2014

Wireless Data Transmission

At the Greenbelt Environmental Monitoring Station

Members:

Andrew Saunders
Tuanlihn Nguyen
Justin Peterson
Jian (Jimmy) Xu

University of North Texas
Spring 2014
4/29/2014

Table of Contents

Abstract

The University of North Texas (UNT) and Texas Parks and Wildlife Department (TPWD) have been working together for over two years to improve environmental monitoring in the parks of north Texas. Currently, researchers must travel to these monitoring stations to retrieve the data. Our objective is to enhance this process by implementing a wireless network that will allow researchers to download environmental data from any location in the world.

Introduction

There is an Environmental Monitoring Station (EMS) located at The Texas Environmental Observatory (TEO) on the Greenbelt Corridor. The Greenbelt, located at the Ray Roberts Lake State Park, currently collects data continuously and stores it in the memory of a data-logger and single board computer. This data contains information about soil moisture and the weather inside of the state park.

Currently, a researcher must go out to the station to collect data manually, which is time consuming and inefficient. Our goal is to create a wireless connection between the Natural Heritage Center (NHC), located at the edge of the state park, and the EMS. The previous team that worked on this project attempted a line of sight wireless connection between the two locations, but realized that the trees next to the EMS were too tall to make the connection without damaging the ground to install a very tall tower.

Our goal is to use a small number of low power transceivers called Moteinos to create a connection through multiple nodes located beneath the tree tops from the EMS to a nearby relay station. We will install a short tower and utilize the Nanostation M2, a higher power outdoor broadband antenna, at both the relay site and the NHC to complete the connection. Once the connection is made we plan to connect the wireless system to the internet to allow a researcher to download the data from any location.

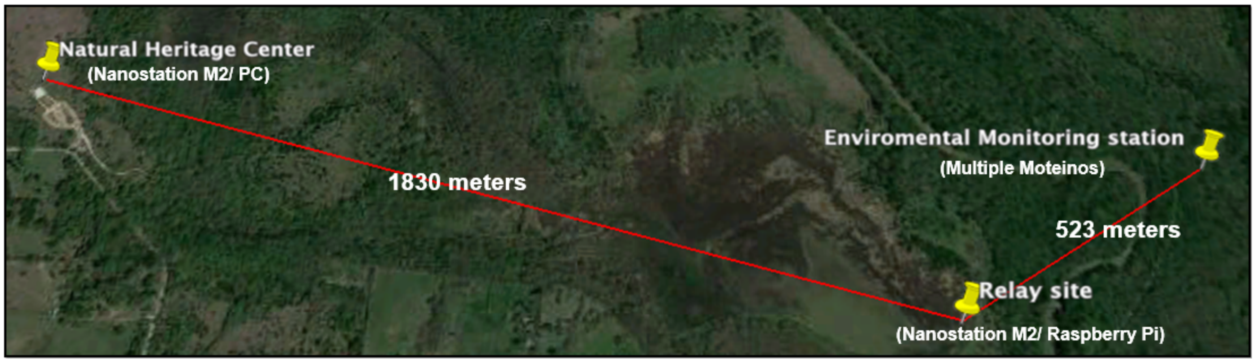


Figure 1: Map overview of our data transmission path.

Design Structure

The components we plan to use in our design include two Nanostation M2s, 7-10 Moteinos with attached radios, two solar panels with set-up equipment, one Raspberry Pi, and the router located at the NHC. The image in figure 2 describes our system and contains appropriate descriptions of each component.

