability (REIS) program to achieve the goal of becoming 100 percent clean energy by 2045. The REIS program's goal is to help UH Manoa to run on its own microgrid that is powered by 100 percent renewable energy.

SCEL's goal is to track and analyze meteorological data such as temperature, humidity, and solar irradiance by designing, building, and deploying weather boxes that can collect and send meteorological data to help UH Manoa run on a microgrid. The meteorological data collected from the weather boxes will help UH Manoa use less energy and help forecasting algorithms[2]. An example of this is by increasing or decreasing temperatures in classrooms appropriate to outside temperatures. These weather boxes are designed and developed to be low-cost in order to be mass-produced. An example of this is by increasing or decreasing temperatures in classrooms appropriate to outside temperatures.

The Bumblebee does not have any sensors; therefore, it does not collect any data but it helps to extend the distance weather boxes can be from the gateway computer. The ideal location of the Bumblebee boxes would be between the distance sensors and the lab gateway. The current hardware of Bumblebee is based on the third generation of sensor nodes and it uses the same components including the Atmega328P microcontroller, the XBee Pro S2C, and solar charging circuit. In order to verify that our Bumblebee module is functioning correctly, the Bumblebee team also conducts range testing and networking with the XBees.

II. Relay Node: Bumblebee Overview

Block Diagrams

Figure 1 shows the block diagram of our signal/communication. It shows how the different nodes communicate wirelessly. It shows the basic path of the packet traveling from the sensor to the gateway computer. This path shows the data being collected by sensors on weather boxes. The weather boxes using the XBee S2C Pro construct the data packets and send them to the Bumblebee relay node. Bumblebee will receive that data and send it to the gateway XBee to send to the lab gateway. Multiple relay nodes can be used in order to send packets to the gateway. Once transmitted to the gateway, data can be displayed and analyzed by the forecasting team. The team hopes to extend the range of the network throughout the UH Manoa campus.

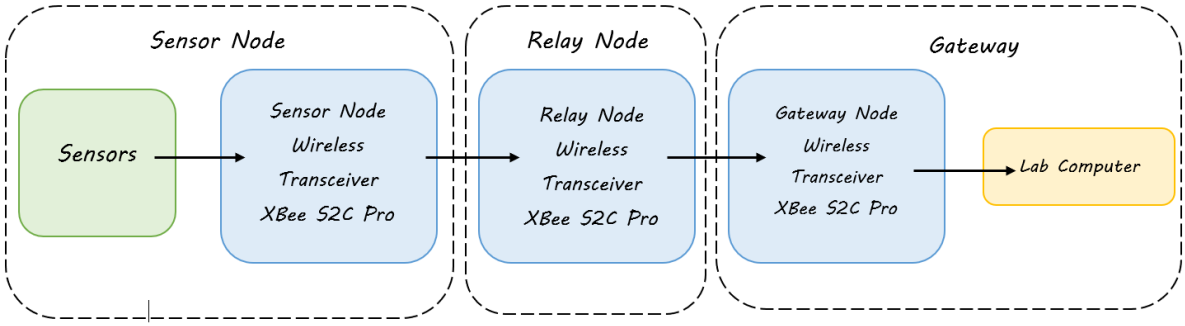


Figure 1. Signal/Communication Block Diagram

Figure 2 shows a block diagram of each hardware component on the Bumblebee PCB board. To power the Bumblebee relay node, the charging circuit is connected to a 3.7V battery that is recharged by a solar panel. The battery is then connected to the 5V boost converter to power the Atmega328P run at 5V with a 16MHz clock and XBee Pro S2C. The Atmega328P

controls the XBee Pro S2C and allows it to function properly and flashes various LEDs on the PCB to tell if packets are sending. The Atmega328P is boot-loaded and programmed using the ISP Programming pins. From the 5V regulator, we step the voltage down to 3.3V to operate the XBee Pro S2C.

