John Iddon
Physics 4990

# LoRa Logs

The past few weeks prior to June 30:

- Transmit and receive tests have been successful.
- The 915MHz LoRa chips are legal in North America (unlike the 433 MHz and 868 MHz variants which would need one to be a HAM radio operator) and have been soldered to back plates that provide easy through-hole pins to mount with. The back plate labels are not correct for the LoRa modules as the back plates are originally intended for ESP8266 chips that have the same form factor as the LoRa modules. The chosen 915MHz LoRa chips are of the typical SX127X family present in almost all LoRa receivers.
- The HopeRFM chips appear to just be re-branded SX127X chips.
- The SX127X chips are 3.3V logic, and while they are not listed as 5V tolerant, they seemed to be in short tests (though I wouldn't want to push it past such and destroy good transceivers).
- Working with ESP8266 modules as the mircocontroller for interfacing with the SX127X chips proved to be more difficult than expected as one needs to use the “Arduino” pin numbering scheme (since we're working in the Arduino IDE) for the ESP8266, which is different from the board numbering, and the many other numbering schemes as well.

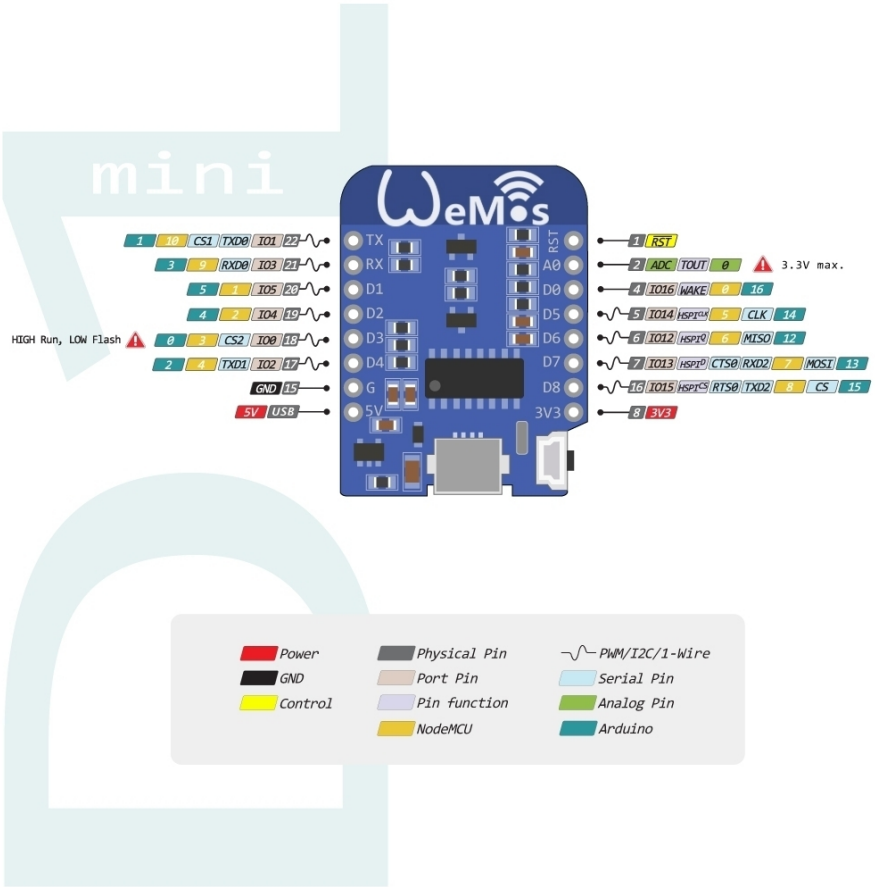


Figure 1: Wemos D1 Mini (ESP8266) pin diagram with various layout numbering schemes. Image sourced from: https://escapequotes.net/esp8266-wemos-d1-mini-pins-and-diagram/

- The website template I created was used to make a website to hold the live LoRa data once sensors are constructed. The website is not yet live.

June 30, 2019

- Power supply options:
  - One interesting method to me has been the idea of directly using power from a solar panel to do transmissions during the day from a remote sensor, that is to say, with no battery attached. I was hoping a typical USB car charger would work, and using two small panels in parallel I was able to run a large USB load (a USB high speed fan), though it dragged down the voltage significantly. It's perhaps possible then that with the most likely lower power consumption of the LoRa+microcontroller setup that it could be run directly from a solar panel without a battery in use, which would make it a very very long term sensor if it indeed will work.
  - I've checked out some of the 18650 Li-ion batteries I have, and two of them have built in protection circuits, or at least claim to have, which should be useful. I also have acquired 18650 battery holders and boost converters that can bring up the 18650's voltage to 5 volts, so they could be a useful option for powering sensors as well, and with a lithium cell charge controller circuit could easily be charged with a solar panel.
- Website:
  - LoRa website is now available live at https://johniddon.github.io . Bandwidth seems to be limited as large pictures take a long time to load, but that's perfectly fine for what I need for this website. Hosting is free too, you simply create a repository named username.github.io where username is your username, and then put your HTML files in the repository and boom, the website is live.
  - One issue with the website I ran into was that the repository needs to be public, as I had initially set it to private and it didn't work.
  - Another issue one may encounter is needing to wait a little time before it goes live and appears initially. It seems to work fine, it just takes patience when GitHub initially processes everything.

July 1, 2019

- Basic transmit and receive test:
  - For some reason my laptop has forgotten the boards configuration for the esp8266 based chips. Under preferences in the Arduino IDE add http://arduino.esp8266.com/stable/package_esp8266com_index.json in the “Additional Boards Manager URLs” section. Clicking OK, then go to Tools --> Board --> Board Manager. Install esp8266, specifically the one by ESP8266 Community.
  - I'm going to redo my code for the basic transmit and receive tests so that they're a touch more elegant.
  - Arduino IDE seems to have messed itself up as nothing will compile properly, not even the simplest of examples that I know should be fine and are designed to run on the selected boards, yet I still get errors along the lines of “Cannot compile for board Wemos D1”. I'll reinstall tomorrow.

July 2, 2019

- Reinstalling Arduino IDE
  - Uninstalled existing Arduino IDE.
  - Manually deleted Arduino file in Documents under my user.
  - Reinstalled Arduino IDE.
    - Specifically I used the portable ZIP file version, so no “install” is required as it's just an executable.
  - Put in the additional boards manager as explained yesterday and install ESP8266 boards.
  - Reinstall LoRa library by Sandeep Mistry from the Libraries Manager.
  - Success! I can now compile example programs properly.
- Rebuilding my receive and transmit programs with thorough commenting.
  - Receive
    - Based upon the example receive program from the LoRa.h library, my receive program will print out received LoRa packets as well as the RSSI (Received Signal Strength Indicator) of that packet to the serial monitor.
    - In the box below is my code.

```
/*
John Iddon - LoRa Rx Basics.
Uses code from examples of the LoRa.h library by Sandeep Mistry.
*/
#include <ESP8266WiFi.h>
#include <SPI.h>
#include <LoRa.h> //Includes LoRa.h library by Sandeep Mistry

//pin definitions
#define ss 15 //defines slave select (SS) pin, AKA chip select.
#define rst 14 //defines reset pin.
#define dio0 5 //defines dio0 pin.

void setup()
{
  //open serial port.
  Serial.begin(115200);
  while (!Serial);
  Serial.println("LoRa Receiver");

  LoRa.setPins(ss, rst, dio0); //set pins


  //433E6 for Asia
  //866E6 for Europe
  //915E6 for North America
  while (!LoRa.begin(915E6)) //sets chip frequency. 915MHz is what's legal in NA without being a HAM.
  {
    Serial.println("b."); //test at this point in program to ensure functionality on serial monitor.
    delay(500);
  }
  // SyncWord helps keep other LoRa traffic out by ensuring only messages with the same SyncWord are output/used.
  // Ranges from 0-0xFF
  LoRa.setSyncWord(0xF3);
  Serial.println("LoRa Initializing OK!");
}


void loop()
{
  // Look to receive any packets if they're available.
  int packetSize = LoRa.parsePacket();
  if (packetSize)
  {
    // if packet has any size to it AKA exists.
    Serial.print("Received packet '");

    //Display the packet to serial monitor.
    while (LoRa.available())
    {
      String LoRaData = LoRa.readString();
      Serial.print(LoRaData);
    }
    // Print RSSI for packet to serial monitor.
    Serial.print("' with RSSI ");
    Serial.println(LoRa.packetRssi());
  }
}
```

- The receiver code works as desired when using an example transmitter.