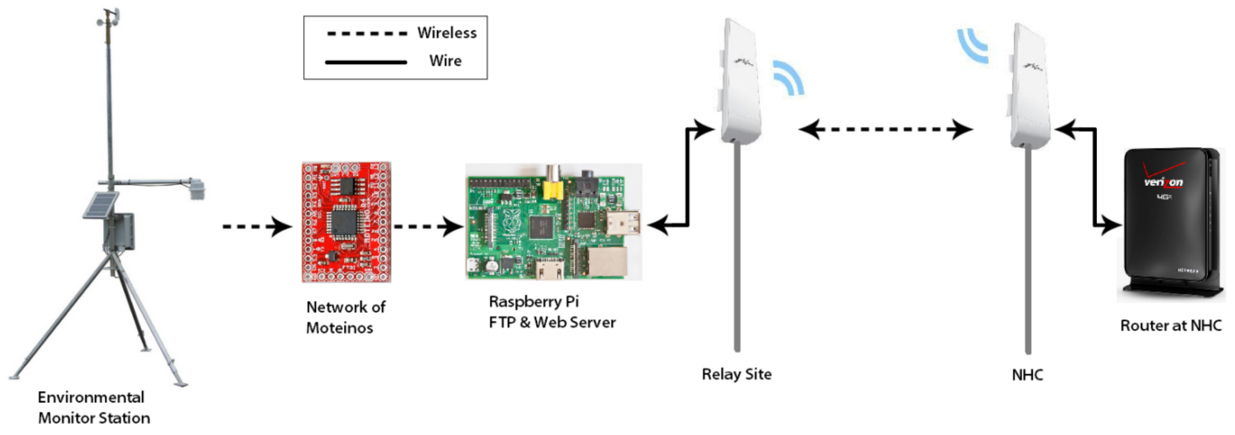


Figure 2: General design of our system, including distinction between wired and wireless sections.

We expect to spend a significant amount of time testing tower and Moteino locations in order to obtain optimal signal power. There is a tree line located near NHC which will require us to be very careful about where we place the towers to obtain the best line of sight without sacrificing the aesthetic integrity of the park. Setting up the Moteinos will provide an additional challenge since we will be dealing with thick foliage in certain locations and a river between the relay site and the EMS. Figure 2 depicts an example of the type of foliage we plan to deal with.

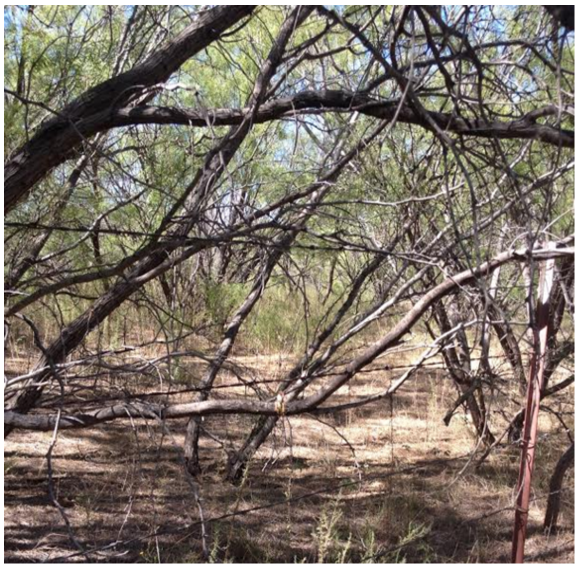


Figure 3: Example of the type of foliage under the trees.

Tower Design

We used identical telescoping, steel pole towers for each of the locations. The tower at the NHC is supported by a steel tripod at the center point and 3 additional steel cables attached from the top of the tower to the ground around it. Each tower is stabilized at the base with a 5 gallon bucket filled with cement. Each tower also has a Nanostation M2 attached to the top with Ethernet cables running alongside the tower to the ground.

The towers are located approximately 1.8 km apart and are 9m in height. In order for the Nanostations to send a clean signal, they must have a dB fade margin lower than -96 dB.

