Final Project: Remote Temperature Monitoring

EEL4599: Wireless and Mobile Networks

Joshua Restuccia
Department of Electrical
and Computer Engineering
University of Florida
Gainesville, FL

Leonardo Leon
Department of Electrical
and Computer Engineering
University of Florida
Gainesville, FL

Carlos Sabogal
Department of Electrical
and Computer Engineering
University of Florida
Gainesville, FL

***Abstract*— We designed a remote temperature monitoring system using two source node XBees, each connected to an Arduino running the temperature monitoring program. The coordinator XBee was running a data collection program to collect temperature readings from each node in real time. This setup allowed us to use the XBees with minimal interference from XCTU.**

I. Introduction

For the final project, we wanted a design that worked as an alert system for a setting that would benefit from remote monitoring. We quickly decided that some kind of storage facility would make the most sense, because it would not be practical to have a security guard or other monitor to stay in a storage room at all times and keep tabs on how the materials are doing. From this, we eventually settled on lumber storage and having a system to continually report back the temperature that the wood was stored in to ensure that it was within an acceptable temperature range. This could be useful in real life because companies in construction and similar industries need to have their lumber within certain specifications, otherwise it goes to waste.

II. System Architecture

For our final project, we used 2 Arduino Unos alongside the XBee 3 Mesh Kit that was provided to us in class. We also used 2 NTC thermistors which allowed us to gather analog temperature data that was transmitted between the devices. For the data collection, one of the three XBees was connected to a laptop running a data collection Python script. The script had checks in it to ensure that the temperature data was both correct (e.g. numeric values were returned) and within spec (below 30°C for our purposes).

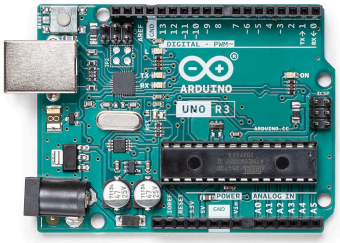

Fig. 1. Arduino Uno Rev 3

Fig. 2. 10k SEN-00250 Thermistor

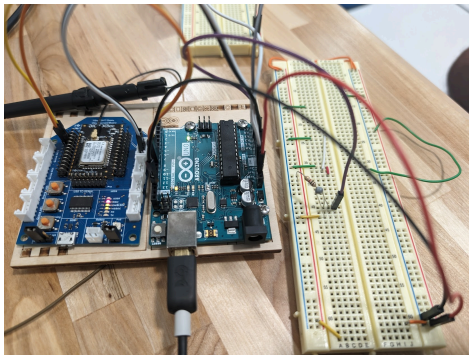

Fig. 3. Source node XBee connected to the Arduino, which collected temperature data from a voltage divider with the thermistor

XBeeRead.py

```python
from digi.xbee.devices import XBeeDevice
import urllib.request

# TODO: Replace with the serial port where your local module is connected to.
PORT = "COM7"
# TODO: Replace with the baud rate of your local module.
BAUD_RATE = 9600
# ThingSpeak API Key and Base URL
baseURL = 'http://api.thingspeak.com/update?api_key=[redacted]'

def main():
    print(" +---------------------------------------+")
    print(" | XBee Python Library Receive Data Sample |")
    print(" +---------------------------------------+\n")

    device = XBeeDevice(PORT, BAUD_RATE)

    try:
        device.open()

        def data_receive_callback(xbee_message):
            # Decoding the message to string
            data_str = xbee_message.data.decode()
            print("From %s >> %s" % (xbee_message.remote_device.get_64bit_addr(), data_str))

            # Assuming the temperature data is well formatted as a string like "22.22C"
            # Extract the temperature value, remove any trailing characters (like 'C')
            temperature_str = data_str.rstrip('C')

            # Convert the extracted temperature string to a float and send to ThingSpeak
            try:
                temperature = float(temperature_str)
                # Check if the temperature is 30 or higher
                if temperature >= 30.0:
                    print("Warning: Temperature too high!")
                # Update ThingSpeak channel with the current temperature value
                f = urllib.request.urlopen(baseURL + str(temperature))
                f.read()
                f.close()
            except ValueError:
                print("Received non-numeric temperature data")

        device.add_data_received_callback(data_receive_callback)

        print("Waiting for data...\n")
        input()

    finally:
        if device is not None and device.is_open():
            device.close()

if __name__ == '__main__':
    main()
```

Fig. 4. Python code to collect temperature data and send it to ThingSpeak

We were provided with example Python code directly from Digi on how to collect data from a remote XBee, and we simply added the checks for temperature monitoring and to send the data to our ThingSpeak URL to this code. The Arduino code, however, was a bit more involved.