- Transmit
  - Also based on the example from the LoRa.h library. Useful in that it will count upward so that one can ensure all of the packets are being received, something that will be especially useful when doing range tests to see when we start to miss certain packets.
  - The following code is used:

```
#include <ESP8266WiFi.h>
#include <SPI.h>
#include <LoRa.h>

#define ss 15 //defines slave select (SS) pin, AKA chip select.
#define rst 14 //defines reset pin.
#define dio0 5 //defines dio0 pin.


int counter = 0; //variable to be incremented for each packet sent

void setup() {
  Serial.begin(115200); //Open serial port.
  while (!Serial);

  Serial.println("LoRa Sender");

  LoRa.setPins(ss, rst, dio0); //set pins


  if (!LoRa.begin(915E6)) { //Sets LoRa chip to 915MHz.
    Serial.println("Starting LoRa failed!");//Prints if initialization of chip fails.
    while (1);
  }
}

void loop() {
  Serial.print("Sending packet: ");
  Serial.println(counter);

  // send packet
  LoRa.beginPacket(); //starts packet.
  LoRa.print("Hello world "); //Adds string to packet.
  LoRa.print(counter);//Appends counter variable value to packet.
  LoRa.endPacket(); //Ends the packet and sends it over LoRa.

  counter++; //increments the counter.

  delay(5000); //Limits transmitter to broadcast once every 5 seconds.
}
```

  - Interestingly, now when using the numbered sends, I see that occasionally we're missing a few transmissions, as shown in this serial monitor screen shot in Figure 2. The true question is however, is the issue with the transmitter or the receiver.

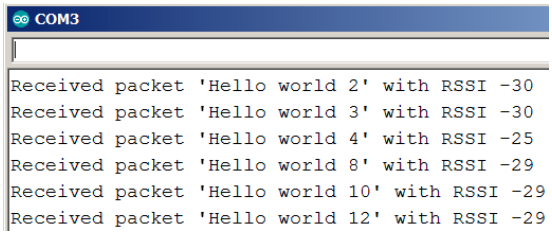


Figure 2: Screenshot of the serial monitor output from the LoRa receiver. It can be seen in the counting numbers how certain transmissions are being lost.

  - Part of me also is leery of the serial monitor, so let's slow that down a touch first to 9600 baud (changed in program line Serial.begin(9600);). At this lower baud rate of the serial port on the receiver module I miss messages between every received one, which leads me to believe perhaps I'm overwhelming the serial port.

- Let's set it the serial port on the receiver back to 115200 baud, as at 115200 baud the serial port of the transmitter never misses any of the prints of the counter, leading me to believe it's either the receiver or it's something to do with the LoRa modules themselves getting overwhelmed at transmitting at such a fast pace perhaps? Let's try slowing down the LoRa transmissions on the transmitter to once every 10 seconds to start off.
- With the transmitter transmitting once every 10 seconds we're still missing transmissions, at once case we even got a string of garbage received, as shown in Figure 3. I wonder if we're perhaps dealing with some sort of interference source.

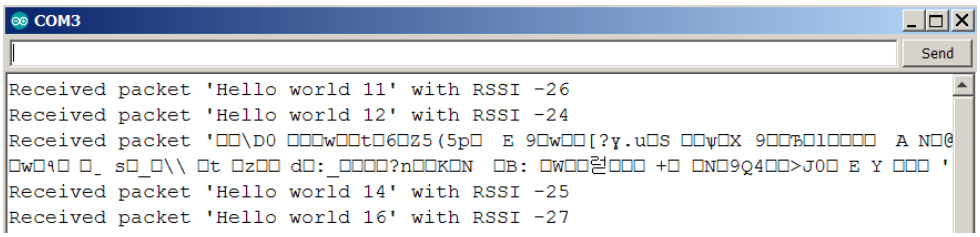

Figure 3: Serial monitor output with transmitter sending once every 10 seconds. Transmissions are still being missed by the receiver, and one transmission received even was a jumbled mess, perhaps due to interference.

- Sometimes we're missing large gaps of transmissions, and other times we get several in a row, so the issue seems to be inconsistent, which is something interference also likes to be. Let's try changing the sync word on both the transmitter and receiver and see if that helps or not.
- Changing the sync word on only the receiver still let it receive some of the messages from the transmitter, which makes me question if the sync word portion of the library is functional or not.
- It appears the joke is on me, so to speak, as I didn't have a sync word set in the transmitter. Now the receiver seems to be getting every sent packet. I'm glad this was a simple mistake and not a larger issue at play.
- Occasionally there is still a missed or jumbled packet, but only 1, never the several in a row I was seeing before. This seems acceptable for now, and is something I can easily overcome with a simple call back in future implementations with a confirmation of a received packet. I've received over 100 packets with only 3 errors, that's probably more reliable than most of our internet connections to remote servers, so I'll happily call this a success for now. Below is the fixed transmitter code.

```
/*
 * Code by John Iddon
 * Includes code from examples in the LoRa.h library from Sandeep Mistry
 */

#include <ESP8266WiFi.h>
#include <SPI.h>
#include <LoRa.h>

#define ss 15 //defines slave select (SS) pin, AKA chip select.
#define rst 14 //defines reset pin.
#define dio0 5 //defines dio0 pin.


int counter = 0; //variable to be incremented for each packet sent

void setup() {
  Serial.begin(115200); //Open serial port.
  while (!Serial);

  Serial.println("LoRa Sender");

  LoRa.setPins(ss, rst, dio0); //set pins


  if (!LoRa.begin(915E6)) { //Sets LoRa chip to 915MHz.
    Serial.println("Starting LoRa failed!");//Prints if initialization of chip fails.
    while (1);
  }
  // SyncWord helps keep other LoRa traffic out by ensuring only messages with the same SyncWord are output/used.
  // Ranges from 0-0xFF
  LoRa.setSyncWord(0xF3);
}

void loop() {
  Serial.print("Sending packet: ");
  Serial.println(counter);

  // send packet
  LoRa.beginPacket(); //starts packet.
  LoRa.print("Hello world "); //Adds string to packet.
  LoRa.print(counter);//Appends counter variable value to packet.
  LoRa.endPacket(); //Ends the packet and sends it over LoRa.

  counter++; //increments the counter.

  delay(5000); //Limits transmitter to broadcast once every 5 seconds.
}
```

July 3, 2019

- Website
  - Website has been updated, making the link bar more compatible with smaller screens, adding sample graphs from my solar voltage sensors to the live data page, and I've also added information to the Guides + Build Log page. This build log is now also attached to the website as a PDF that opens in a new tab for the viewer when they click the link.
  - QR codes successfully can redirect a phone to the website, which will be handy for marking the sensors with. I'll do up some high resolution ones soon, and will perhaps see if I can cut them out of vinyl accurately enough to be scannable as that would be useful for marking the outsides of sensor containers.

July 6, 2019

- LoRa Connection Improvement
  - With the testing out of the way, two side by side mounts for the ESP8266 based Wemos Minis and the LoRa modules were done up, as shown below in Figure 4 and Figure 5.

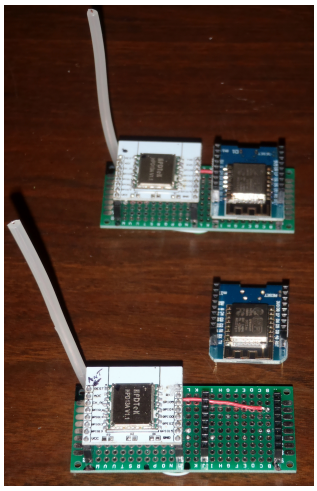

Figure 4: Top view of the two LoRa-ESP8266 interface boards made on simple perfboard.

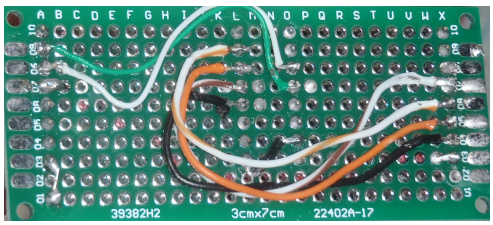


Figure 5: Back side of new LoRa-ESP8266 interface boards made on simple perfboard.

- As can be seen in Figure 6 below, the screenshot of the serial monitor output from the new receiver board receives messages from the new transmitter board very well, obtaining great RSSI values. The nice thing about these new boards is I've soldered on the pins for the ESP8266 boards that have connectors on the top and pins on the bottom, allowing other connections to the ESP8266 other than just the LoRa transceiver, such as sensors, screens, etc.
- It's interesting, as shown in the screenshot in Figure 6, that the interference with garbage data comes in between packet 14 and packet 15, so it seems to be unrelated to my packets, which is good. With some simple filtering of the messages, it will be easy to ensure the only messages acted on will be the ones with proper data.

