UNIVERSITY OF ALASKA ANCHORAGE

CSCE A470

CAPSTONE PROJECT

Remote Sensing in the Arctic

Author:

Mike Mobley

Supervisor:

Prof. Martin Cenek, PhD

Anchorage AK, April 2016

Computer Science & Engineering Department
UNIVERSITY of ALASKA ANCHORAGE

mmobley4@alaska.edu

Version 5.3

Abstract

The purpose of this document is to describe, explain, and present the capstone project by Mike Mobley, Matthew Devins, and Lance Leber. The capstone project is titled Remote Sensing in the Arctic. The purpose of this project is to create a communications network capable of relaying sensor data back to a home base station for analyzation. This project is funded by the Department of Homeland Security as part of the Arctic Domain Awareness Center of Excellence.

Motivation: With climate change having substantial effects in the arctic region, increased accessibility will impact how these areas are used. Commerce, research, and access to natural resources will result in a noteworthy increase of human traffic. The ability to monitor the impact of this activity is vital to help determine the environmental impact of the increased use.

Problem Statement: The arctic region's vast size coupled with the diversity of potential conditions to monitor requires a low cost option to detect conditional changes or events and isolate their locations within a reasonable area.

Overview:

This project consists of two major parts, the communication network and the simulator. The communication network is designed to be self-healing and operate in a hostile environment. Each node will be attached to a sensor type based on the application the network is intended for. We will test our network with audio sensors. If the node's sensor detects an event, the node will append information to uniquely identify the event in a data structure and transmit the event to one of four readout nodes/stations for storage or analysis.

This project was developed in short iterations utilizing the agile methodology. as new features were introduced, they would get coded and tested then implemented. Because testing the network in a deployed environment was not feasible, we decided to test many of the features with agent based modeling software.

The software for the nodes was developed in a specialized IDE made for our chosen hardware devices. The language utilizes a derivation of C, C++, and some functionality is provided by third party libraries. The simulator software was developed using NetLogo IDE and software. This platform enabled the development of a potential triangulation method that does not require any synchronized communication.

Approach: We used a low-cost off the shelf programmable sensor with RF capabilities to construct a decentralized asynchronous network to detect a desired event or condition. The sensors propagate the information through the network to specially configured terminals to interpret the data. We also used agent based model simulation software to predict behavior and isolate potential problems.

Result: We have been able successfully configure the hardware and develop software that effectively and efficiently detects and routes data through the network. The limitations of the sensors and the asynchrony of the network has made location determination challenging. We are currently researching different possibilities that will allow us to isolate given events to a reasonable area within the network topology.

Conclusion: Successful conclusion of this research will provide a low cost option for event monitoring in the arctic region. The versatility of the sensors will enable adaptation to cover a variety of monitoring scenarios.

ACKNOWLEDGEMENTS

The completion of this capstone project was the result of the hard work and support of many different people. I would like acknowledge all of their help throughout the project.

First I would like to thank Dr. Martin Cenek. As project supervisor, his leadership and mentorship was vital to both my academic growth and ensured successful project progression. He always aimed to challenge the team while keeping expectations within scope of an undergraduate project. I have learned a great deal throughout this process that will stay with me for years to come.

Next, I would like to thank all of my of my group members, Matt Devins and Lance Leber for all of their hard work and determination to work together, meet frequently, and develop a high quality research project.

As this project represents the culmination of my undergraduate studies here at UAA, I would like to thank the professors and staff for their excellent instruction and willingness to go the extra mile to ensure my educational experience was both enjoyable and successful.

Additionally, I would like to thank the Department of Homeland Security who's funding and direction made this opportunity possible.

Finally, I would like to thank my dedicated family who supported me throughout this project as well as my entire educational experience. Without their love and support, it would not have been possible.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

This research aims to find a solution for an inexpensive, reliable, low power communication platform for sensor networks in the arctic and sub-arctic regions. This project is funded by the Department of Homeland Security as part of the Arctic Domain Awareness Center of Excellence (ADAC) [1]. This network needs to possess some attributes based on the location and environment it will be deployed in. Some of these attributes include: asynchronous operation, low power consumption, decentralized operation, redundancy, self-healing, limited localized processing, and flexible design. Today, remote sensing technologies are deployed from a variety of innovative platforms, such as satellites, airplanes, boats, and unpiloted aerial vehicles, terrestrial and underwater vehicles, as well as fixed and mobile distributed networks [2]. However, most of these networks are expensive and require significant infrastructure support for operations and maintenance. We will demonstrate a low cost reliable alternative that requires minimal support.

Our solution is to create a homogeneous peer-to-peer sensor network with the capability for each node within the network to self-discover the shortest path, based on the number of hops, to each of four edge nodes configured as readout nodes. This is done using a distant vector algorithm designed specifically for our sensor network. The nodes do not have global knowledge of the rest of the network nor are they aware of their own geo-location. Because the nodes are small and identical, deployment into a network grid can be easily accomplished using airplane, boat, all-terrain vehicle, or by foot. The sensor data that is collected will be limited only to the types of sensors available to connect to the network nodes themselves. This provides exceptional flexibility for connecting sensors that produce either a digital or analog output as well as multiple senor types simultaneously.