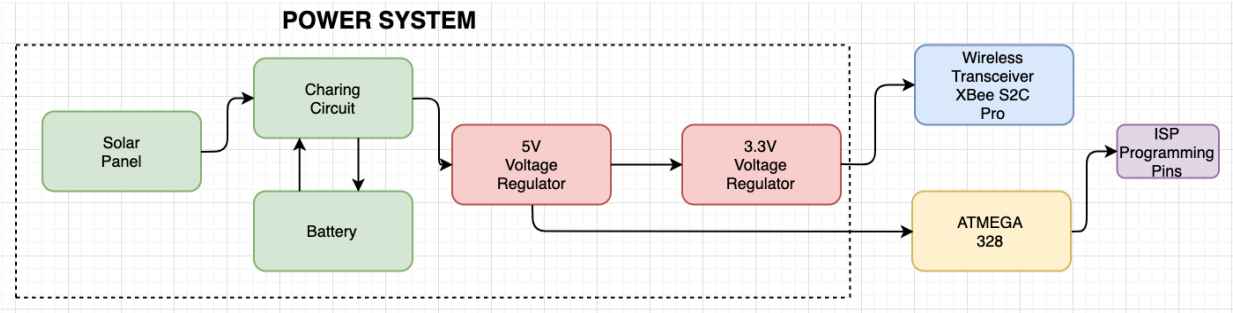


Figure 2. Power Block Diagram

Designs: Schematic and PCB Layouts

The main design for the Bumblebee module has been improved from many semesters of development. Originally, the relay node was based on the Cranberry weather box except with fewer components needed; this is because no sensors were needed for our Bumblee relay node. Hence, the design has been modified in order to fix the connection mistakes. The main design challenge this semester was how to design our board using EAGLE software since the old member of the team did not have any experience building PCB boards. Last semester, the Bumblebee team only focused on bootloading the PCB Board, range testing, and finding the connection mistakes. However, all the members in our team watched videos on designing the PCB board using EAGLE software. The version 4.1 design, as shown in Figure 3 uses the 5V SMD voltage regulator. However, one of the connection issues with this board is that Feedback

was not connected to the V_{out} or output voltage pin. V4.2 reflected this design as shown in Figure 4. We fixed the connection issue on our new board by connecting FB to V_{out} . We increased the analog and digital filtering of the XBee 3.3V power supply for better signal quality. We also improved the decoupling of the Atmega 328p at a 5V power supply to improve the range and the performance of the board. On this version 4.2 board, we labeled all the components to make it easier to solder and this version is slightly bigger than version 4.1. We spent two weeks designing the PCB board and we met with our advisor to make sure all the connections were correct. It also took about two weeks to receive our PCB board. We also spent a week soldering our first version 4.2 PCB board to bootload.