We had Arduino code for sending “Hello World” from one XBee to another from the previous modules in this project, and that code served as the framework for sending temperature data. Since our packets would vary with each temperature reading, we split the packet into a constant part that contained data such as the MAC address, and then a variable part that contained the temperature. The trickiest thing we had to account for in the variable packet was changing the checksum value for each temperature reading, but the equation for checksum calculation was well-documented in Digi’s reference manuals for the XBee and XCTU.

The code for reading the temperature is actually code for reading the voltage drop on the analog pin, and then this voltage value is converted to a real life temperature reading based on the specifications for a 10K NTC thermistor, like the one we had. The equation is in Kelvin, so the temperature is first instantiated as a Kelvin reading, and then at the end of the calculation it is converted to Celsius to be sent out to the coordinator.

Fig. 5. Arduino code to read temperature from the voltage divider and transmit that data as an XBee packet

III. Experimental Procedure

Our procedure was to place both sensors in opposite parts of a room to cover the most area with our devices and placed the coordinator node in another corner. To induce variation, we are simply putting our fingers over the temperature sensor to simulate an increase in temperature.

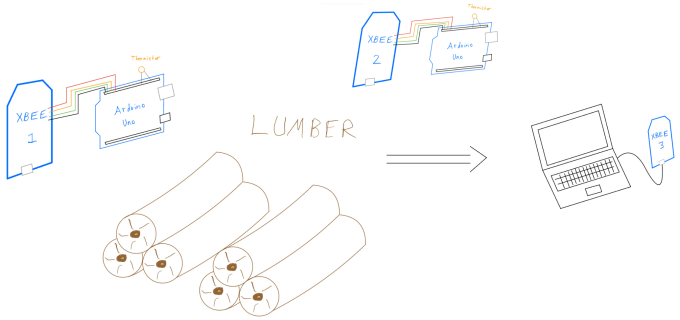

Fig 5. Environment layout with sensors and coordinator node

IV. Experimental Results

For our results, we printed the live temperature reading from both of our circuits to the python terminal. This data is also getting sent to ThingSpeak, with each sensor node plotting to its own graph at the free rate that ThingSpeak offers. The data we collected is what we expected as room temperature is generally considered to be 20 to 25 degrees Celsius or 68 - 78 degrees Fahrenheit. We created an alert to go off at 30 degrees Celsius which we got to by putting our fingers around the sensor which was an expected rise in temperature.

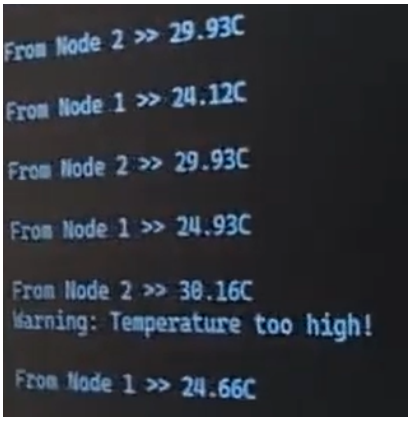

Fig. 6. Data collected from the Python script showing Node 2 above 30°C

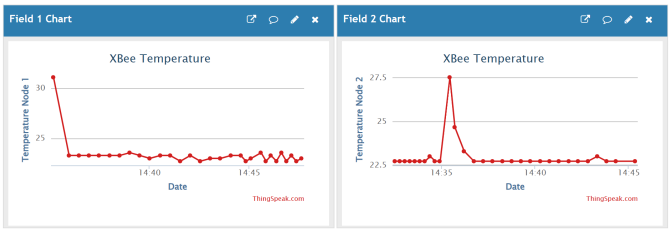

Fig. 7. Data output for both nodes in ThingSpeak

V. Conclusion

This project allowed us to obtain useful and accurate data with very minimal and inexpensive hardware. As-is, our project could be used in real life to monitor temperatures in a controlled environment, and with even more development or more powerful hardware, this project could easily fit the needs and specifications of a big company.