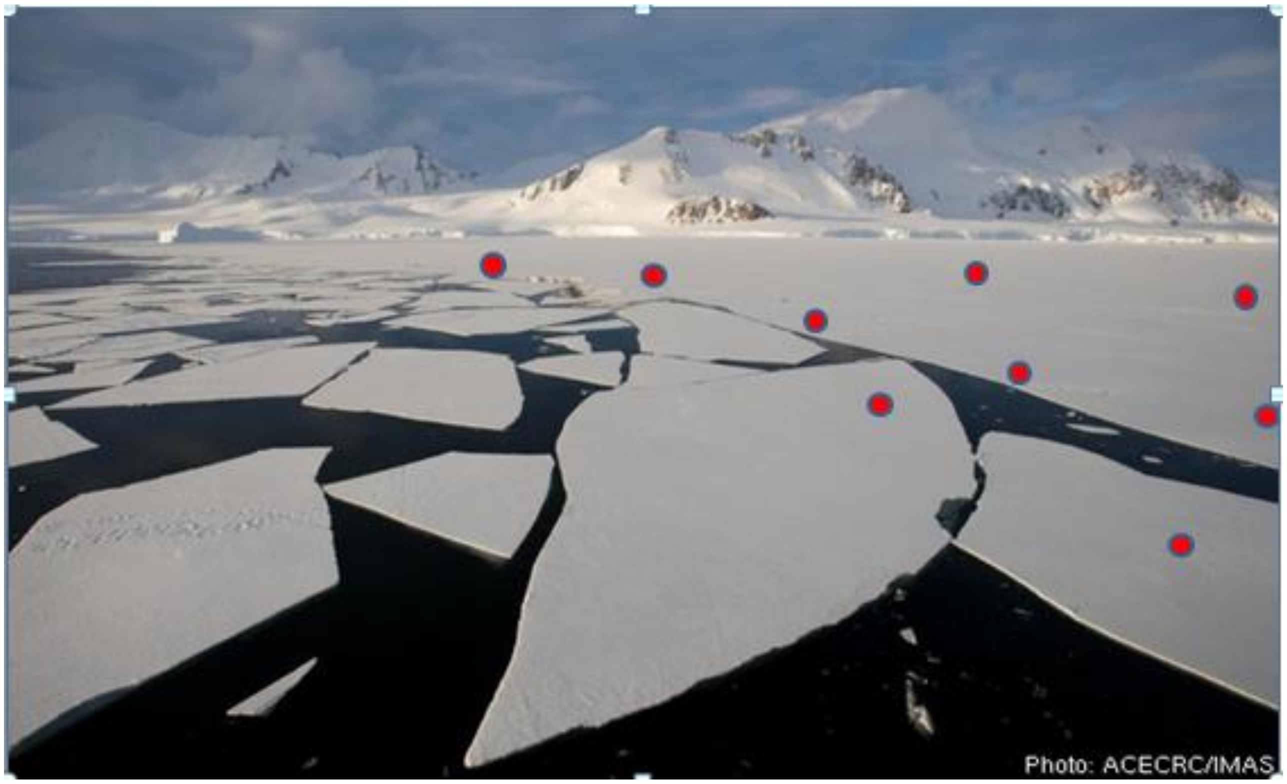


Figure 1.1 Conceptual application of the sensor network deployed to monitor and report ice sheet break up by sensing sound and/or movement within the ice sheet. Network nodes are simulated by red dots. This photo was altered to show conceptual design from a photo at: http://www.antarctica.gov.au/science/cool-science/2010/measuring-fast-ice-in-antarctica

1.2 Application

Our implementation uses a low-cost off the shelf programmable sensor with RF capabilities to construct a decentralized asynchronous network to detect a desired event or condition. The sensors propagate the information through the network to specially configured terminals known as readout sensors or nodes to interpret the data. This application will be tested in a controlled laboratory environment with sensor nodes laid out in a grid pattern. By testing the network in this manner, we will be able to quickly isolate bugs and refactor code as necessary. Additionally, by demonstrating network operation with so many sensors in close proximity, we can build a robust application that can handle interference and packet collisions with minimal affects. We also use agent based model simulation software to predict behavior and isolate potential problems. The agent based software used for this purpose will be NetLogo. The platform chosen for the sensor network platform is the Moteino with an integrated ultra-high frequency (UHF) transceiver embedded. The Moteino is used for application execution and sensor signal processing and the UHF transceiver provides connectivity links required for network operations.

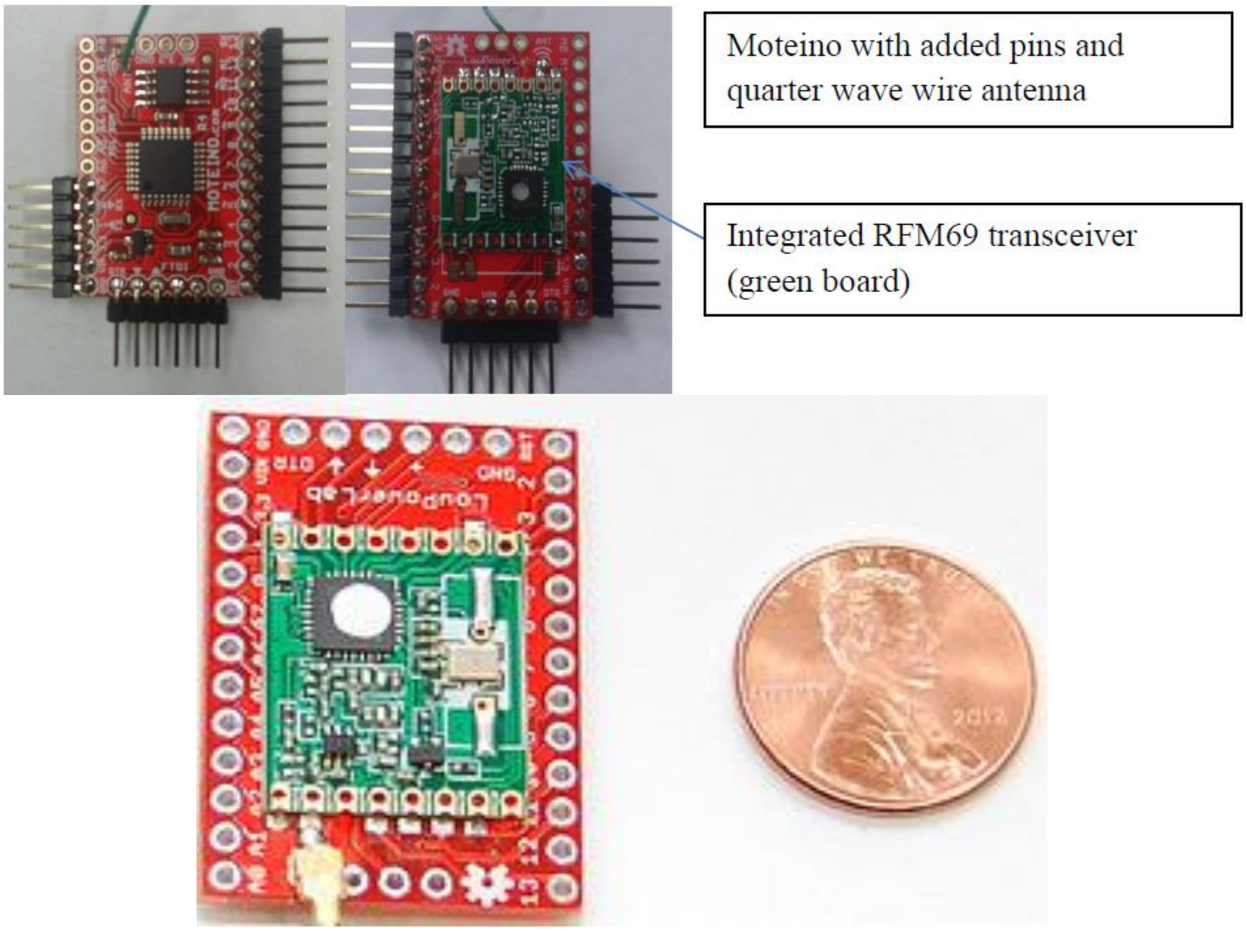

Moteino with added pins and quarter wave wire antenna
Integrated RFM69 transceiver (green board)