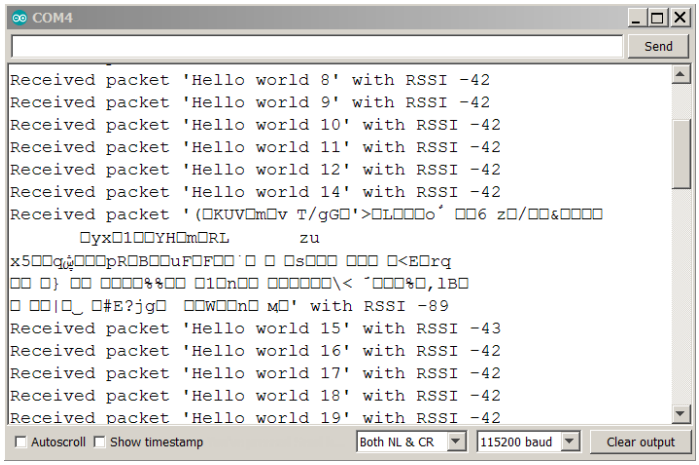


Figure 6: Serial monitor output from the new receiver module. Note the interesting interference between packets 14 and 15, which hopefully means the interference is from an external source and not a loss of data at all in this case.

July 9, 2019

- Sensor Data over LoRa
  - I'll start by using a TMP36 temperature sensor for my first LoRa based sensor since it's a simle analog interface. The TMP36 is a 3 pin sensor (ground, source voltage of either 3.3 volts or 5 volts, and the sensor output pin). Attaching the sensor to the 3.3 volt rail of the Wemos Mini and the output to A0, the only analog input on the ESP8266 based Wemos mini, the sensor can be read at any time simply by reading A0 and applying the following formula: $Temperature = \left(\left(\frac{analog\ input\ value}{1024} \times 3.3\ volts\right) - 0.5\ volts\right) \times 100\ ^\circ C / volt$
  - Note in the code below that I had to let sensorValue be a foating point type, as if I used integer when I divided by 1024 and tried to store it in a floating point for variable t, it always gave back zero, which I find to be very strange. Using sensorValue as a floating point solves all of the issues though.
  - The code below is used for the new TMP36 temperature transmitter, while the receiver code remains unchanged. It's simply a matter of reading the analog input, converting to temperature, adding the data to the packet, and sending the packet. As can be seen in Figure 7 and Figure 8, the system works very well at transmitting the data, but I question the temperature value and think my TMP36 needs calibration.

```
/*
 * Code by John Iddon
 * Includes code from examples in the LoRa.h library from Sandeep Mistry
 */

#include <ESP8266WiFi.h>
#include <SPI.h>
#include <LoRa.h>

#define ss 15 //defines slave select (SS) pin, AKA chip select.
#define rst 14 //defines reset pin.
#define dio0 5 //defines dio0 pin.


int counter = 0; //variable to be incremented for each packet sent

void setup() {
  Serial.begin(115200); //Open serial port.
  while (!Serial);

  Serial.println("LoRa Sender");

  LoRa.setPins(ss, rst, dio0); //set pins


  if (!LoRa.begin(915E6)) { //Sets LoRa chip to 915MHz.
    Serial.println("Starting LoRa failed!");//Prints if initialization of chip fails.
    while (1);
  }
  // SyncWord helps keep other LoRa traffic out by ensuring only messages with the same SyncWord are output/used.
  // Ranges from 0-0xFF
  LoRa.setSyncWord(0xF3);
}

float sensorValue = 0; //Initialize variable to hold analog input value.
float t = 0; //Initialize variable for temperature.

void loop() {

  sensorValue = analogRead(A0); //Read analog pin A0, store in sensorValue.
  t = (((sensorValue/1024)*3.3)-0.5)*100; //Compute temperature based on sensorValue.
  //t = sensorValue/1024; //For testing the issue with int vs float for sensorValue.
  Serial.print("A0 Voltage: ");
  Serial.print(sensorValue);
  Serial.print("   Temperature: ");
  Serial.println(t);
  Serial.print("Sending packet: "); //Allows us to know when a packet is being sent.
  Serial.println(counter);

  LoRa.beginPacket(); //Starts packet.
  LoRa.print("Current temperature: "); //Adds string to packet.
  LoRa.print(t); //Adds value of t to packet.
  LoRa.print("  Packet number: "); //Adds string to packet.
  LoRa.print(counter); //Appends counter variable value to packet.
  LoRa.endPacket(); //Ends the packet and sends it over LoRa.

  counter++; //increments the counter.

  delay(5000); //Limits transmitter to broadcast once every 5 seconds.
}
```

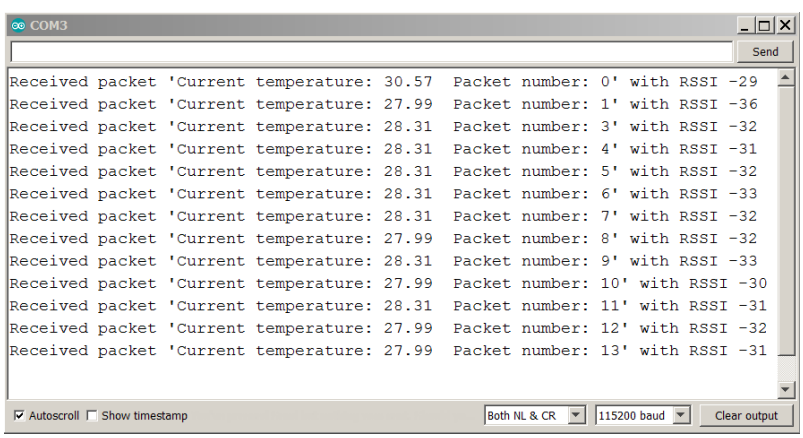


Figure 7: LoRa receiver printing out temperature data that was sent over LoRa.

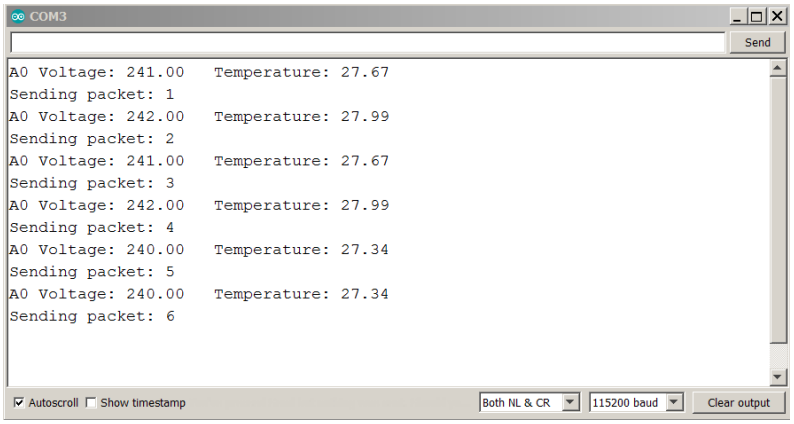

Figure 8: LoRa transmitter printing the temperature data to the serial monitor and confirming the sending of the data.

- Currently the TMP36 is reading 27.34±0.7 °C, while my Fluke 52 thermocouple (of which I cannot guarantee the accuracy of) is reading 20.9±0.3 °C. **So the TMP36's conversion from voltage to degrees Celsius will certainly need some tweaking in the future**, but there's no point in that now until I can compare it to some values measured by a known calibrated source.
- Adding a small ceramic capacitor across the analog input and ground in the future might help eliminate some of the noise I've observed from these TMP36 sensors.
- Any other data needed to be transmitted will be similarly simple, just a matter of reading it from the source and adding it to the LoRa packet.

July 10, 2019

- Startup Temperature Irregularities
  - I've noticed that when initially booting the LoRa-ESP8266 system with the TMP36 attached, the initial values that the TMP36 outputs while “warming up” so to speak are higher by several degrees than the value it settles to a few seconds later. I have added a delay to the setup section of the code (which only runs once) to wait for 3 seconds after the LoRa module is initialized before TMP36 values start being read and transmitted.
- Power Optimization
  - Currently the receiver with the LoRa transceiver and ESP8266 draws about 90 mA (I say about as this is measured with a cheap USB ammeter), while the transmitter draws about 90 mA as well.
  - Microcontroller Optimization
    - One of the best ways to shave down on the power consumption is by optimizing the microcontroller running this system, the ESP8266 by taking advantage of the powerful sleep states available. I've found some great information at the following link: https://randomnerdtutorials.com/esp8266-deep-sleep-with-arduino-ide/ .
    - The link above claims to be able to get an ESP8266 down to around 20 µA in deep sleep mode, though keeping in mind that is a bare ESP8266 and not a full board with things like voltage regulators, USB to serial converters, etc.
    - Utilizing deep sleep should be very possible for many of my planned sensors. For

temperature sensors, having the microcontroller wake every X seconds to send a temperature and then go back to sleep for X seconds, repeating the cycle endlessly, will definitely work well. For something like a passive infrared sensor, having the sensor itself trigger a reset could just have the ESP8266 send a packet at startup and then go back to sleep indefinitely until it's woke again by the next reset from the infrared sensor.