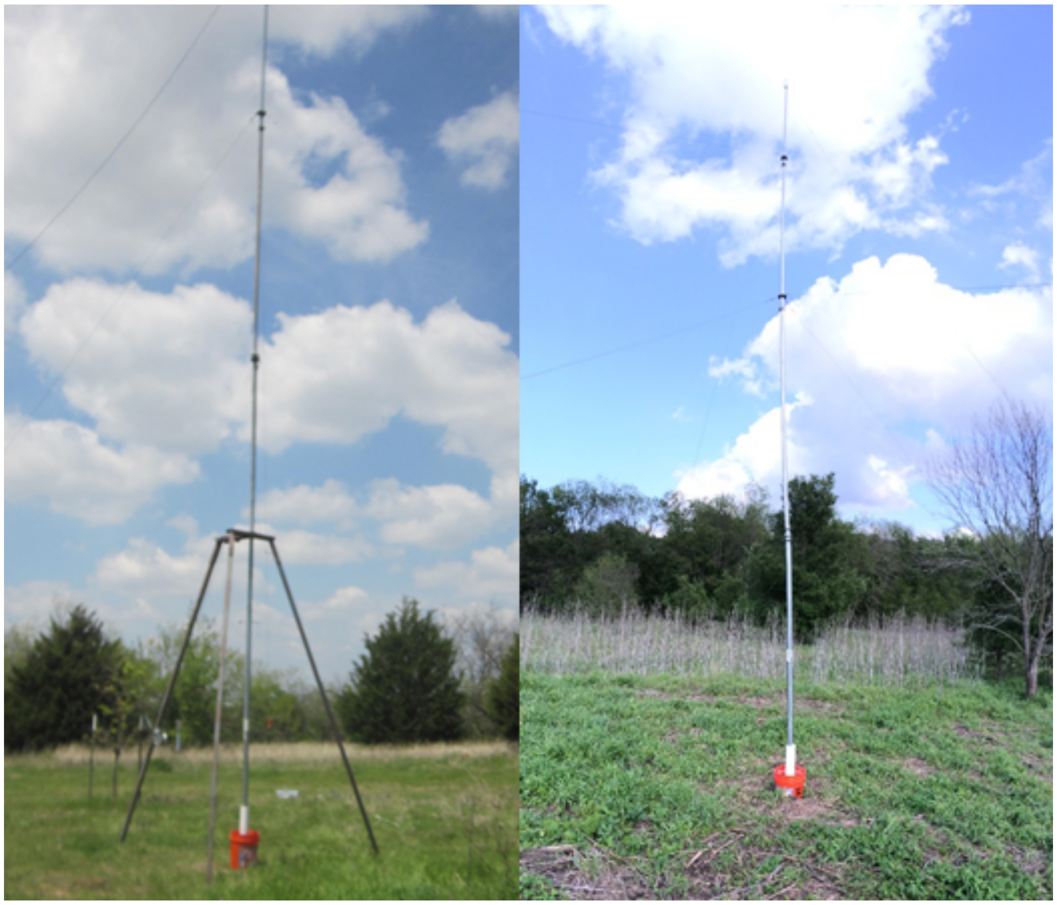


Figure 4: Tower at the NHC (left) and tower at the relay site (right).

Tower Testing

The process of finding locations for each tower was very time consuming. We examined the topography very closely in order determine the ideal locations for each tower. Each location between the relay site and the NHC had to be changed multiple times to find the optimal location combination. This process included constructing and deconstructing the tower multiple times in several locations.

Since each Nanostation has its own IP address, we could connect a computer at each location and use the AirOS software designed for the Nanostation M2 to observe our signal strength and dB fade.

Moteino Design

The connection between the relay station and the EMS is made by a series of Moteinos. We chose the Moteino in our design because it takes the concept of an Arduino and condenses it into a smaller, low power design. This design is ideal for our setup since each Moteino will be powered by a non-recharging external power source.

Each Moteino is equipped with an RFM69W, 915 MHz transceiver that will transmit and receive data. The radio appears as a small green circuit board which can be seen in figure 5 on the back of the Moteino. The Moteino requires an FTDI adaptor to connect it to a computer in order to edit the code. Once the code has been written and uploaded, the FTDI adaptor can be removed and the Moteino can be connected to an external power source and acts as a stand-alone processing unit. This design allows for very low power consumption compared to the Arduino, since the Moteino is not powering the FTDI adapter unnecessarily.

The Moteino takes very little power and can stay online with 3 AA batteries for approximately 5 weeks with our current design. Since we only need to send data every 15 minutes, we have implemented a sleep schedule to save power for most of the down-time. Currently, each Moteino sleeps for 14 minute and 50 seconds before waking up to pass data through.