Figure 1.2 Moteino processor with embedded UHF transceiver.
http://lowpowerlab.com/moteino/#antennas

1.3 Motivation

With climate change having substantial effects in the arctic region, increased accessibility will impact how these areas are used. Commerce, research, and access to natural resources will result in a noteworthy increase of human traffic. The ability to monitor the impact of this activity is vital to help determine the environmental impact of the increased use. Additionally, today's security needs are an ever growing concern. With miles and miles of both land and maritime borders, the challenge of monitoring such vast areas with current techniques such as manned patrols or satellites is both expensive and limited in capabilities. Although illegal border crossings are usually thought of at the southern U.S. border the northern border is porous and vast and illustrates a significant security threat. Last year, in an attempt to slow the flow of illegal immigration from Canada, CBP (U.S. Customs Border Patrol) spent $20 million on a surveillance system that monitors 34 miles of the St. Clair River bordering Michigan and Canada—a popular destination for illegal immigrants crossing from Canada

[4].

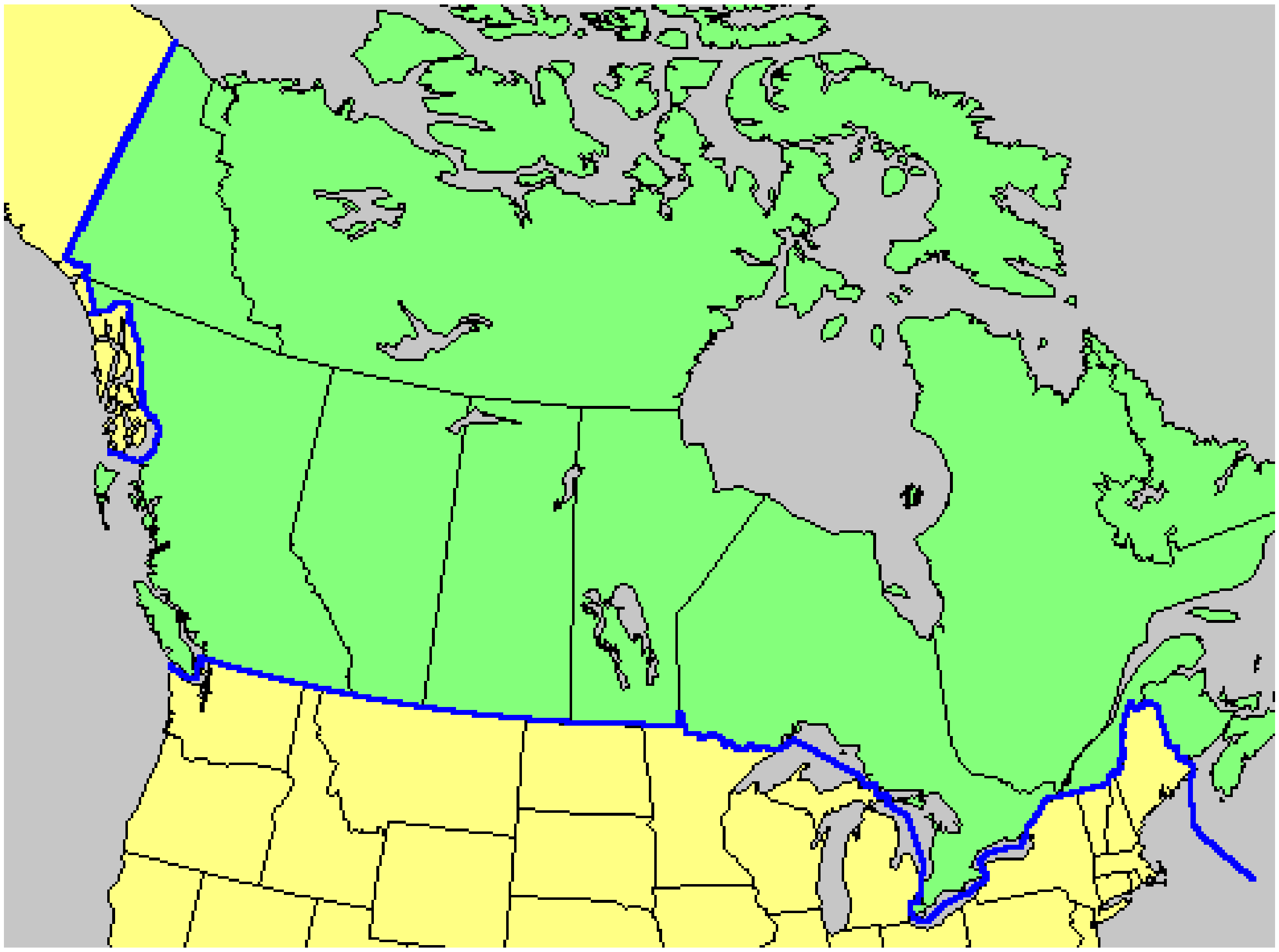


Figure 1.3 Image depicting the 5,525 mile U.S. Canadian border. Most of which is not actively monitored by security personnel. Image source: cars-memes.com

1.4 Recent Developments

With more and more powerful computing capabilities in an increasingly small and efficient package, remote sensing data has become more accessible and widespread in recent years. The exploitation of this technology has gone from developments mainly conducted by government intelligence agencies to those carried out by general users and companies. Sensing, extracting and correlating data poses quite the challenge in remote regions with limited infrastructure. For this purpose, high performance computing such as clusters, distributed networks or specialized hardware devices provide important architectural developments to accelerate the computations related with information extraction in remote sensing [5]. With well thought out software and a flexible hardware platform, highly complex and capable networks can be designed.