July 12, 2019

- LoRa Range Testing
  - Using the default spreading factor, power, and data rate in the LoRa.h library (not declaring a setting sets them to the default values listed in the library documentation) I have done some range testing down my street, which is not all that long.
  - With my transmitter set to transmit every second, I should be able to record at what distances it stops being received reliably by using Google Maps' “Measure Distance” tool found by right-clicking on the map. By measuring the RSSI at various locations I can also gauge the signal strength and how the signal fares with line of sight transmission vs. transmissions without line of sight.
  - The transmitter is placed on the dashboard of a car and the receiver in the front window of my house. The following data was captured from the receiver.

| **Data Point Number** | **Approximate Distance (m)** | **RSSI (Closer to 0 is stronger.)** | **Line of Sight? (YES or NO)** |
|---|---|---|---|
| 1 | 1 | -34 ± 5 | YES |
| 2 | 8 | -62 ± 5 | YES |
| 3 | 101 | -89 ± 3 | NO |
| 4 | 150 | -89 ± 4 | NO |

  - When we lose line of sight when just going barely down the street the signal also cuts out at around -90dB.
  - Interestingly on the next street over through all the houses at data point 3 the signal is still reliably received at -89 dB ± 3 dB.
  - Through dense forest behind my house at data point 4 the transmitter was still able to pick up the signal reliably at -89 dB ± 4 dB.

July 13, 2019

- Interference Tests
  - I started with the receiver and transmitter approximately 1 to 2 meters apart, placing the interference sources nearby (almost directly against) the transmitter and then nearby the receiver. The interference sources used were 2 A/C motors (one with a large heating coil), and one D/C motor. No matter the location of the interference sources (near transmitter or near receiver) there was no noticeable effect on data integrity and negligible change in RSSI value.
  - Next I tested with approximately 10 meters between the sensors, as well as 3 walls. Placing the same interference sources near the transmitter and then near the receiver. No matter the location of the interference sources (near transmitter or near receiver) there was no noticeable effect on data integrity and negligible change in RSSI value.

- These results are great, so these “noisey” (in more ways than one) motors were not able to noticeably affect the transmission of data with the LoRa system, which will be great for actual sensor deployment and creating a reliable system.

July 14, 2019

- Putting Data on the Cloud
  - I'll be using ThingSpeak to do my cloud data for the sensors since I have prior experience with it.
  - From my previous solar panel voltage monitors, this shouldn't be too different a task, the hardest part being taking apart the data packet sent over LoRa if I want to send various numbers in one packet. I'll start with just one for simplicity however and move on to more as the project progresses.
  - I have the LoRa receiver waiting 15 seconds after a packet was last sent to ThingSpeak to send another as that is the fastest transmission rate allowed by ThingSpeak for a free account. **The LoRa transmitter sends ever 17 seconds so that it shouldn't fall in the 15 seconds of delay from the receiver in case their clocks run slightly differently**.
  - One current issue with my code is that there's no quality control. For example, if I get one of those random corrupted messages from nowhere it will be sent to ThingSpeak (I'm interested as to how ThingSpeak will react to such a data string.), as would rogue data from a malicious individual. I will implement a preamble that will be recognized by my LoRa receiver in the future and utilize some sort of encryption to hopefully prevent malicious use.
  - I've created a new ThingSpeak account for LoRa sensors, the user is iddonlora, which can be used to find the channels for my LoRa transmitted data on ThingSpeak.
  - The LoRa transmitter now simply sends the temperature as its data packet, and the LoRa receiver takes in the data packet sent over LoRa and connects to my WiFi and sends it over the internet to ThingSpeak. The code functions properly and is shown below, as well as a screenshot of the current LoRa receiver serial monitor output shown in Figure 9. Note that I've removed my API key and WiFi information from the code.
  - I also received one of those strange corrupted messages between packets, and ThingSpeak simply displayed it as “NaN” on the graph in between the two adjacent points and upon refreshing the page just shows a tiny gap there as if no message were received, so those occasional strange corrupted looking messages should not be an issue right now.

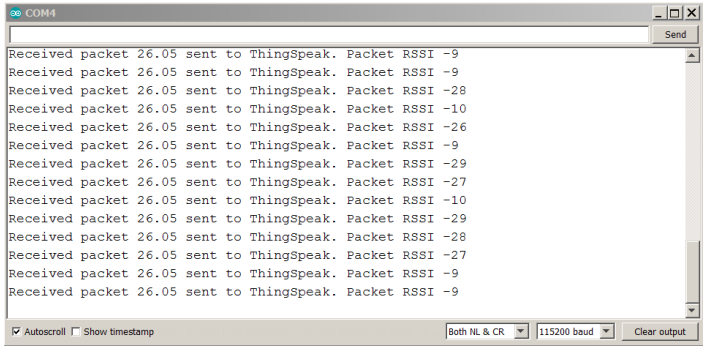


Figure 9: Serial monitor output from new LoRa receiver code that takes in temperature data over LoRa and sends it over WiFi and the internet to ThingSpeak.

```
/* John Iddon - LoRa receiver sends data to ThingSpeak.
Uses code from examples of the LoRa.h library by Sandeep Mistry.*/
#include "ThingSpeak.h" //Includes the functions used for uploading to ThingSpeak.
#include <ESP8266WiFi.h> //Allows for usage of the ESP8266's WiFi modem.
#include <SPI.h>
#include <LoRa.h> //Includes LoRa.h library by Sandeep Mistry

//Pin definitions.
#define ss 15 //defines slave select (SS) pin, AKA chip select.
#define rst 14 //defines reset pin.
#define dio0 5 //defines dio0 pin.

char ssid[] = "XXXXXXXXX"; //your network SSID (name)
char pass[] = "XXXXXXXXX"; //your network password

int status = WL_IDLE_STATUS; //Used for connection loop.
WiFiClient  client; //Creates client for connection.

unsigned long myChannelNumber = XXXXXXXX;  //Number for ThingSpeak channel.
const char * myWriteAPIKey = "XXXXXXXXXXXXX";  //API key for my Thingspeak channel.


void setup()
{
  //Start of WiFi configuration.
  WiFi.mode(WIFI_STA); //Forces esp8266 into station mode.
  WiFi.begin(ssid, pass); //Opens wifi connection to your network.
  WiFi.mode(WIFI_STA); //Forces esp8266 into station mode.

  Serial.begin(115200); //Open serial port.
  while (!Serial);
  Serial.println("LoRa Receiver");

  while (WiFi.status() != WL_CONNECTED)
  { //prints dots on the serial monitor until wifi connects
     delay(500);
     Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());  //Prints the IP address the module picks up via DHCP

  ThingSpeak.begin(client);
  //End of Wifi configuration.

  //Start of LoRa configuration.
  LoRa.setPins(ss, rst, dio0); //set pins
  //433E6 for Asia
  //866E6 for Europe
  //915E6 for North America
  while (!LoRa.begin(915E6)) //sets chip frequency. 915MHz is what's legal in NA without being a HAM.
  {
   Serial.println("b."); //test at this point in program to ensure functionality on serial monitor.
   delay(500);
  }
  // SyncWord helps keep other LoRa traffic out by ensuring only messages with the same SyncWord are output/used.
  // Ranges from 0-0xFF
  LoRa.setSyncWord(0xF3);
  Serial.println("LoRa Initializing OK!");
  //End of LoRa configuration.
}


void loop()
{
  // Look to receive any packets if they're available.
  int packetSize = LoRa.parsePacket();
  if (packetSize)
  {
   // if packet has any size to it AKA exists.
   Serial.print("Received packet ");

   //Display the packet to serial monitor.
   //while (LoRa.available())
   //{ //While loop not needed since using readString rather than read.
     String LoRaData = LoRa.readString();
     Serial.print(LoRaData);
     //Sends LoRaData to ThingSpeak for my channel in field 1.
     ThingSpeak.writeField(myChannelNumber, 1, LoRaData, myWriteAPIKey);
     Serial.print(" sent to ThingSpeak. ");
   //}
   // Print RSSI for packet to serial monitor.
   Serial.print("Packet RSSI ");
   Serial.println(LoRa.packetRssi());
   delay(15000); //15 second delay before another packet
   //can be sent to ThingSpeak due to their rules.
  }
}
```

```
/* Code by John Iddon - LoRa transmitter - Only sends temperature data
 *  over LoRa for use by the receiver to send to ThingSpeak.
 Includes code from examples in the LoRa.h library from Sandeep Mistry
 */

#include <ESP8266WiFi.h>
#include <SPI.h>
#include <LoRa.h>

#define ss 15 //defines slave select (SS) pin, AKA chip select.
#define rst 14 //defines reset pin.
#define dio0 5 //defines dio0 pin.


int counter = 0; //variable to be incremented for each packet sent

void setup() {
  Serial.begin(115200); //Open serial port.
  while (!Serial);

  Serial.println("LoRa Sender");

  LoRa.setPins(ss, rst, dio0); //set pins


  if (!LoRa.begin(915E6)) { //Sets LoRa chip to 915MHz.
    Serial.println("Starting LoRa failed!");//Prints if initialization of chip fails.
    while (1);
  }
  // SyncWord helps keep other LoRa traffic out by ensuring only messages with the same SyncWord are output/used.
  // Ranges from 0-0xFF
  LoRa.setSyncWord(0xF3);


  //Wait 3 seconds for TMP36 to warm up.
  Serial.println("Wait1");
  delay(1000);
  Serial.println("Wait2");
  delay(1000);
  Serial.println("Wait3");
  delay(1000);
}

float sensorValue = 0; //Initialize variable to hold analog input value.
float t = 0; //Initialize variable for temperature.

void loop() {

  sensorValue = analogRead(A0); //Read analog pin A0, store in sensorValue.
  t = (((sensorValue/1024)*3.3)-0.5)*100; //Compute temperature based on sensorValue.
  //t = sensorValue/1024; //For testing the issue with int vs float for sensorValue.
  Serial.print("A0 Voltage: ");
  Serial.print(sensorValue);
  Serial.print("   Temperature: ");
  Serial.println(t);
  Serial.print("Sending packet: "); //Allows us to know when a packet is being sent.
  Serial.println(counter);

  LoRa.beginPacket(); //Starts packet.
  LoRa.print(t); //Adds value of t to packet.
  LoRa.endPacket(); //Ends the packet and sends it over LoRa.
  Serial.print(t);
  Serial.println(" was sent via LoRa.");

  counter++; //increments the counter.

  delay(17000); //Limits transmitter to broadcast once every 17 seconds.
}
```