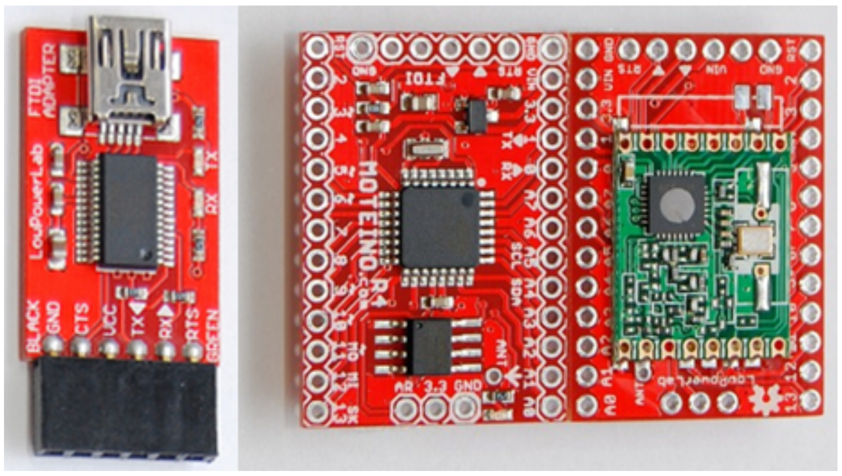


Figure 5: Image of the FTDI Adaptor (left) and Moteino front (center) and back (right).

Moteino Testing

Upon testing the Moteinos in the foliage near the relay site location, we measured varied distances between 150 to 350 feet between each node. The measured distance between the relay site and the EMS is close to 520 meters when taking a direct path. The changes in foliage and the river between the locations will not allow us to use the shortest possible path. By using the surface of the river as an open line of sight, we can situate Moteinos along the riverside to take a more efficient path to the EMS.

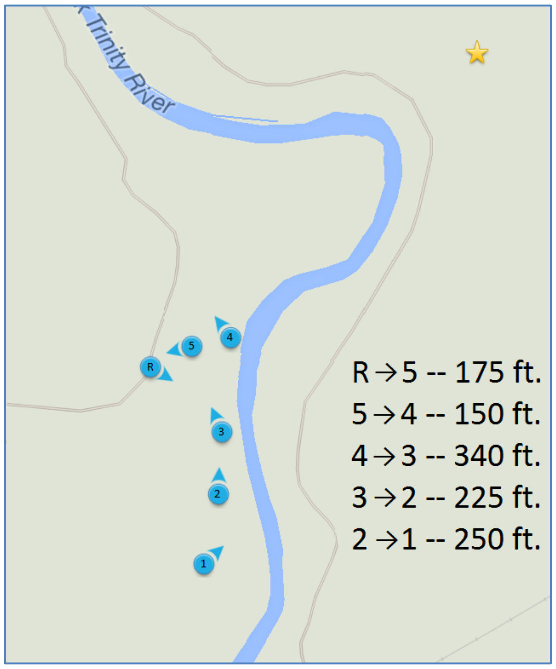


Figure 6: Testing locations for each Moteino and distances between each node.
The star represents the location of the EMS.

Raspberry Pi Implementation

The Raspberry Pi is a single-board minicomputer that we are using at our relay site. This computer allows us to complete the connection between the Moteinos and the Nanostation M2s. It also lets us store the data that is being transmitted to it every 15 minutes. The main advantage of using this device is that we can set up an FTP server. This will allow us to access the Raspberry Pi through the internet and edit, change, download, and upload any data located on the device. This is one of the most important aspects of our design.

We connected the Nanostation M2 at NHC to a router, which allows us to forward the IP addresses to the public. This means that we can connect to the Raspberry Pi at the relay site through the network at NHC from anywhere in the world. The Raspberry Pi is included in the NHC's wireless network because the Nanostations extend the signal to the relay site. The Raspberry Pi includes a Secure Shell for the FTP server and a required login ID and password needed to access it.

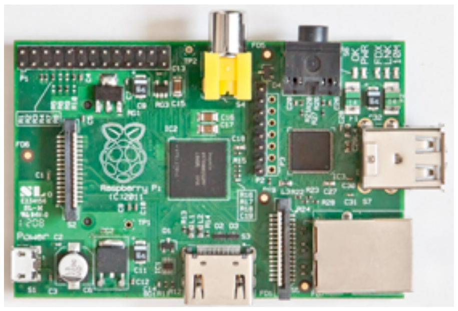


Figure 7: The Raspberry Pi.

Relay Site Power

We purchased two solar panels to install at the relay site tower. We took advantage of implementing a harmless renewable power source since the tower is located in a field. The solar panels come with a controller that moderates the power between the panels, the battery, and the load. In this case, the load includes a Nanostation and a Raspberry Pi. The calculated power consumption for each of these devices can be seen in figure 9.