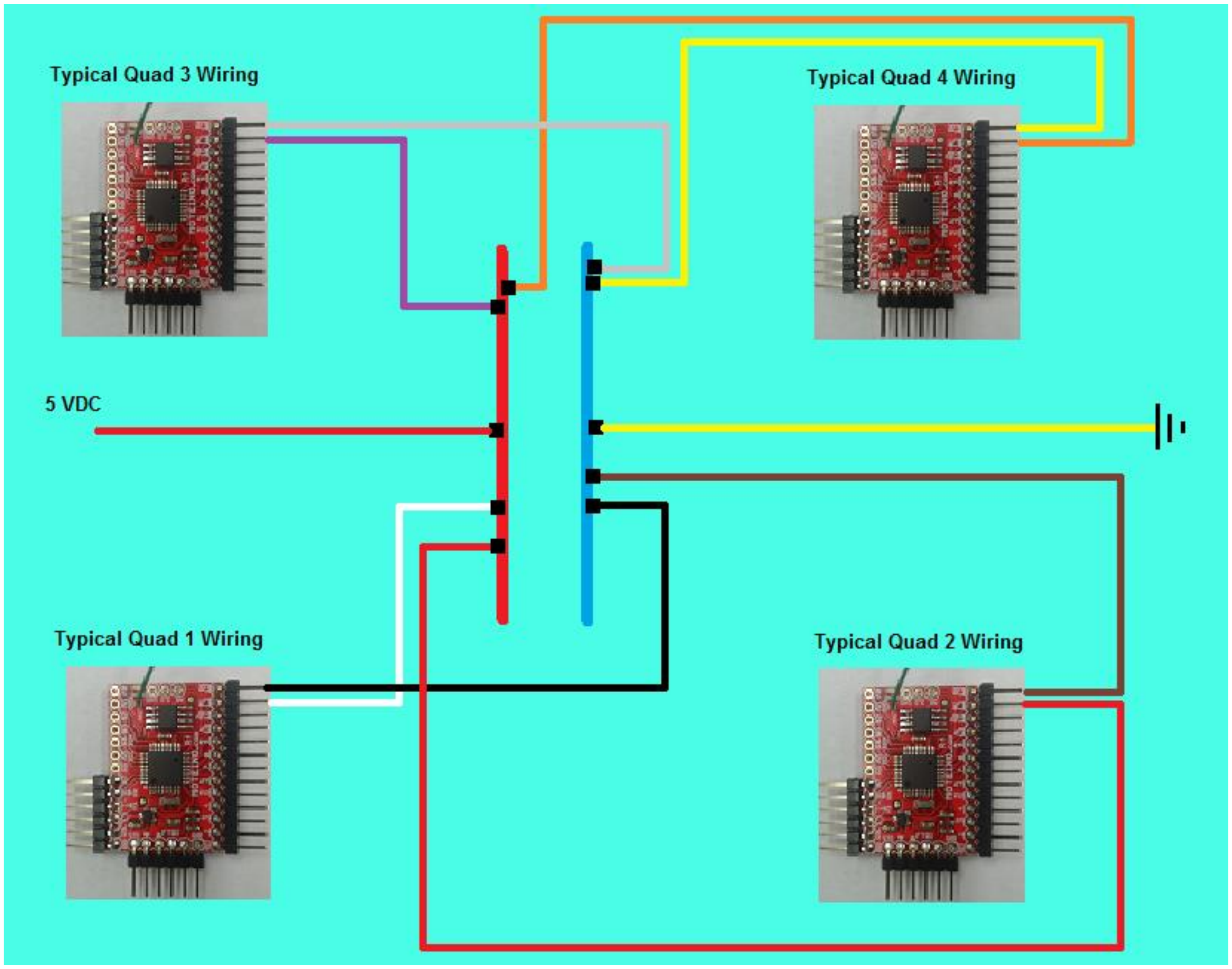


Figure 1.4 Initial wiring diagram design that depicts a basic network configuration in the lab. In deployment configuration, power and sensor inputs will not be shared and will be unique to each individual sensor.

Chapter 2

System Integration and Modeling

2.1 Technology Used for Implementation

As with any project idea and plans, decisions have to be made based on a number of considerations. A few of those considerations include cost, performance, suitability, and implementation options. For our project, we needed to consider both hardware and software. Since our project is to develop a communications infrastructure for a decentralized asynchronous sensor network that would be stable and robust enough to operate autonomously in the harsh arctic regions for long periods of time and with minimal maintenance, our first priority was to select the hardware that would serve as backbone to this project. The hardware that would serve as the network nodes to relay vital information of concern is the Arduino clones, the Moteino. Although this platform may not turn out to be suitable for final development, they will serve well as our proof of concept mechanism.

What is a Moteino? Moteino is a low cost low-power open-source wireless Arduino compatible development platform based on the popular ATMega328 chip used in traditional Arduinos, making it 100% compatible with the Arduino IDE (programming environment) [1]. One of the most attractive things about this platform for an undergraduate research project standpoint is the cost to performance ratio. The Moteino we chose is quite capable in a $20 package. Some of the specifications are [1]:

- Microcontroller – Atmega328
- Transceiver – RFM69
- Operating frequencies – 434MHz, 868MHz, 915MHz
- Pins – 14 digital, 8 analog
- Clock speed – 16MHz
- Flash memory – 32KB
- SRAM 2KB
- EEPROM – 1KB

In addition to the specifications, our choice for the Moteino was also based on a platform with such features as being small for ease of deployment with low visible and environmental footprint. Versatile design allows a wide range of sensor inputs for increased flexibility. Low power consumption along with the ability to turn on and off features in software maximizes power efficiency based on performance and deployment needs. The relatively long RF range increases communication reliability and reduces required number of nodes to monitor a given area. Finally, the ability to wirelessly program the devices makes for an attractive feature when considering the project’s deployment area.