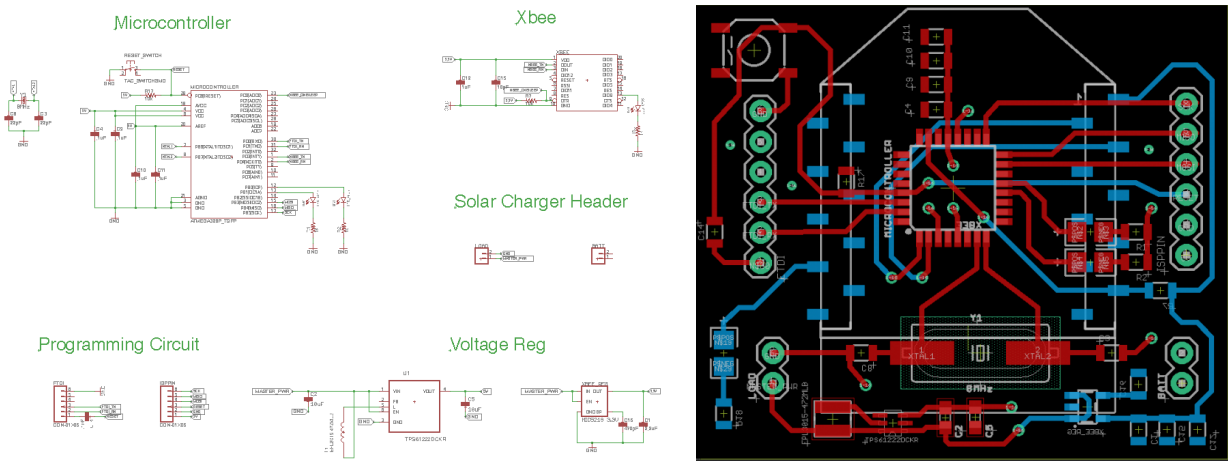


Figure 3. Schematic and PCB Layout of Ver. 4.1

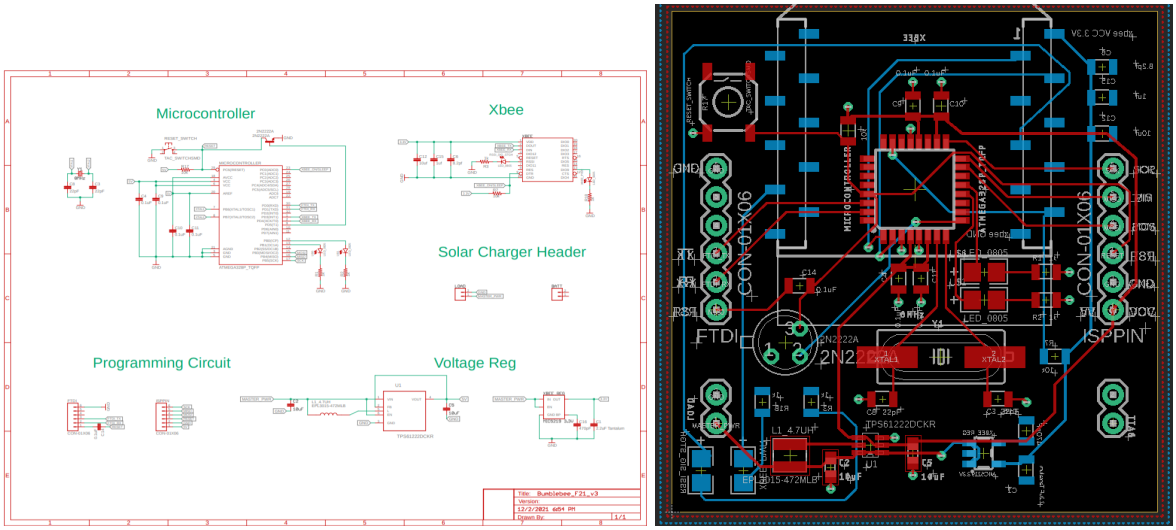


Figure 4. Schematic and PCB Layout Ver. 4.2

Bare Arduino Board

The bare Arduino board which is also known as the bare Bumblebee is used to replicate the circuit that will be on v4.1 and v4.2. The main purpose of the bare Bumblebee is to see how the circuit will work before soldering components to the PCB. In Figure 5, the bare Bumblebee replicates the circuit that will be used for v4.0. This semester, new members of team Bumblebee replicated the v4.1 circuit by reconstructing the board. Furthermore, the bare Bumblebee was used to test bootloading procedures of the Atmega328p to troubleshoot the same process onto the PCB board. This semester new team member has learned how to bootload and they successfully boot-loaded the Atmega328p on the bare Bumblebee.

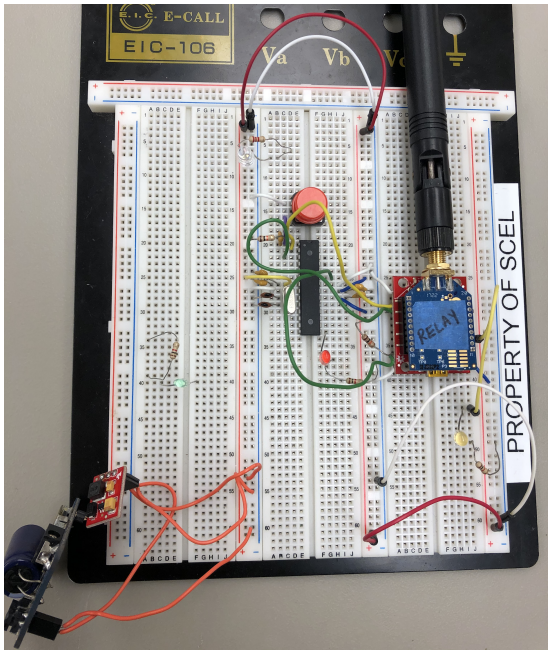


Figure 5. Bare Bumblebee

Packet Range Testing

The XCTU and Arduino IDE programs are used to test the abilities of our XBee in transmitting and receiving packets. Range testing is used by the team in order to see signal strength at varying distances between XBees. XCTU is a free multi-platform application designed to allow developers to interact with Digi RF modules. The XCTU allows one to easily set up, configure, and test XBee modules. With that being said, this program is one of our main tools in testing our XBees' ability. This semester, we compared our range testing results using version 4.0 and version 4.1. We used only two Xbees: one will be the coordinator and one will be the router. The purpose of the routers is to act as the sending-end, or sensory node, that can transmit packets to other XBees. While the coordinator is to act as the gateway which can only transmit to itself.

Before jumping into range testing, we want to make sure that the configuration of each XBee is correct and that they are able to recognize each other. Starting with the configuration, we want to plug all three XBees into a computer and confirm the communication of each XBee. If needed, we would want to update the firmware of each module to the function set DigiMesh 2.4 TH PRO and the firmware version to the newest value. Once the firmware has been updated, one must click to the default setting first and then make any necessary changes if needed. In this case, our team decided to change the API setting for each XBee. The coordinator API setting was to be enabled while the routers API settings were left in transparent mode. We also checked and confirmed that the PAN ID for each XBee is the same. This is because the PAN ID, also known as the destination address, allows the sending XBees to transmit to another which has the same address.

Field Range Testing

Once the communication of XBees has been established, we can start conducting range testing. The main purpose of range testing is to record the signal strength and packet loss while taking into account as many variables as possible, such as obstacles and weather. Eventually, it helps us to determine the maximum distance of XBee communication. To conduct this testing, we use the XCTU program. The data values that we will be looking at are local strength, remote strength, packets sent and received, TX errors, packets lost and percentage of packets received. By determining those data values, we can conclude how the signals would behave with different variables and how strong the signals are. By using those data values, we can determine what variables will affect the XBee that would be more likely to interrupt the signal. For example,

when students are passing between the two XBee modules, we can see that our results will show some disruption in our signal strength and could result in packet loss.