July 17, 2019

- Website Updates
  - I've updated the log as of today's entry on the website and added the LoRa transmitted ThingSpeak data so that it's public.
  - The section explaining how LoRa functions has been updated as well with the summary I made last week.

July 19, 2019

- Range Test 2

- I did a second range test today down the street, this time with the receiver on top of a car on the street, thus giving me line of sight down most of the street. Since the receiver and transmitter are still close to the ground though, within Fresnel zone 1, there will be reflection from the ground and other objects that will degrade the signal quality. The data is shown in the following table:

| **Data Point Number** | **Transmitter Location Description** | **Approximate Distance (Very rought estimate.) (m)** | **RSSI** | **Line of Sight?** |
|---|---|---|---|---|
| 1 | Near receiver. | 2 | -42 ± 4 | YES |
| 2 | End of this block. | 192 | -85 ± 5 | YES |
| 3 | End of next tiny block. | 275 | -92 ± 6 | YES |
| 4 | Quarter down third block. | 375 | -95 ± 4 | YES |
| 5 | Half down next block. | 450 | -97 ± 3 | Intermittent |

- The distance between the transmitter and receiver is very approximate, as it's hard to gauge how far down the road the transmitter was. Again though, using approximate landmarks and Google Maps' distance tool, the distances were able to be roughly approximated.
- Once data point 5 was reached a lot of data corruption started to occur in the transmitted data and it could be seen that there were a lot of messages being missed by the delay between many of the transmissions that should have been arriving every 1 second.
- As can be seen in the data, the range is outstanding with line of sight compared to the previous range test without much line of sight. With the receiver raised higher in the air, reducing the reflections as well as increasing the area over which it has line of sight, I think the range should be able to successfully go much further with these transceivers.
- Once I have my GPS sensor up and running I should try to do some sort of a data logging setup to store data about RSSI and GPS coordinates automatically, perhaps plotting them onto a map with a colour code similar to Thomas Telkamp's graph in the lecture linked in my references on the About LoRa page of the website.

- Rough TMP 36 Temperature Calibration
  - Adjusting the voltage offset for the TMP36 sensor (originally -0.5 volts, now -0.567 volts), I've brought it within approximately 0.5 of a degree constantly of room temperature (around 22 °C). When heated with a hair dryer to approximately 41 degrees Celsius the TMP36 was within 2 °C of my Fluke 52 thermocouple, though my thermocouple's calibration has not been confirmed. The lower end drifts in calibration a bit, using an ice water bath enclosed in plastic bags my Fluke thermocouple reads 0.3 °C ± 0.2 °C, while the TMP36 over LoRa reads 1.95 °C ± 0.5 °C. So for the quick calibration adjustment I did, to always be within less than 2.5 °C difference from my expensive Fluke thermocouple is a rather good result for now. This level of accuracy would probably be fine for many applications as well, say for identifying cooling or heating failure in a building or refrigeration system. Again though, the Fluke has not been calibrated in a long time (as I purchased it used), so it may well not be very accurate, though the calibration seems at least decent due to its performance in the ice water test.

- July 30, 2019
  - Heliostat
    - Design has been progressing for the heliostat project, hence the lack of progress for LoRa so far this week.
  - GPS
    - I've begun work on the GPS module to use with LoRa hopefully, specifically a NEO-6M GPS module. It has 5 pins, VCC (3.3 V though supposedly 5 V tolerant, but I wouldn't push my luck), ground, transmit (TX), receive (RX), and pulse per second (PPS – a high accuracy clock signal). PPS will not be used in my project.
    - Connecting to a Wemos D1 (the big Wemos), the most difficult part was getting the pin numbers correct again. D8 is 0 and D2 is 16 on the large Wemos D1 when using the Arduino IDE.
    - The NEO-6M outputs on a serial interface a large amount of GPS data in NMEA format. Software based serial on capable pins of a Wemos or Arduino board can easily read this data.
    - Attaching the transmit of the NEO-6M to the software serial receive port on D8 (pin 0 in the Arduino IDE) and the receive of the NEO-6M to the software serial transmit port on D2 (pin 16 in the Arduino IDE), the NEMEA messages were able to be printed to the serial monitor using a simple loop to print out any received message to the serial monitor of my computer.
    - Taking the longitude and latitude data from the NEMEA messages output by the GPS receiver, with arc minutes properly converted to degrees, and searching it on Google maps brought me right to my location with striking accuracy. Using Google Maps' measure distance tool again, my location was placed via GPS within less than 8 meters of my actual location, remarkably accurate seeing as the test was done from inside my house only near a window, so I can only imagine how good the data would be if the GPS antenna were actually outside.

- August 1, 2019
  - Lethbridge Range Testing
    - Today range testing was done transmitting LoRa data across the coulee from the University of Lethbridge. Using the same transmitter and receiver pair as in the previous range tests, this should ensure the same quality of antennas used in the previous range tests. The transmitter is sending temperature data from a TMP36 once again.
    - Starting with the transmitter across the coulee (near the Lethbridge Radiology Associates building and the off-leash dog area) and the receiver in the new science building, specifically the physics department, I was unable to receive any of the messages being sent once every second. The moment I stepped outside the back of the building however, I started receiving the data with an RSSI value of -88 ± 5. This successful transmission distance, approximated using Google Maps' measure distance tool, was approximately 1.5 kilometers.
    - Returning to the science building and attempting the test again, since I then knew the transmitter was actually working, again there was not a single packet able to be received in the new science building, even with the receiver right against the large glass windows facing across the coulee in the physics department.
    - Moving to U-hall, I was able to receive the data all along E and D blocks on the 6$^{th}$ floor

with very little data corruption, with an RSSI of -87 ± 5. This successful transmission distance, approximated using Google Maps' measure distance tool, was approximately 1.6 kilometers since the transmitter had moved south down the coulee a bit.

- Going to the 5th floor of U-hall in C through E blocks I was able to receive the data again, with very little data corruption and an RSSI around -88 ± 5, in the main hallway through doors with small glass windows (since there are no direct windows as the hall is lined with offices, so I couldn't try the receiver right up against the glass of one of the office windows).
- Screenshots of some of the received data is shown below. In Figure 10 it's shown how loss of line of sight, such as when the transmitter was moved into a dip in the coulee, or tilting one of the antennas drastically while the other remained upright could cause sever data corruption. Figure 11 shows however that with line of sight and parallel antenna orientation the data would have long sequences of perfect data, free from corruption.