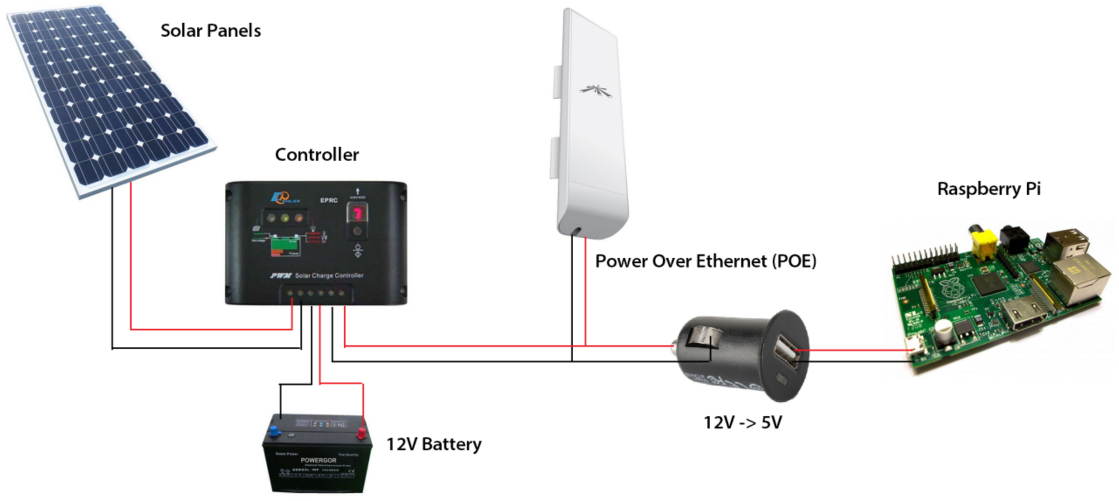
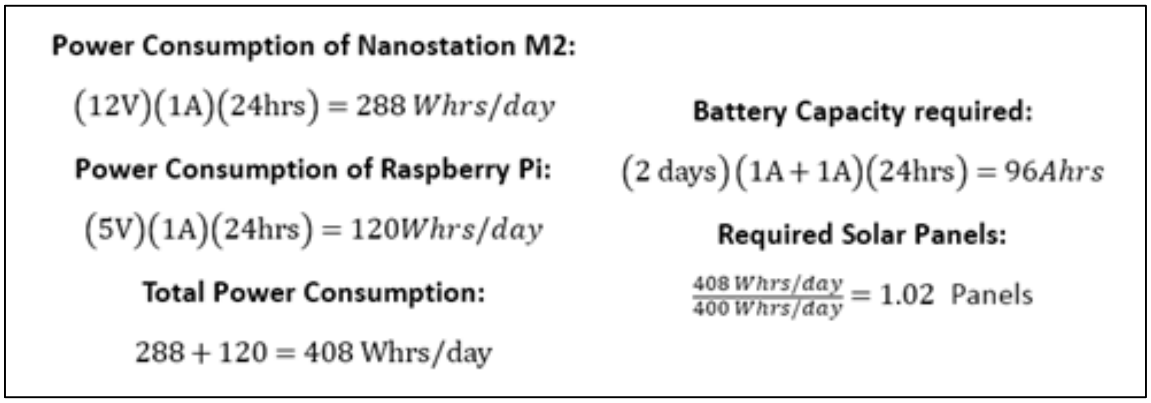


Figure 8: Power diagram at the relay site.

The controller is designed such that it splits the responsibilities of the solar panel and the battery. The solar panel is solely responsible for charging the battery and the battery is solely responsible for providing power to our system. When the battery is at full capacity, it will provide power to our system and the solar panels will simply act as a trickle charger. If the battery falls below a capacity threshold, the controller is designed to cut power to our system until the solar panel is able to recharge the battery. The controller does not allow the solar panel to directly power our system.



Power Consumption of Nanostation M2:

$$(12\text{V})(1\text{A})(24\text{hrs}) = 288\ Whrs/day$$

Power Consumption of Raspberry Pi:

$$(5\text{V})(1\text{A})(24\text{hrs}) = 120 Whrs/day$$

Total Power Consumption:

$$288 + 120 = 408\ \text{Whrs/day}$$

Battery Capacity required:

$$(2\ \text{days})(1\text{A} + 1\text{A})(24\text{hrs}) = 96Ahrs$$

Required Solar Panels:

$$\frac{408\ Whrs/day}{400\ Whrs/day} = 1.02\ \ \text{Panels}$$

Figure 9: Power consumption and required power measurements.