Set-up

When setting up to do our range testing, we will be conducting our testing in two different settings. The first setting that the team did was a line of sight range testing. Using the hallway of Holmes Hall, we were able to make a clear pathway that goes from one end of the hallway to the other. Figure 6 shows a picture of our line of sight setup. This testing helps us see how good our signal is between our local and remote XBees without any obstructions in the way since there is no wall or tree. Moving the remote XBee every 30 feet while still maintaining a line of sight with the local XBee.

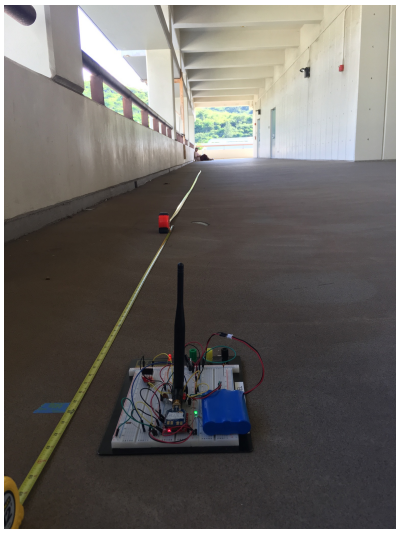


Figure 6. Line of Sight Set-up

The second set-up that we conducted was to include obstacles in between the local and remote XBees. We performed our range testing at McCarthy Mall since there are trees and a lot of people walk around there. For example, trying to see how good the signal is when there is a tree in between the local and remote modules or if they can communicate through walls. From

there, we would increase the distance and the amount of obstruction between the local and remote XBees.

Range (Field) Testing Results

This semester, the team has conducted several different range testing experiments for different scenarios. As we mentioned before, range testing is very important because it allows us to determine the performance of the XBee. With that being said, instead of using our bare Bumblebee for testing, we will be using our version 4.1 PCB board and version 4.2 PCB board that we have populated and programmed to complete our range testing. The type of range-testing we did includes: line-of-sight, not line-of-sight, with or without 8.2pF capacitor on version 4.2.

Starting off the semester, we began range testing using our bare Bumblebee board that replicated the v4.0 board to show our new members. The range testing results lets the team compare the results found with the v4.0 PCB and gives us a baseline of how well the PCB should perform. Figure 7 shows the line of sight range testing results and not line of sight range testing. Similar to previous semesters, the results show a good signal strength where the signal does not drop below -80 dBm for the average line of sight and averages for the not line of sight drops below -80 dBm once for the remote XBee at 400ft. Also, there was no packet loss seen when performing the tests. When it comes to range testing, having a signal under -80 dBm indicates a weak signal and could result in packet loss.

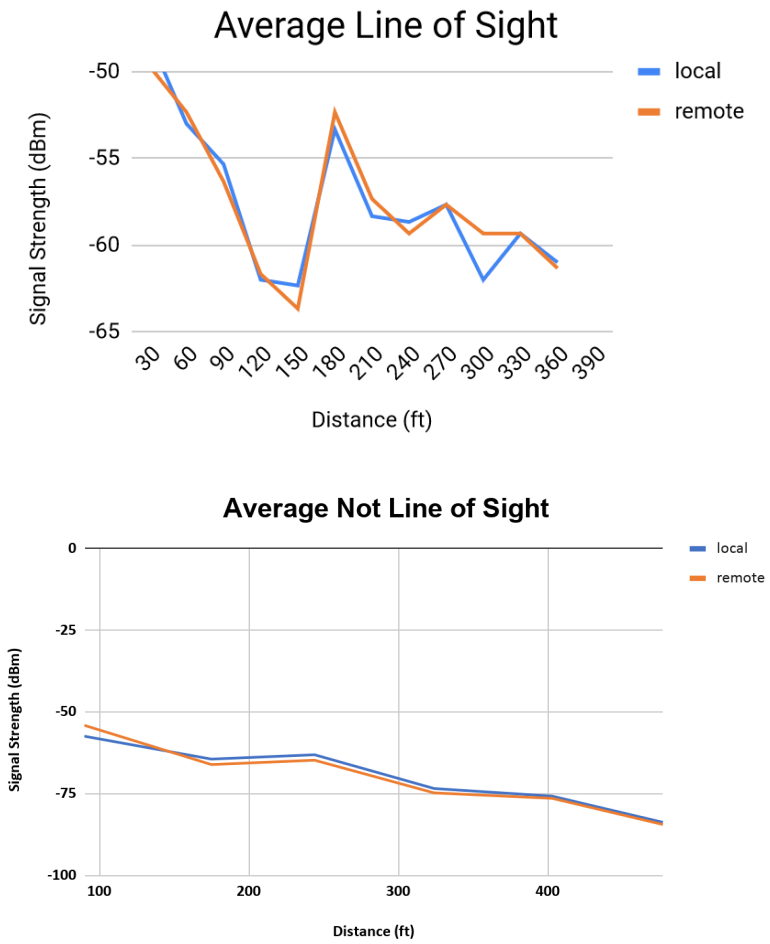
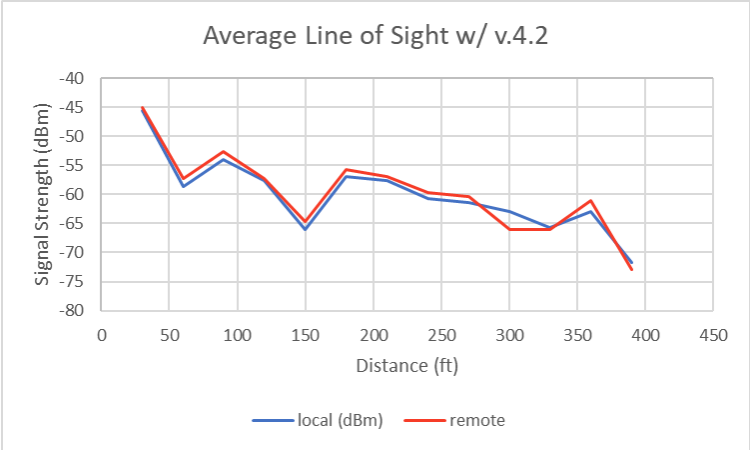