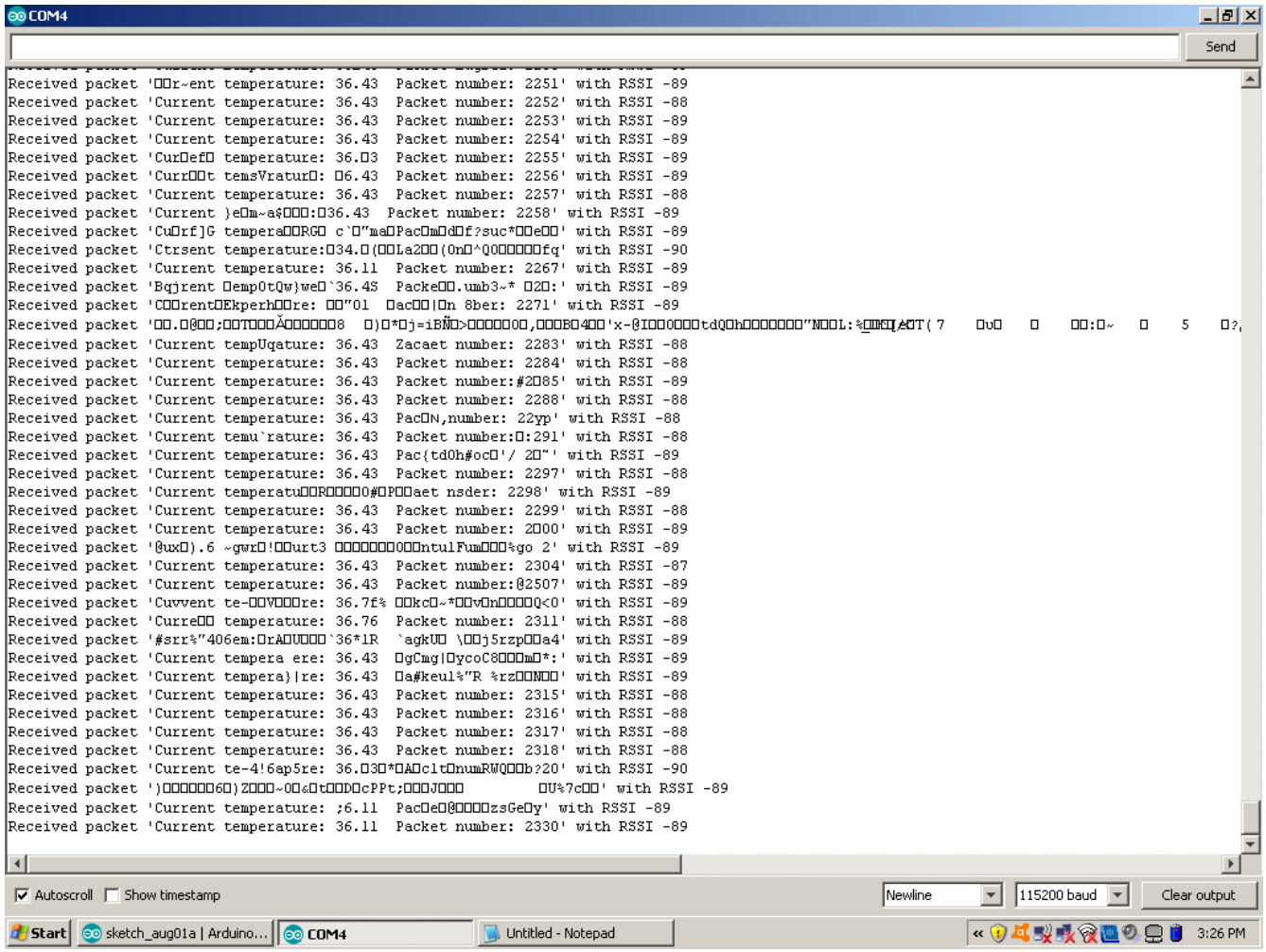


Figure 10 – Mostly corrupted data sent across the coulee back to the University of Lethbridge. Data corruption occurs when line of sight is lost or the antennas are not approximately parallel in orientation.

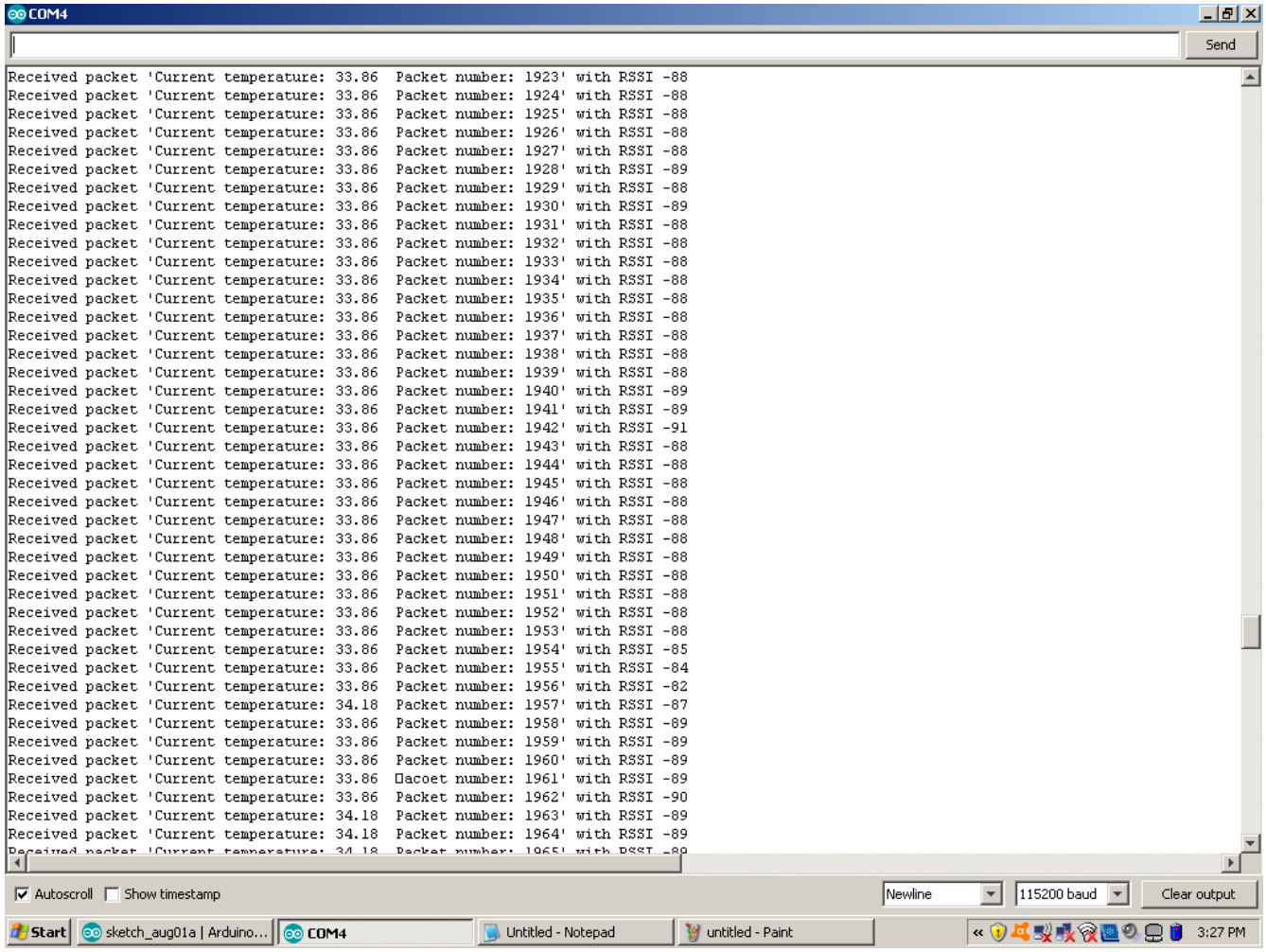


Figure 11 – Data sent across the coulee to the University of Lethbridge, free from data corruption. With line of sight and parallel antennas the packets received would rarely be corrupted.

- I find it shocking that I could receive the data quite well on the 5th floor of Uhall through offices, yet couldn't receive a single packet right against the glass of the new science building up high on what I believe is the 9th floor.
- In the future I should test just how shielded the new science building is using the transmitter nearby and seeing if the new science building still blocks all of the packets.

- August 8, 2019
  - LoRa Range Testing at the U of L New Science Building
    - To determine how badly the new science building at the U of L blocks signals I decided to place the transmitter on the window sill, facing the coulee, in the Physics department, the location where I tried to receive signals sent from across the coulee on August 1.
    - With the transmitter in position I was able to receive packets with little to no corruption all the way down the main Physics department hallway, and out into the main hallway on the 9th floor before losing the signal and then picking it back up at the doors to the

other hallway of the Physics department.