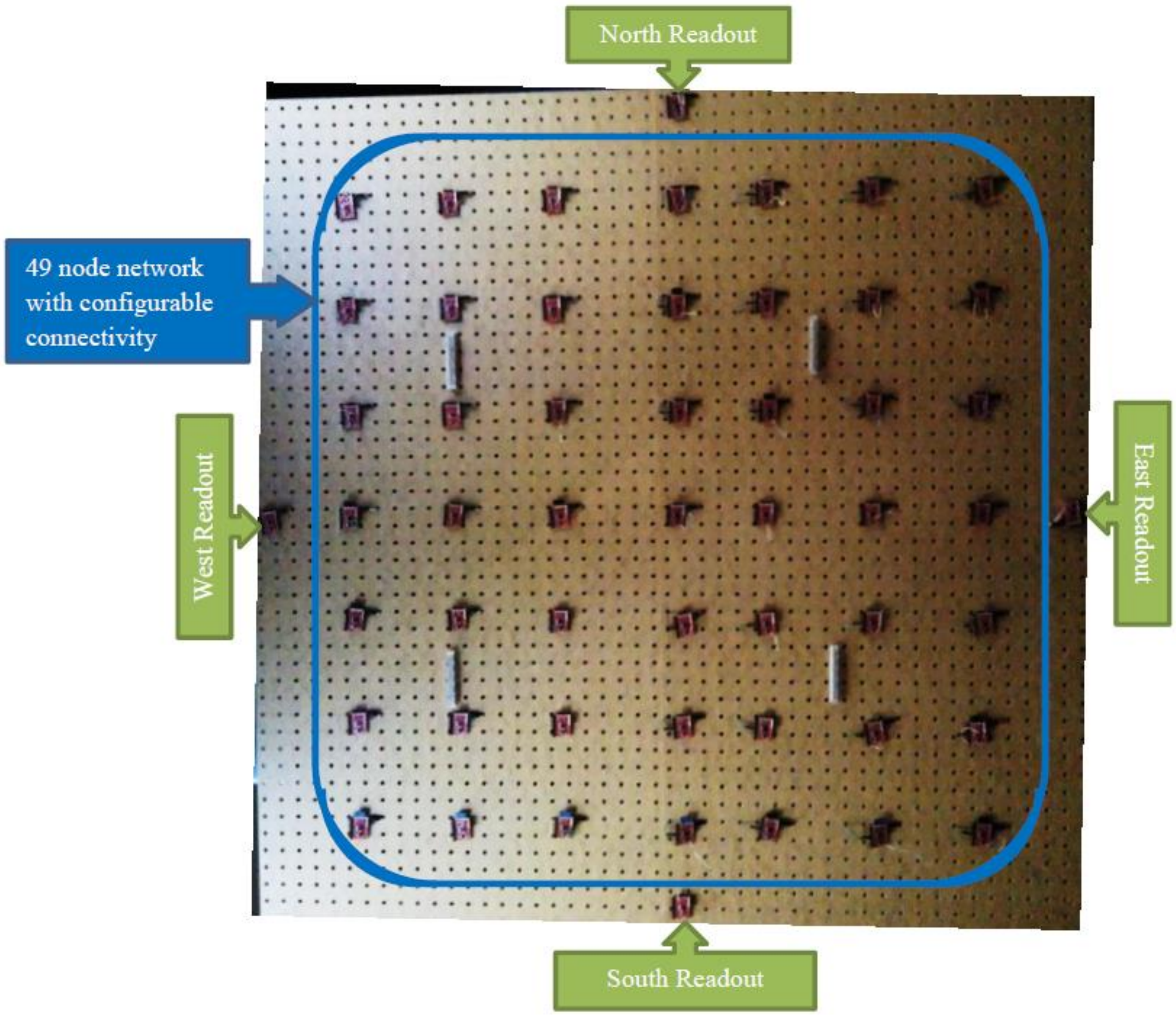
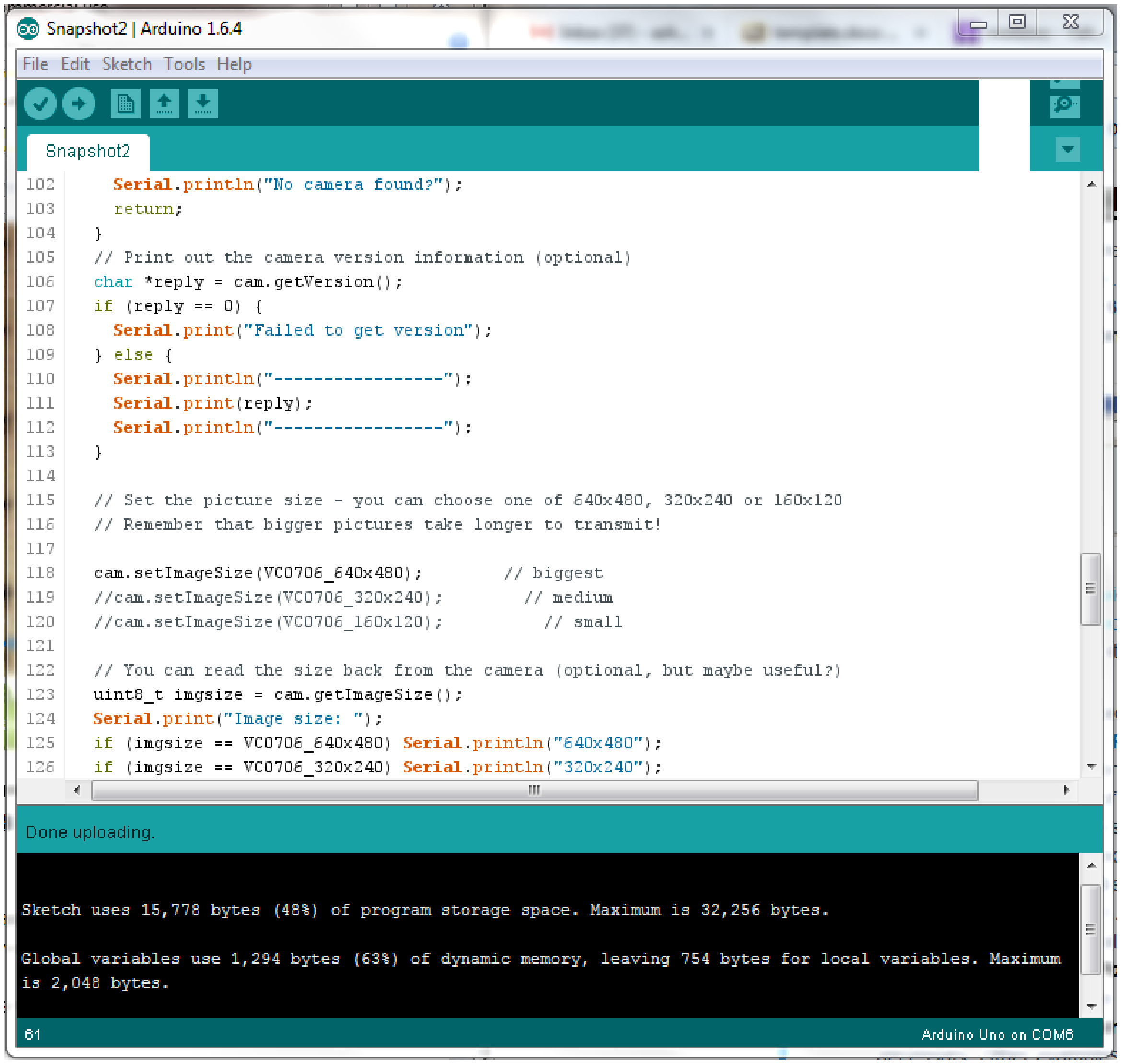


Figure 2.1 Matrix of 49 Moteinos in a 7x7 grid with 4 readout nodes. This configuration is used within the lab for software testing and demonstrations.