Figure 7. Range Testing Results with Bare Bumblebee

The next set of range testing was done using the version 4.1 and version 4.2 PCB board. After bootloading and programming the v4.2 PCB board, average line of sight, and not line of sight tests were done. In Figure 8, we can compare the results of our version 4.2 with 8.2pF capacitor tests for line of sight and not line of sight. Comparing the two results from Figure 7 and Figure 8, the average line of sight signals at various distances are similar but version 4.2 has a better signal strength than Bare Bumblebee. The same can be said when comparing the average and not line of sight test. There was no package lost on both Bare Bumblebee and our new version 4.2 PCB board.



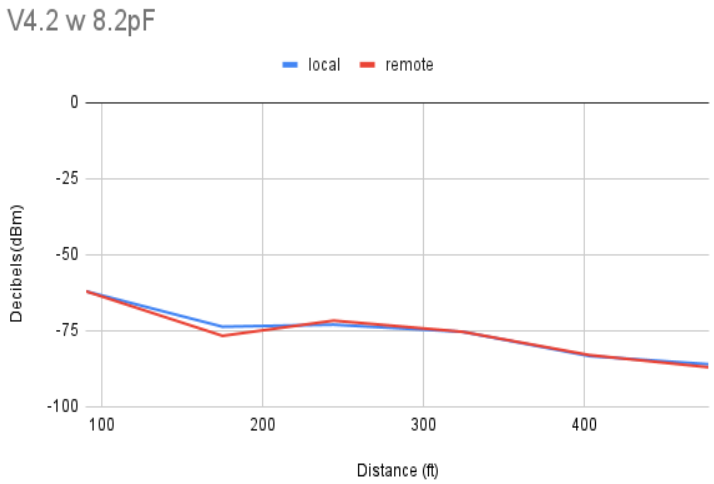


Figure 8. Average Line of Sight and Not Line of Sight Range Testing Results with v4.2

The last test that we performed during the semester was range testing with v4.1 and v4.2 (without the 8.2pf ceramic capacitor). The reason for this test was to see if the v4.2 without the 8.2pf will function the same as the v4.1 with all the components soldered on the PCB board. Figure 9 below shows the test data results for the average not line of sight for both v4.1 and v4.2. Comparing the data using version 4.2 with and without the 8.2 pF capacitor, the team concluded that the 8.2 pf does affect the range testing results. Figure 8 above shows that around 250 ft distance, the v4.2 with the 8.2pf flatten the graph and have the same signal as the remote whereas

Figure 9 below shows around that same distance, the v4.2 without the 8.2 pf peaks. Furthermore, without the 8.2pf, it affects the packets lost as compared to our completely soldered v4.2 PCB board with the 8.2pf, we had no packets lost at the same locations. We also went farther all the way to the Hawaii Hall to do the range testing with v4.2 and concluded that both the local and remote signals were around -88 decibels.