Design Considerations

There are several design considerations for each system of the project. Engineering standards, constraints, and various other aspects of the design are important to take into account before a final product is implemented.

Engineering Standards

It is important to consider what type of standards the devices use. Standards help dictate what frequencies are used for transmission in order to ensure a consistent use of frequency bands throughout the United States. For example, the Nanostations use IEEE 802.11n and IEEE 802.11g on 2.4 GHz. These standards mean that the devices use an OFDM based transmission with multiple-input multiple-output (MIMO). The IEEE 802.11 standards operate at a data rate between 54 Mbit/s to 600 Mbit/s, which is more than enough for our system.

The Moteinos are based on the IEEE 802.15.4 standard at 915 MHz. As long as we transmit under 50 mV per meter, we are within the constraints of the standard. We are transmitting well below this threshold so are well within compliance.

Realistic Constraints

Constraints restrict the freedom in designing the system. For example, we are not permitted to penetrate the ground more than 6 inches when setting up our towers. This particular constraint is why the original design idea of transmitting the signal directly from the EMS to the NHC was unachievable. A much taller tower would have to be installed to make that design work. Placing a tall tower requires the base to be situated deep below ground level and will result in a much larger diameter for the steel cable supports.

The land belongs to the TPWD and is regulated to minimize as much artificial invasion as possible. Our system is deemed appropriate because it facilitates in monitoring the well-being of the park. We made every effort to minimize the intrusiveness of our design by using small towers and avoiding a wired system.

An additional constraint involves powering our Moteinos. Even though our design implements very lower power consumption, we must power each node with batteries because there is very little sunlight beneath the tree tops to facilitate the use of solar panels. Each Moteino is stand-alone and so compact that we only require a 4 foot steel pole and a small box for each node. There are no additional components required.

Ethical, Professional, and Contemporary Issues

As engineers, we had to think about various issues that may arise during this project. If the team built a tower without any concern for the constraints, the tower could become a safety hazard for everything around it. One solution to this problem is to build a barrier around the tower so that nothing can approach the tower. Unfortunately, this would also introduce more artificial structures into the design. Thus, the best approach is to use steel cables and concrete to stabilize the tower more than necessary to ensure the safety of everything around it.

Conclusion

The full cover forest conditions and the restrictions instilled by the state park made this project a challenge. We were ultimately successful in designing the system and providing access to the Raspberry Pi through the internet. Currently, the system provides a connection between the NHC and the relay site, where a computer is driven by a renewable power source. Furthermore, the system is able to reach the EMS by using Moteinos beneath the tree tops. In the future, another team will have to update the system at the EMS to include wireless data gathering and connect it to our system. Once this is completed, the whole system will be finished and will only require maintenance and future upgrades as technology develops.

References

GPIO Serial connection:
http://blog.oscarliang.net/raspberry-pi-and-arduino-connected-serial-gpio/

Python:
http://www.classthink.com/2013/09/15/getting-started-python-raspberry-pi/