The selection of a design platform as well as a software language was an easy one. The Moteinos work well using the integrated development environment designed to support the Arduinos (the devices the Moteino was developed from). The IDE is free and can be downloaded from https://www.arduino.cc/ , and provides the functionality necessary to interface with the hardware via universal serial bus connectivity. For the software, we chose C/C++ for a couple of reasons. Moteino architecture is already set up to execute C code. Also, the amount of control with pointers and available data structures provided in the C language made it easy for us to configure the software to meet our needs. We use several provided libraries including those to help with serial I/O, radio functionality, sound processing, and camera interface.



```
    Serial.println("No camera found?");
    return;
  }
  // Print out the camera version information (optional)
  char *reply = cam.getVersion();
  if (reply == 0) {
    Serial.print("Failed to get version");
  } else {
    Serial.println("-----------------");
    Serial.print(reply);
    Serial.println("-----------------");
  }

  // Set the picture size - you can choose one of 640x480, 320x240 or 160x120
  // Remember that bigger pictures take longer to transmit!

  cam.setImageSize(VC0706_640x480);        // biggest
  //cam.setImageSize(VC0706_320x240);        // medium
  //cam.setImageSize(VC0706_160x120);          // small

  // You can read the size back from the camera (optional, but maybe useful?)
  uint8_t imgsize = cam.getImageSize();
  Serial.print("Image size: ");
  if (imgsize == VC0706_640x480) Serial.println("640x480");
  if (imgsize == VC0706_320x240) Serial.println("320x240");
```

```
Done uploading.

Sketch uses 15,778 bytes (48%) of program storage space. Maximum is 32,256 bytes.

Global variables use 1,294 bytes (63%) of dynamic memory, leaving 754 bytes for local variables. Maximum
is 2,048 bytes.

61                                                                    Arduino Uno on COM6
```

Figure 2.2 Screenshot of the Arduino IDE used to develop and load software on the Moteino boards.

The software architecture is like a typical C program with some variation. Due to the nature of the software running in a continuous loop and performing a very specialized function, the code is mostly one continuous file although inner classes are used for the linked-list implementation. Like many C programs include statements and global variables are declared at or near the top of the program. For the hardware to be configured and run properly, two required functions are required, setup() and loop(). As the name implies, the setup function is used to initialize variables and perform any hardware configuration the application may need. A few examples include the radio frequency, the pin configuration for I/O, serial baud rates, node identifiers etc. The loop function is equivalent to a main function in most software programs. The difference here is that the loop function repeats itself over and over until the Moteino is shut down. Because of this ever looping nature of the software, particular considerations must be taken into account when designing the software. Outside of the main loop, other developer defined functions can be written that are called at various points from within the main loop function.

2.2 Design View of System Architecture

The system architecture consist of a development machine, individual communication nodes, specially configured readout nodes, a stand-alone simulator for testing and analysis, and a "base station" used to collect, store, and analyze data from the network.

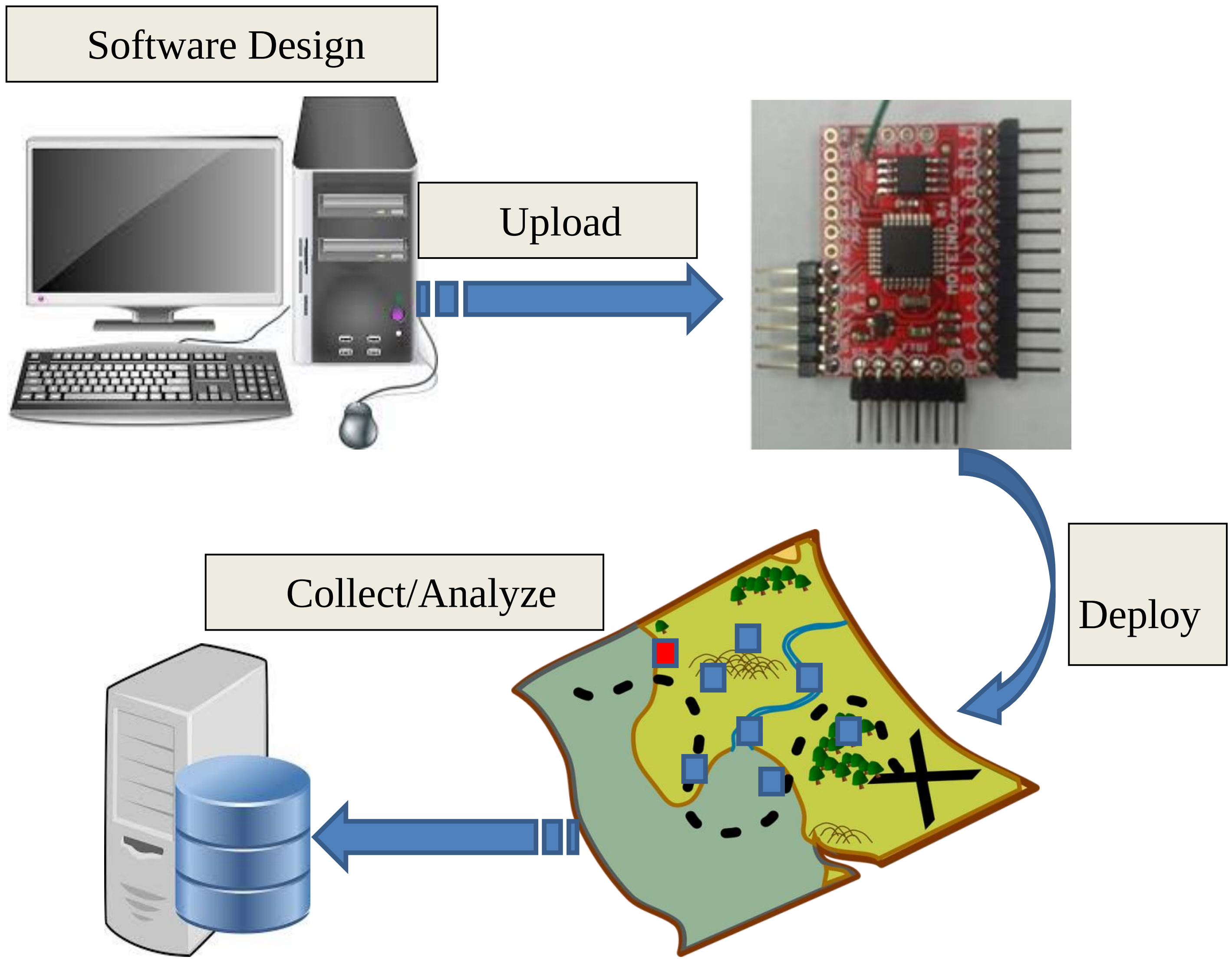


Figure 2.3 Conceptual architecture of project

The software architecture of the Moteino is a program that 1) initializes the parameters of the individual node, 2) completes neighbor discovery/shortest path calculations, 3) listens for and queues potential events, 4) shares information about events with neighbors, 5) receives potential events from neighbors, 6) analyzes event based on predefined parameters to determine event authenticity, 7) relays potentials events from neighbors, and 8) repeats steps 1-7.