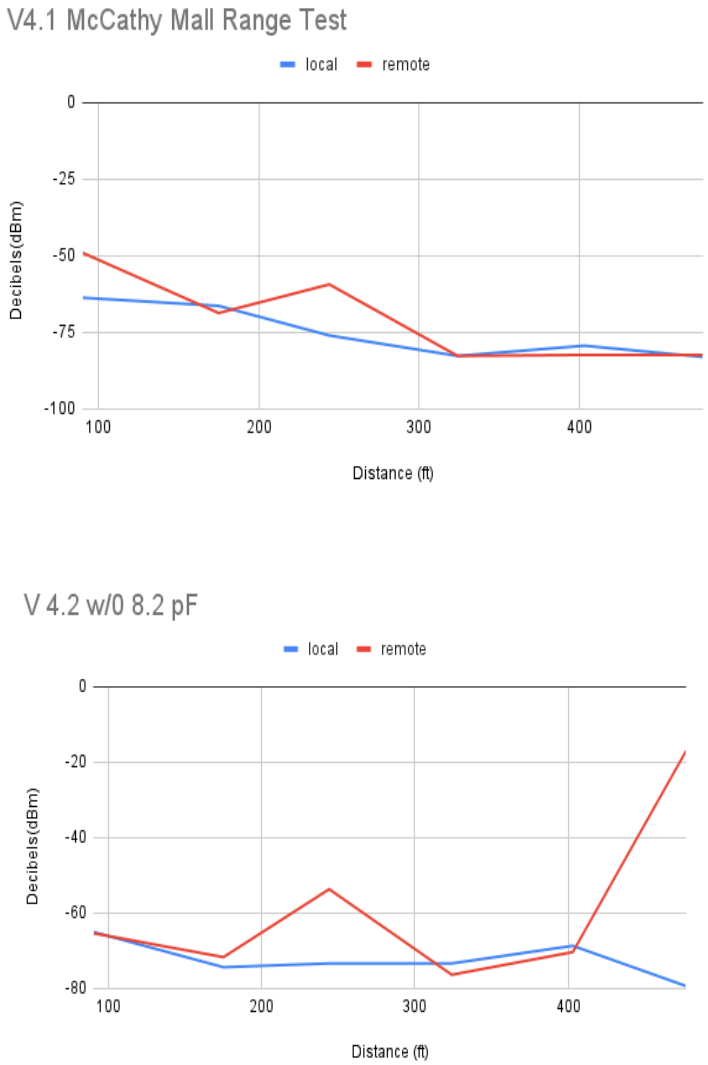


Figure 9. Average Not Line of Sight Range Test Results with v4.1 and v4.2 (w/o 8.2pf)

III. IoT Test Network Creation

While we were working on our 396, Junior Project, team bumblebee was able to attempt and conduct a line of sight range testing while setting up several XBees in a many-to-one routing network. This semester with the XCTU program, we were able to further construct a test Internet Of Things or IoT using Zigbee protocol. This was a many-to-one wireless routing network where several nodes are communicating with our single node that performs the centralized function as the coordinator. Figure 10 shows an example of the many-to-one routing network. How this routing works is that the data collector would send a broadcast transmission of a many-to-one, establishing a reverse route on all devices for those who have received the request. This then reverses the routes on all devices. After other modules have received the request, it then creates a path that goes back to the collector by storing a reverse many-to-one routing table entry. When all XBee devices discover a route to a collector, it will generate a broadcast route discovery message. Based on our research, using several XBees, there is a chance that it can cause the network to be overloaded with a bunch of messages. However, in this configuration, no responses will be generated therefore minimizing the network traffic congestion[3].

Figure 10. Many-to-one network

To create a test IoT there are a few basic but critical steps to set up the individual devices to be discovered by the coordinator. It should be noted that a network should never have TWO coordinators turned on at the same time as this can lead to network errors. The coordinator for each network continuously monitors for any devices that are discoverable to connect to the IoT once XCTU begins to construct the network. How to use XCTU is to create the IoT which will be talked about later in this section. First, how to ensure the devices will be discoverable with proper configuration will be discussed. During the discussion, it is best to note the register identifiers such as (OI) as the description of the different pan id is sometimes similar if not the same but with a different function. However, all registers, don't care don't care are used here as an example (XX) commands will be unique and are the best to follow.

For a device to be discoverable to the coordinator, two critical initial conditions must be met. First, the (OI) Operating 16 bit PAN ID must be set to an initial value of ZERO. This makes the device discoverable to any coordinator. This is a READ ONLY value in the XCTU software. The coordinator scans the environment to check for background power that will interfere with the network's signal strength, after this scan the coordinator assigns the (OI). Once a routing device has been paired with a coordinator, the coordinator assigns this 16bit (OI). The only way to clear the (OI) to make it discoverable to another coordinator is to perform a ***full XBee recovery*** of the device in XCTU.

Second, the Operating Channel (CH) for all router devices that are attempting to be connected to the IoT coordinator must be the same as the coordinator (CH). This channel is also assigned by the coordinator once a device is discovered by the (OI) value being zero upon power-up within range of the target coordinator. The most efficient way to ensure that a router is

going to acquire the same (CH) as the coordinator is to connect via USB to the same computer that the coordinator is powered by when performing the XBee Recovery. Also if possible, do not do it within range of, or power off all other coordinators that are not wanting to be connected to. Once the XBee recovery process completes, a coordinator within range will attempt to have the router join its network if within range. This can lead to some issues in the SCEL lab where more than one coordinator can be active at the same time. The recovered routing device will connect to the first coordinator it discovers; the (OI) and (CH) will be assigned by that coordinator. If it was not the intended coordinator, an XBee recovery would need to be performed again to clear this coordinator assigned read-only values and attempt another configuration of the router to pair it with the intended coordinator.