FTP web server:
http://tinkernut.com/wiki/page/Episode_320

Raspberry Web Server:
http://raspberrywebserver.com/

Moteino
http://lowpowerlab.com/shop/moteino-r4

Nanostation M2
http://dl.ubnt.com/datasheets/nanostationm/nsm_ds_web.pdf

Appendix

Moteino Code – End location

```
#include <RFM69.h>
#include <SPI.h>
#include <SPIFlash.h>

#include <Wire.h>
#include "RTClib.h"
RTC_DS1307 RTC;

#define NODEID      99
#define NETWORKID   55
#define GATEWAYID   77  //send to test monitor
#define FREQUENCY   RF69_915MHZ //Match this with the version of your
Moteino! (others: RF69_433MHZ, RF69_868MHZ)
#define KEY         "thisIsEncryptKey" //has to be same 16
characters/bytes on all nodes, not more not less!
#define LED         9
#define SERIAL_BAUD 115200

int counter = 0;

RFM69 radio;
SPIFlash flash(8, 0xEF30); //EF40 for 16mbit windbond chip
bool promiscuousMode = false; //set to 'true' to sniff all packets on the
same network

typedef struct {
  float         temp1;   //temperature maybe?
  float         temp2;
  float         analog3;
  int           counter;
} Payload;
Payload theData;

void setup() {
  Serial.begin(SERIAL_BAUD);
  Wire.begin();
  RTC.begin();
  // following line sets the RTC to the date & time this sketch was
compiled
  //RTC.adjust(DateTime(__DATE__, __TIME__));
  delay(10);
  radio.initialize(FREQUENCY,NODEID,NETWORKID);
  //radio.setHighPower(); //uncomment only for RFM69HW!
  radio.encrypt(KEY);
  radio.promiscuous(promiscuousMode);
  char buff[50];
  sprintf(buff, "\nListening at %d Mhz...", FREQUENCY==RF69_433MHZ ? 433 :
FREQUENCY==RF69_868MHZ ? 868 : 915);
  //Serial.println(buff);
```

```
}

byte ackCount=0;
void loop() {



  if (radio.receiveDone())
  {

    if (radio.DATALEN != sizeof(Payload))
      Serial.print("Invalid payload received, not matching Payload
struct!");
    else
    {
      theData = *(Payload*)radio.DATA; //assume radio.DATA actually
contains our struct and not something else

      DateTime now = RTC.now();
      Serial.print(now.year(), DEC);
      Serial.print('/');
      if(now.month() < 10)Serial.print('0');
      Serial.print(now.month(), DEC);
      Serial.print('/');
      if(now.day() < 10){Serial.print('0');}
      Serial.print(now.day(), DEC);
      Serial.print(" , ");
      if(now.hour() < 10){Serial.print('0');}
      Serial.print(now.hour(), DEC);
      Serial.print(':');
      if(now.minute() < 10){Serial.print('0');}
      Serial.print(now.minute(), DEC);
      Serial.print(':');
      if(now.second() < 10){Serial.print('0');}
      Serial.print(now.second(), DEC);

      Serial.print(" , ");
      Serial.print(theData.temp1);
      Serial.print(" , ");
      Serial.print(theData.temp2);
      Serial.print(" , ");
      Serial.print(theData.analog3);
      delay(100);

      counter = counter + 1;
      theData.counter = counter;
      if (counter > 99){
      counter = 1;
      }

      radio.send(GATEWAYID, (const void*)(&theData), sizeof(theData));
    }

    Serial.println();
    Blink(LED,3);

  }
}

void Blink(byte PIN, int DELAY_MS)
{
  pinMode(PIN, OUTPUT);
  digitalWrite(PIN,HIGH);
  delay(DELAY_MS);
  digitalWrite(PIN,LOW);
}
```

Moteino Code – Receive

```
#include <RFM69.h>
#include <SPI.h>
#include "LowPower.h"

#define NODEID      2
#define NETWORKID   55
#define GATEWAYID   99
#define FREQUENCY   RF69_915MHZ
#define KEY         "thisIsEncryptKey"
#define LED         9
#define SERIAL_BAUD 115200

RFM69 radio;

bool promiscuousMode = false; //set to 'true' to sniff all packets on the
same network

typedef struct {
  float         temp1;   //RFM69W Build-in Temp
  float         temp2;   //Analog Temp (Outside)
  float         analog3; //Moister Sensor
  int           counter; //Numbers of Testing data
} Payload;
Payload theData;


void setup() {
  Serial.begin(SERIAL_BAUD);
  delay(10);
  Blink(LED,1000);

  radio.initialize(FREQUENCY,NODEID,NETWORKID);
  radio.encrypt(KEY);
  radio.promiscuous(promiscuousMode);
  char buff[50];
  sprintf(buff, "\nListening at %d Mhz...", FREQUENCY==RF69_433MHZ ? 433 :
FREQUENCY==RF69_868MHZ ? 868 : 915);
}

void loop() {
      radio.receiveDone(); //Put RFM69W in RX mode (listening mode)
      LowPower.powerDown(SLEEP_FOREVER, ADC_OFF, BOD_OFF);

      if (radio.receiveDone()){
      if (radio.DATALEN != sizeof(Payload))
            Serial.print("Invalid payload received, not matching Payload
struct!");
      else
          {
            theData = *(Payload*)radio.DATA; //assume radio.DATA actually
contains our struct and not something else

            Serial.print(theData.temp1);
            Serial.print(" , ");
            Serial.print(theData.temp2);
            Serial.print(" , ");
            Serial.print(theData.analog3);
            delay(100);
            radio.send(GATEWAYID, (const void*)(&theData),
sizeof(theData));
          }
      Serial.println();
      Blink(LED,3);
      radio.sleep();
      //Put Moteino to sleep mode for 14mins 50secs
      for(int x = 0; x <89; x++){
      LowPower.powerDown(SLEEP_8S, ADC_OFF, BOD_OFF);
      LowPower.powerDown(SLEEP_2S, ADC_OFF, BOD_OFF);
      }
     }
}

void Blink(byte PIN, int DELAY_MS)
{
  pinMode(PIN, OUTPUT);
  digitalWrite(PIN,HIGH);
  delay(DELAY_MS);
  digitalWrite(PIN,LOW);
}
```