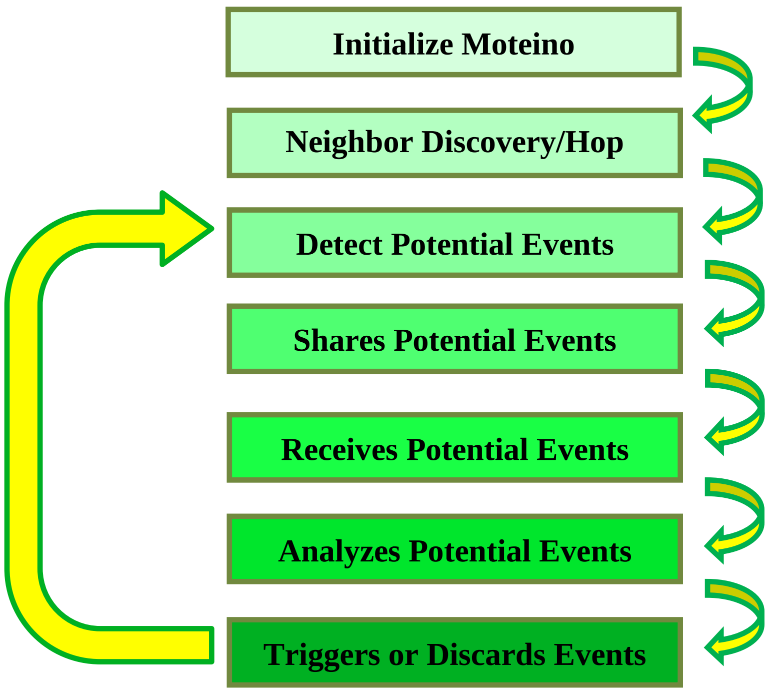
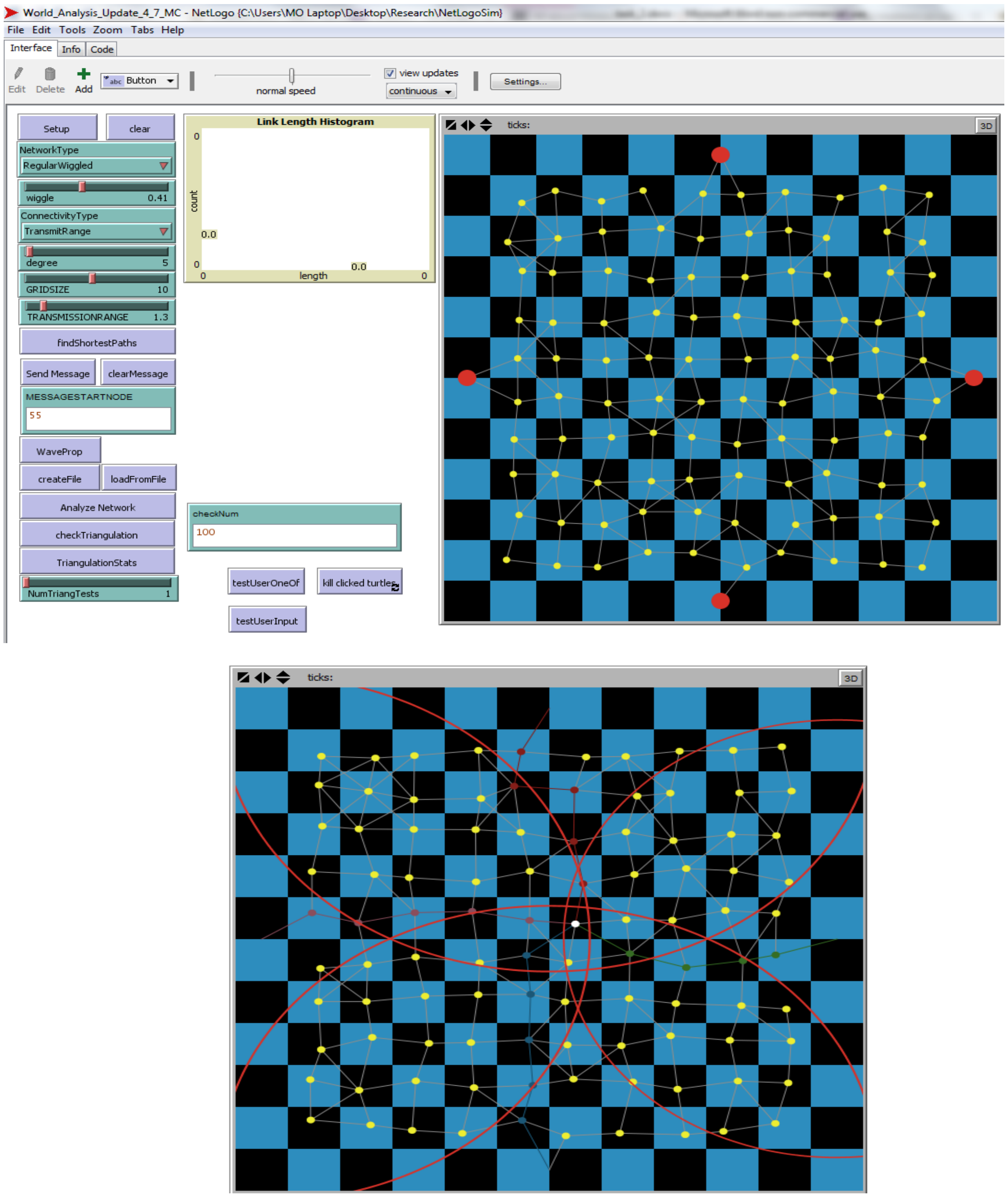


Figure 2.4 Basic software flow/architecture

2.3 Components Used in Project Development

Because the hardware and software basics have been discussed previously, in this section we will focus on the simulation software utilized to model the network behavior [6]. One of the challenges with this project is the terrestrial footprint required to test deployment and functionality. Because it is unrealistic to deploy and test the network after each software or hardware change, we produced a mockup of the network using the actual hardware attached to a peg board substrate in a 7x7 matrix (see Figure 2.1). This arrangement works well for much of the testing, however, we are dealing with 52 radios in a confined space all trying to talk to each other at the same time. Additionally, the nodes have no readout or screen printing capabilities as configured so it is not always apparent what the nodes' behavior is under certain conditions. We were able to overcome some of these congestion problems with a robust software solution but it did not solve all of our issues. We decided to use an agent based modeling simulation software to work in tandem with the hardware so we could observe behavior and simulate deployment behavior more closely in a controlled environment. This has enabled us to troubleshoot and add analysis tools such as event triangulation.