Being read-only coordinator assigned values the two preceding register values for (OI) and (CH) can be the most challenging to ensure they are the same for all routers when devices are configured between teams and semesters. The other operating pan ID in register (ID) is user-configurable. However, this pan (ID) being set the same as a network, it's intending to connect to without the (OI) and (CH) being correct will create an undiscoverable device and no communication.

Once the (OI) and (CH) are the same as the target coordinator, the user can use the XCTU interface to set the (ID). This writable value (ID) will be set to the operating channel PAN ID in register (OP). (OP) is also a read only variable that must be cleared by a XBee recovery once it is set. Changing the writable (ID) will not change the read-only value in the (OP). The last steps to have the radios fully configured for the desired IoT test is to set all radios to API 2 mode in the (AP) register, both coordinator and routers, and ensure the routers have the “scan channel” or

(SC) register value set to 7FFF in the bitfield. This will put the radio in broadcast mode. The node identifier or (NI) can be set by the user and will help identify the individual routers in the network however it is not necessary to the functionality of the IoT[3],[4].

This semester the Firmware team worked within Visual Studio to make the Xbee configurable without XCTU. This could be an option going forward that can change the permissions of the read only register values to make the Xbee more easily configurable. However, being late in the semester, we were not able to work with the new firmware version. This could be future work. The preceding paragraphs go over in detail what must be done to connect many routers to one coordinator within XCTU; further, many common issues that can be faced and how to address them. After this portion of configuration is complete the last steps to create an IoT test network that is relatively simple. **A sample of an XCTU configuration page is in figure 11 below with the critical fields highlighted that are read-only and set by the coordinator. (ID) is user set but is highlighted here as it sets the read-only (OP). Again, to reset these values, an XBee recovery must be performed in XCTU!**