Moteino Code – Send

```
#include <RFM69.h>
#include <SPI.h>
#include <SPIFlash.h>
#include "LowPower.h"

#define PHS          5   // photosensor output pin 5
#define PHS1         6
int analogPin = 5;
int analogPin1 = 6;
int PHSREAD = 0;
int PHSREAD1 = 0;

#define NODEID      1
#define NETWORKID   55
#define GATEWAYID   2
#define FREQUENCY   RF69_915MHZ //Match this with the version of your
Moteino! (others: RF69_433MHZ, RF69_868MHZ)
#define KEY         "thisIsEncryptKey" //has to be same 16
characters/bytes on all nodes, not more not less!
#define LED         9
#define SERIAL_BAUD 115200


RFM69 radio;

typedef struct {
  float         temp1;
  float         temp2;
  float         analog3;
  int           counter;
} Payload;
Payload theData;

void setup() {
  Serial.begin(SERIAL_BAUD);
  pinMode(PHS, OUTPUT);
  pinMode(PHS1, OUTPUT);
  radio.initialize(FREQUENCY,NODEID,NETWORKID);
  //radio.setHighPower(); //uncomment only for RFM69HW!
  radio.encrypt(KEY);
  char buff[50];
  sprintf(buff, "\nTransmitting at %d Mhz...", FREQUENCY==RF69_433MHZ ?
433 : FREQUENCY==RF69_868MHZ ? 868 : 915);
  //Serial.println(buff);
}


void loop() {

    byte temperature =  radio.readTemperature(-4); // -1 = user cal
factor, adjust for correct ambient
```

```
    theData.temp1 = 1.8 * temperature + 32; // 9/5=1.8
    digitalWrite(PHS1, HIGH);
    delay(50);
    PHSREAD1 = analogRead(analogPin1);

    int Vo; // Integer value of voltage reading
    float R = 9890.0; // Fixed resistance in the voltage divider
    float logRt,Rt,T,TF;
    float c1 = 1.009249522e-03, c2 = 2.378405444e-04, c3 = 2.019202697e-
07;
    PHSREAD1 = analogRead(analogPin1);
    Rt = R*( 1023.0 / (float)PHSREAD1 - 1.0 );
    logRt = log(Rt);
    T = ( 1.0 / (c1 + c2*logRt + c3*logRt*logRt*logRt ) ) - 273.15;
    TF = ((T*1.8)+32);
    digitalWrite(PHS1, LOW);
    theData.temp2 = TF;
    Serial.print(theData.temp1);
    Serial.print(" , ");
    Serial.print(theData.temp2);
    Serial.print(" , ");
    digitalWrite(PHS, HIGH);
    delay(50);
    PHSREAD = analogRead(analogPin);
    PHSREAD = map(PHSREAD,77,284, 0, 100);
    theData.analog3 = PHSREAD;
    Serial.print(theData.analog3);
    digitalWrite(PHS, LOW);

    radio.send(GATEWAYID, (const void*)(&theData), sizeof(theData));

    Serial.println();
    Blink(LED,3);


    radio.sleep();
    //for(int x = 0; x < 1; x++){
    for(int x = 0; x < 90; x++){
    LowPower.powerDown(SLEEP_8S, ADC_OFF, BOD_OFF);
    LowPower.powerDown(SLEEP_2S, ADC_OFF, BOD_OFF);
    }

  }


void Blink(byte PIN, int DELAY_MS)
{
  pinMode(PIN, OUTPUT);
  digitalWrite(PIN,HIGH);
  delay(DELAY_MS);
  digitalWrite(PIN,LOW);
}
```