- Heading to the next floor down, uncorrupted packets could again be received near the long hallways that have windows at the opposite end facing the coulee.
- Outside on the back balcony area of the building, the RSSI of the packets received was around -88 ± 5, which is just as bad as when I was getting data outside the new science building from ~1.5 km away across the coulee, so the science building definitely causes significant issues with wireless signals, perhaps due to all of the concrete, rebar, and metal of the siding and window frames.
- Near the windows on what I believe is the 7th floor in the main sitting areas packets can still be received with very little corruption, **so there is decent coverage of parts of the science building, meaning sensors could be deployed around the new science building with only a few receivers around the building.**

- Brief Spectrum Analysis
  - Using the new Keysight spectrum analyzer and antenna kit, Professor Furgason and I were able to look at the 900 MHz to 930 MHz range and, with some tweaking to the sweep speed, were able to see a chirp every second, centered on 915 MHz and spread over the surrounding frequencies.

- August 17, 2019
  - GPS Testing in Lethbridge
    - Setting up the GPS sensor on a LoRa equipped Wemos Mini (TX of GPS hooked to software serial RX of D4 on Wemos mini, RX on GPS not required), the positions were rather out, and this was because of the large amount of walls and metal near my GPS antenna. Placing it into the window the position data now reported as being less than ~5 meters from my actual location, measured using Google Maps' measure distance tool. My commented code for the simple reading of the NEO-6M's output is shown below.

```
#include <SoftwareSerial.h>
//John Iddon - Basic NEO-6M GPS NMEA sentence display - 2019
//Software serial port to talk to NEO-6M GPS module.
//Only rx is needed, but you have to declare both for SoftwareSerial.
//rx doesn't seem to like to work on D0 (Arduino 16).
SoftwareSerial softSer(2, 16); //Setup pins for software serial port.
//rx on D4 of wemosMINI (Aruidno 2), tx on D0 (Arduino 16)

void setup(){
  Serial.begin(115200); //Start actual serial port.
  softSer.begin(9600); //Start software serial port for GPS module.
}

void loop(){
  while (softSer.available() > 0){
    //while software serial available, read the current character.
    byte gpsData = softSer.read();
    Serial.write(gpsData); //Print received character to real serial port.
  }
}
```

  - Parsing GPS Data
    - Next up I'd like to use the TinyGPS++ library (by Mikal Hart) to parse the NMEA sentences for me. There's a lot of data there coming in all of those NMEA sentences, and there's no sense reinventing the wheel when someone has already made code to parse that data. Modifying my existing GPS reader code to include the library and with a small snipped inspired by the examples of the library and by Rui Santos, I wrote the code below. This program successfully extracts the current longitude and latitude from the GPS sensor data in the NMEA sentences and prints it to the serial monitor any time the

GPS data changes.

```
#include <TinyGPS++.h>
#include <SoftwareSerial.h>
//John Iddon - GPS Long.-Lat. NEO-6M GPS display. - 2019
//Software serial port to talk to NEO-6M GPS module.
//Only rx is needed, but you have to declare both for SoftwareSerial.
//rx doesn't seem to like to work on D0 (Arduino 16).
SoftwareSerial softSer(2, 16); //Setup pins for software serial port.
//rx on D4 of wemosMINI (Aruidno 2), tx on D0 (Arduino 16)

TinyGPSPlus gps;

void setup(){
  Serial.begin(115200); //Start actual serial port.
  softSer.begin(9600); //Start software serial port for GPS module.
}

void loop(){
  while (softSer.available() > 0){
    //while software serial available, parse the data.
    gps.encode(softSer.read()); //
    if (gps.location.isUpdated()){ //if the location changed.
      Serial.print(gps.location.lat(), 6);//print latitude.
      Serial.print(F(", ")); //the F allows me to use the comma.
      Serial.println(gps.location.lng(), 6); //print longitude
    }
  }
}
```

- Using the new code and with my current sensor position, the GPS position is again less than ~5 meters away from my actual position using Google Maps' measure distance tool.
- The serial monitor output is nice in that it's latitude comma longitude, so I can paste it into Google and it will instantly show me a point on the map to verify if the data is correct.
- As a random note, I find that sometimes when a piece of code is uploaded to the board, the serial monitor output you open after will be one ever growing line of gibberish being printed out. I find that closing the serial monitor, re-uploading the code, and reopening the serial monitor resolves this.

- Sending GPS Data Over LoRa
  - Now that the GPS data can be easily extracted, it's time to send it over LoRa.
  - Using the simple way to extract the data from the previous code, and then using the latitude and longitude variables as the variables sent in my LoRa packet, I should be able to send that data. The code I wrote below functions properly and my usual LoRa receiver that prints packets to the serial monitor receives the data properly, as shown in Figure 12 below.

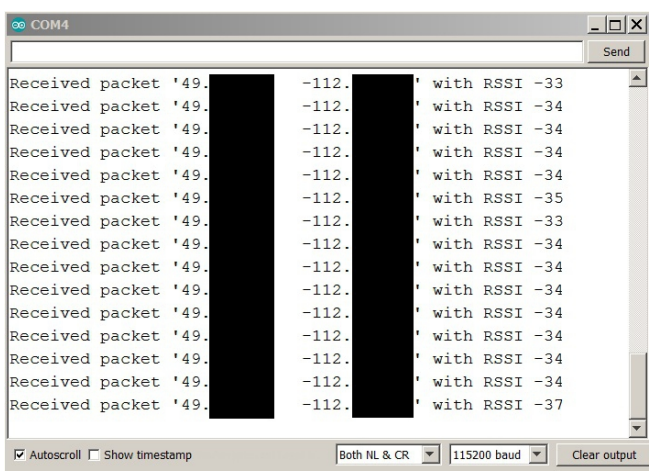


Figure 12 – GPS data received over LoRa. Decimal values have been masked so as not to reveal exact location of sensor to the world on this public log.

```
/*
 * Code by John Iddon
 * Includes code from examples in the LoRa.h library from Sandeep Mistry
 */
#include <ESP8266WiFi.h>
#include <SPI.h>
#include <LoRa.h>
#include <TinyGPS++.h>
#include <SoftwareSerial.h>

#define ss 15 //defines slave select (SS) pin, AKA chip select.
#define rst 14 //defines reset pin.
#define dio0 5 //defines dio0 pin.

SoftwareSerial softSer(2, 16); //Setup pins for software serial port.

TinyGPSPlus gps;

int counter = 0; //variable to be incremented for each packet sent

void setup() {
  Serial.begin(115200); //Open serial port.
  softSer.begin(9600); //Start software serial port for GPS module.
  while (!Serial);
  Serial.println("LoRa Sender");

  LoRa.setPins(ss, rst, dio0); //set pins


  if (!LoRa.begin(915E6)) { //Sets LoRa chip to 915MHz.
    Serial.println("Starting LoRa failed!");//Prints if initialization of chip fails.
    while (1);
  }
  // SyncWord helps keep other LoRa traffic out by ensuring only messages with the same SyncWord are output/used.
  // Ranges from 0-0xFF
  LoRa.setSyncWord(0xF3);


  //Wait 3 seconds for warm up.
  Serial.println("Wait1");
  delay(1000);
  Serial.println("Wait2");
  delay(1000);
  Serial.println("Wait3");
  delay(1000);
}

void loop() {

  while (softSer.available() > 0){
    //while software serial available, parse the data.
    gps.encode(softSer.read()); //
    if (gps.location.isUpdated()){ //if the location changed.
      Serial.print(gps.location.lat(), 6);//print latitude.
      Serial.print(F(", ")); //the F allows me to use the comma.
      Serial.println(gps.location.lng(), 6); //print longitude

      LoRa.beginPacket(); //Starts packet.
      LoRa.print(gps.location.lat(), 6); //Adds lat. to packet.
      LoRa.print("   "); //Adds space to packet.
      LoRa.print(gps.location.lng(), 6); //Adds long. to packet.
      LoRa.endPacket(); //Ends the packet and sends it over LoRa.
      Serial.print("Sent over LoRa packet number ");//confirmation of send.
      Serial.println(counter);
      counter++; //increments the counter.
      delay(1000); //Limits transmitter to send max once every 1 second.
    }
  }


}.
```

- I'm noticing that the data is rather noisy from the GPS however, drifting around an ~12m circle around me. Worried it could be LoRa messing with the GPS signal, I re-uploaded the previous GPS program without LoRa and that drift is still present, but it seems, from my very very small set of samples, to be slightly smaller without LoRa, which is worrisome. Going back to the LoRa program the drift increase seems negligible now, only with the very rare occasional slightly larger deviation from my actual location (by a few extra meters, nothing huge), but it's entirely possible this happened occasionally too on the module without LoRa running and I just didn't test enough points to see it. Without doing a large sample of points from each case, I can't prove whether or not LoRa is increasing the drift.
    - In either case the drift is more than ~2 meters, which is somewhat significant, but could probably be dealt with by averaging several sequential points.