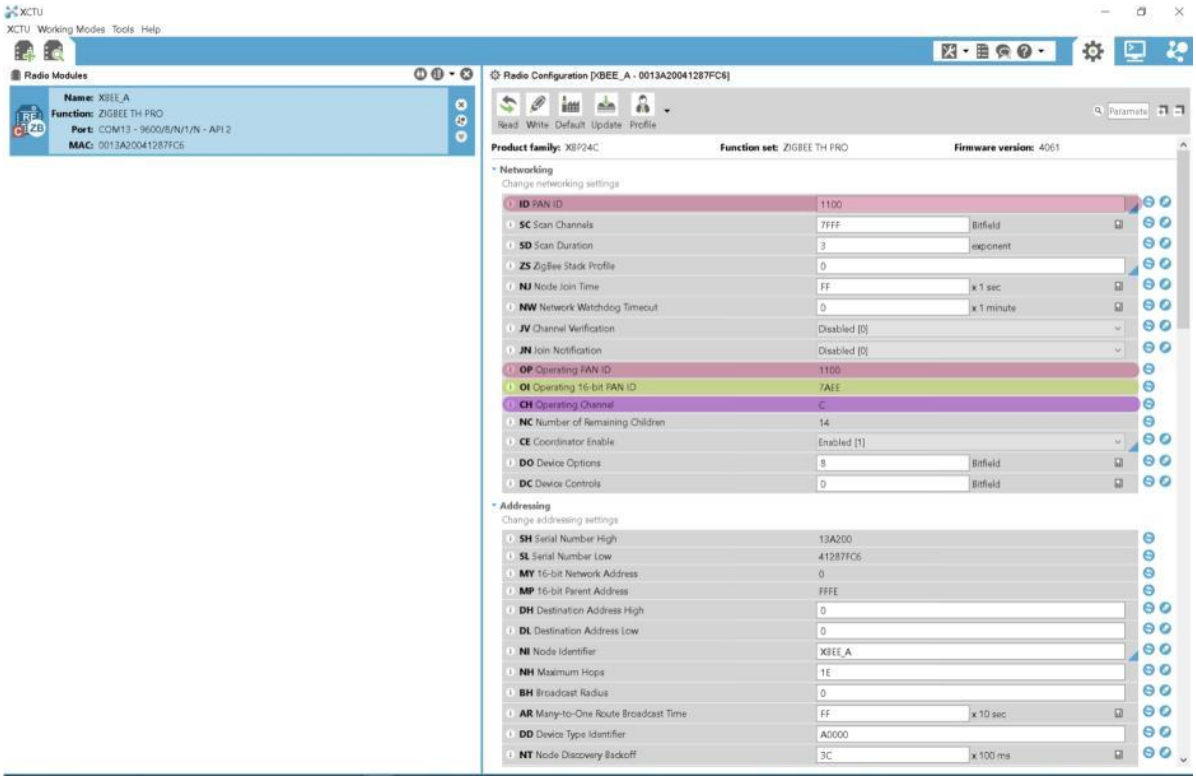


Figure 11: XCTU with the coordinator

To construct a test, IoT assumes the IoT is being constructed from scratch having to match the coordinator and router devices configured as described above. Deploy the coordinator by connecting to a laptop or similar device with XCTU. Discover the coordinating device in the XCTU by selecting the port and adding the device. For the routing devices, they only need to be powered by a stable 3.3 volt power supply to test an IoT spacing and strength. Connection to an Arduino or team PCB is not necessary for this test. Once the router devices are powered and placed in the intended locations for the signal test go to “switch to network working mode” in the top right hand corner of the XCTU software and select it. The software will have a blank white screen now after selecting. Under XBee in the upper left middle of the screen will be a “Scan”; if hovered over it will say scan the radio module network. Select scan and the XCTU software will

scan for all devices configured to the coordinator and connect. Any device powered and within range will discover and add to the XCTU screen where it was blank white.

The software will continuously scan for discoverable routers and test different configurations. A line will be drawn between all detected devices, often more than one as more than one path will be created as this adds to the network's robustness if a router is to go down. A signal strength value of 0 being the lowest and 255 being the highest will be shown between the modules. There is a value on each side that shows the strength in either direction between the two devices. Right-click and hold on a device to drag the icon around and arrange the network as desired to match the deployed network layout.

IV. Problems and Solutions

While working on bumblebee this semester, we ran into some previous and new problems. Due to the covid-19 pandemic still going on, all members still have to follow the health safety guidelines. Therefore, due to only having one team in the lab at a time, this reduced the amount of work we could complete in the lab this semester as well as communicating with other teams.

Besides Covid-19, one of the problems that we had at the beginning of the semester was designing the PCB board on EAGLE software as well as bootloading. We had some issues with bootloading while we were showing our new members how to bootload and program the PCB board, and we concluded that it was because of the capacitor in the reset. While we did more research on that, we found out that there are three scenarios of the capacitor in reset. The first reason is that FTDI is in series on the rest line. FTDI usually goes low and stays low for RST or

reset control and the capacitor allows RST to go high after it charges. The second scenario is that the ISP between the reset and ground on both breadboard and PCB. The ground plane (zero voltage) on the battery power is not well defined and the capacitor keeps reset low for the needed three seconds due to the charging time or just to power the breadboard or PCB from Arduino/ USB 5V or 3.3V as needed. The third scenario is that the Arduino has the Programmer ISP capacitor between reset and ground on the programmer. It is intended to prevent the programmer from resetting during boot loading. Modern Arduino as ISP has dealt with the reset in the sketch and it is not needed anymore.

V. Future Work

Next semester, we plan to populate our version 4.2 since there is no connection error and increase the robustness of the PCB for deployment. We would like to design another version of 4.2 to incorporate FT232 which is known as FTDI, and solar charging circuit on the PCB board. We would also like to incorporate the BMP280 SMD mount with I2C. Since our version 4.2 works well, we want to do range testing with other hardware teams to create a relay network of sensor data to test the gateway. We also want to test our Visual Studio code on our version 4.2 PCB board to verify the communication between the ATmega software serial and the XBee.

VI. Conclusion

This semester, we made a lot of progress as far as redesigning our v4.1 to make v4.2 have the correct voltage step up and step down in EAGLE. We did have a huge learning curve as we were both learning and applying at the same time. However, we were able to overcome and have the v4.2 that’s robust and reliable. We also used the IoT network to further range tests from our

coordinator that was in the SCEL lab and all the other different routers were in different positions especially some were through the concrete walls on the 4th floor and two of the routers were in Hale Manoa and East-West Center. With our current v4.2 design, we can populate more v4.2 adding additional components such as adding a solar charging circuit, incorporating FT232 also known as FTDI to avoid any other pin connection issues that we dealt with during the semester. Overall, the goal of team Bumblebee is to extend the range of our network from our sensor nodes to the gateway computer and deploy our v4.2 PCB Boards on Holmes Hall.

References

[1] Hawaii Clean Energy Initiative. (n.d.). Retrieved December 18, 2021, from http://www.hawaiicleanenergyinitiative.org/.

[2] (n.d.). Retrieved December 18, 2021, from http://scel-hawaii.org/research/.

[3] Wireless Connectivity Kit Getting Started Guide Getting Started Guide. (n.d.). Retrieved December 18, 2021, from https://www.digi.com/resources/documentation/digidocs/pdfs/90001456-13.pdf

[4] Map Your Digi XBee IoT Network with Digi XCTU.(n.d). Retrieved December 18, 2021, from www.digi.com/resources/examples-guides/map-your-digi-xbee-iot-network-with-digi-xct.