        - The light on the GPS module flashes approximately once per second, which is when it sends the NMEA sentences. Perhaps the LoRa transmission, with the delay after of 1 second was making it so the location picked up my the GPS sensor was picked up right as the LoRa message was sent perhaps having an affect on the GPS data? Either way, the delay needs to be increased to at least 15 seconds for ThingSpeak upload (max upload rate on a free account), which I'll make 17 seconds to be safe again. This change doesn't seem to change much when looking at a few new data points, as the extra LoRa induced drift may well be non-existent to begin with.
        - I swapped out the space between latitude and longitude in the LoRa packet for the same F(“,”) to print a comma instead, which should be nice for parsing the data on the receiver in the future. The change works fine for both the transmitter and receiver.

- August 18, 2019
    - Possible Way to Filter Some Interference
        - While letting the basic GPS receiver pick up LoRa packets, I very rarely got those strange corrupted packets. Looking at each of them, they had RSSI values of -90 and -89, while my nice GPS data had RSSI values around -33. With a simple if statement for RSSI values above a threshold one wanted to set, you could cut off packets you suspect will be corrupted easily. Then again, I was getting lots of great data in the -87 range for RSSI across the coulee, so cutting off data with this method could end up losing valuable good data, and thus valuable range for sensors to be placed at. I have not implemented this method, but this method of filtering out bad data could be useful in the future still though.

    - ThingSpeak and LoRa GPS
        - I'd like to place the GPS data on ThingSpeak so it's accessible for use. I'll need to use a different function to do such, as in this case I'll be using a channel with 3 data fields that all upload together. Referencing the multiple writes example from the ThingSpeak library, I was able to find the needed function (*ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey)*) to upload multiple fields at once.
        - To get the longitude and latitude data separated from the string into two separate variables, I need to parse the string received. Taking some inspiration from the code at https://forum.arduino.cc/index.php?topic=236070.0 , as well as hours and hours of troubleshooting, research, and most likely unnecessary conversion I finally finished the parsing code for the LoRa data. The code I made is shown below, and takes a string with a comma delimiter and breaks it into two floating point numbers. This code works well.

```
void setup() {
  Serial.begin(115200);
}

void loop() {

String LoRaData = "49.671450,-112.901528";

int str_len = LoRaData.length() + 1; //Length of string plus 1 for end.
char parseInput[str_len]; //make char array to feed strtok

//Copy string into char array because strtok needs it.
LoRaData.toCharArray(parseInput, str_len);


float jlatitude = 0.0;
float jlongitude = 0.0;

Serial.println("Before");
Serial.println(jlatitude,6);
Serial.println(jlongitude,6);

char * partOfString; //Used by strtok() as an index.

partOfString = strtok (parseInput, ","); // Creates tokens using , delimiter.
jlatitude = atof(partOfString);     // Converts first part to float.

partOfString = strtok (NULL, ","); // Continues where the previous strtok left off.
//Serial.println(partOfString);
jlongitude = atof(partOfString);   // Converts second part to float.

  Serial.println("after");
  Serial.println(jlatitude,6);
  Serial.println(jlongitude,6);

  delay(3000);

}
```

- Next up I placed this into the simple receiver, and it functioned as desired, parsing the data as expected.
- Placing this into the main section of the code for the LoRa receiver sending multiple fields to ThingSpeak, it functions as desired. The code is long, so it is shown on the following page.

- August 19, 2019
  - Mapping the GPS and RSSI Data
    - Google offers custom maps through Google My Maps ( https://www.google.com/mymaps ) that you can make yourself using a spreadsheet, or many other data types. Exporting the GPS data from ThingSpeak as a csv file and importing it to Google Sheets, it recognized the format and converted it nicely into a spreadsheet. I labelled the columns and then after creating a new map in My Maps selected import, and then pointed it to the Google Sheets file in my Google Drive. After confirming which columns were longitude and latitude, and labeling them with the time and date column, I then set the points to be grouped (click the little paint roller “Style by” option) by the RSSI column. This creates a nice map with colour coded points for the RSSI values.
    - This map will be embedded on the GPS Data page of the website.

```
/* John Iddon - LoRa receiver sends data to ThingSpeak.
Uses code from examples of the LoRa.h library by Sandeep Mistry.*/
#include "ThingSpeak.h" //Includes the functions used for uploading to ThingSpeak.
#include <ESP8266WiFi.h> //Allows for usage of the ESP8266's WiFi modem.
#include <SPI.h>
#include <LoRa.h> //Includes LoRa.h library by Sandeep Mistry

//Pin definitions.
#define ss 15 //defines slave select (SS) pin, AKA chip select.
#define rst 14 //defines reset pin.
#define dio0 5 //defines dio0 pin.

char ssid[] = "**********"; //your network SSID (name)
char pass[] = "**********************"; //your network password

int status = WL_IDLE_STATUS; //Used for connection loop.
WiFiClient  client; //Creates client for connection.

unsigned long myChannelNumber =*************;  //Number for ThingSpeak channel.
const char * myWriteAPIKey = "********************";  //API key for my Thingspeak channel.

void setup()
{
  //Start of WiFi configuration.
  WiFi.mode(WIFI_STA); //Forces esp8266 into station mode.
  WiFi.begin(ssid, pass); //Opens wifi connection to your network.
  WiFi.mode(WIFI_STA); //Forces esp8266 into station mode.

  Serial.begin(115200); //Open serial port.
  while (!Serial);
  Serial.println("LoRa Receiver");

  while (WiFi.status() != WL_CONNECTED)
  { //prints dots on the serial monitor until wifi connects
      delay(500);
      Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());  //Prints the IP address the module picks up via DHCP

  ThingSpeak.begin(client);
  //End of Wifi configuration.

  //Start of LoRa configuration.
  LoRa.setPins(ss, rst, dio0); //set pins
  //433E6 for Asia
  //866E6 for Europe
  //915E6 for North America
  while (!LoRa.begin(915E6)) //sets chip frequency. 915MHz is what's legal in NA without being a HAM.
  {
    Serial.println("b."); //test at this point in program to ensure functionality on serial monitor.
    delay(500);
  }
  // SyncWord helps keep other LoRa traffic out by ensuring only messages with the same SyncWord are output/used.
  // Ranges from 0-0xFF
  LoRa.setSyncWord(0xF3);
  Serial.println("LoRa Initializing OK!");
  //End of LoRa configuration.
}

void loop()
{
  // Look to receive any packets if they're available.
  int packetSize = LoRa.parsePacket();
  if (packetSize)
  {
    // if packet has any size to it AKA exists.
    Serial.print("Received packet ");

    //Display the packet to serial monitor.
    //while (LoRa.available())
    //{ //While loop not needed since using readString rather than read.
      String LoRaData = LoRa.readString();
      Serial.println(LoRaData);


      int str_len = LoRaData.length() + 1; //Length of string plus 1 for end.
      char parseInput[str_len]; //make char array to feed strtok

      //Copy string into char array because strtok needs it.
      LoRaData.toCharArray(parseInput, str_len);


      float jlatitude = 0.0;
      float jlongitude = 0.0;

      Serial.println("Before");
      Serial.println(jlatitude,6);
      Serial.println(jlongitude,6);

      char * partOfString; //Used by strtok() as an index.

      partOfString = strtok (parseInput, ","); //Creates tokens using , delimiter.
      jlatitude = atof(partOfString);     //Converts first part to float.

      partOfString = strtok (NULL, ","); //Continues where the previous strtok left off.
      //Serial.println(partOfString);
      jlongitude = atof(partOfString);   //Converts second part to float.

      Serial.println("after");
      Serial.println(jlatitude,6);
      Serial.println(jlongitude,6);

      ThingSpeak.setField(1, jlatitude); //sets latitude to field 1
      ThingSpeak.setField(2, jlongitude); //sets longitude to field 2
      ThingSpeak.setField(3, LoRa.packetRssi()); //sets RSSI to field 3

      ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey); //sends all fields to ThingSpeak
      Serial.println("Sent 3 fields to ThingSpeak.");
    //}
    Serial.print("Packet RSSI "); // Print RSSI for packet to serial monitor.
    Serial.println(LoRa.packetRssi());
    delay(15000); //15 second delay before another packet
    //can be sent to ThingSpeak due to their rules.
  }
}
```