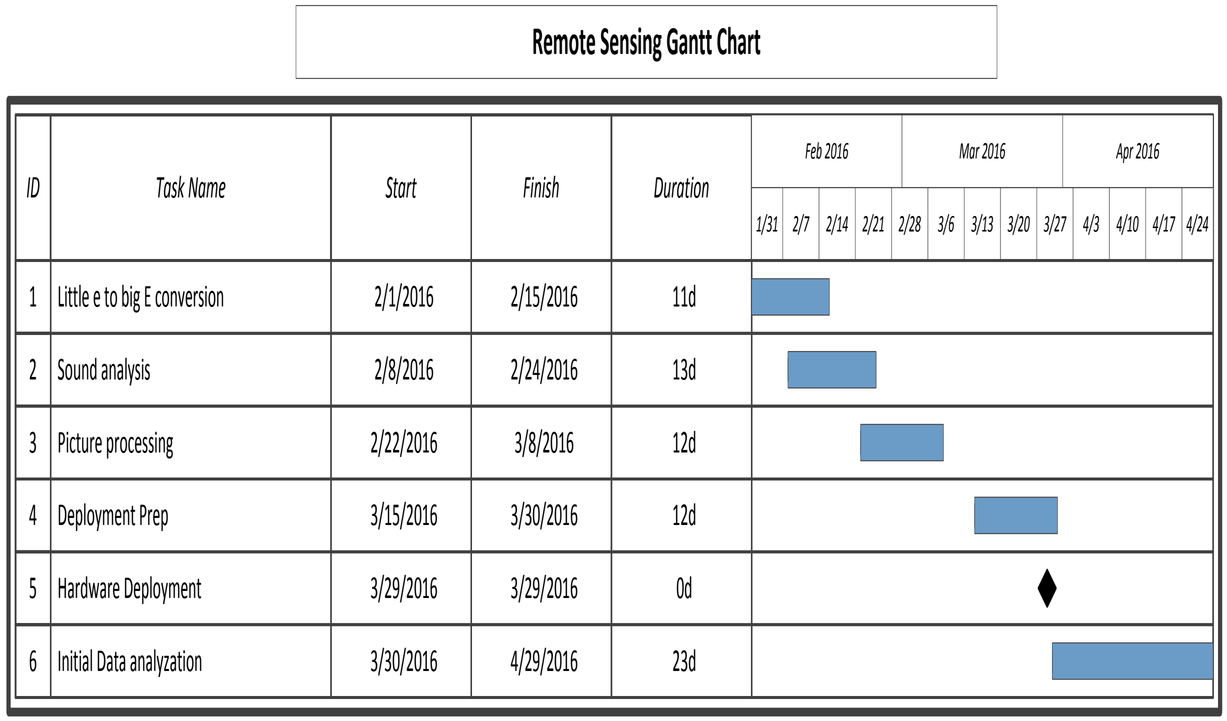


NetworkType: RegularWiggled
wiggle 0.41
ConnectivityType: TransmitRange
degree 5
GRIDSIZE 10
TRANSMISSIONRANGE 1.3
MESSAGESTARTNODE: 55
NumTriangTests 1
checkNum: 100

Figure 2.5 NetLogo Simulation Software showing Network configuration(Top). Triangulation from white node (Bottom)

2.4 Coding Methodology

Coding methodologies for this project mostly follow the agile process. All team members are assigned tasks on a weekly basis by the project manager Dr. Martin Cenek. Although each member has their own area of strengths and responsibility, all members contribute as necessary to each of the other areas. We started by implementing basic functionality in a simple but working network architecture. As the project has progressed we have added features and functionality that increase the complexity of the design.



ID	Task Name	Start	Finish	Duration
1	Little e to big E conversion	2/1/2016	2/15/2016	11d
2	Sound analysis	2/8/2016	2/24/2016	13d
3	Picture processing	2/22/2016	3/8/2016	12d
4	Deployment Prep	3/15/2016	3/30/2016	12d
5	Hardware Deployment	3/29/2016	3/29/2016	0d
6	Initial Data analyzation	3/30/2016	4/29/2016	23d

Figure 2.7 The Gantt chart for project development.

Chapter 3

Design and Testing / User Interface

3.1 Project Management

For the new developer, project management is not well understood or appreciated for its importance to the success and failure of software design, development, and implementation. Most developers just want a problem statement or requirements list and they want to start coding. After all, isn't that the whole point to software development; producing working code? For a lot of people, it is difficult to see the fruits of project management. In most cases, the contributions are abstract. While the developers produce something that can be seen and measured [1]. To appreciate project management, we must define and understand some of the roles of they play and how they fit into the larger puzzle.

Team Leadership: As talented as a group of good developers are, the nature of their jobs lend themselves to working alone. The project manager, as a team leader, is the glue that binds teams into cohesive working groups and makes sure that all members are focused on the right areas and pointed in the same direction. The ability to do this effectively keeps the project on track and ensures it does not stray from the desired end result. Additionally, a good project manager exercises situational leadership to exploit and maximize team member strengths while controlling any potential team conflicts. While team leadership is difficult to measure in terms of time and money, the dividends are unmistakable.

Project Leadership: Although a project is not a living breathing thing, if not properly managed it can take on a life of its own. Instead of the team leading the project, the project leads the team. This can result in a product that diverges from its intended path and ends up being incomplete or ineffective at its intended purpose. A good project manager keeps a project on track by continually clarifying requirements, redirecting the team as required, and ensuring the project stays within its intended scope.

Crisis Manager: If team members had to drop what they were doing each time a client or management issue arose, nothing would ever get done. Issues can range from technology, personnel, budget, timing, and client interface just to name a few. The project manager can deflect or absorb most of these issues and mitigate any potential delays they would otherwise cause [2]. This keeps projects on track and in scope.

Design Methodology: There are many different design methodologies in use today from agile and scrum to waterfall. Many developers prefer one approach over the others based on personal experience and preference. If left up to the team, the design methodology could turn into a contentious debate. There are arguments over which is the best approach but the fact of the matter is that it is entirely dependent upon the situation. While the waterfall method has fallen out of favor in recent years, there may be projects that lend themselves to this development style. Additionally, a particular client may require a certain methodology based on its own internal structure or policy. A project manager can look at the big picture and work with the customer to make an informed decision on design methodology that is best suited for the project at hand.

For our particular project, the project lead, Dr. Cenek fulfills the role of project manager. We have largely utilized the agile development methodology. During our weekly meetings, he discusses the previous weeks progress (or lack thereof), and assigns tasks for the upcoming week. He also removes any obstacles that come up such as equipment issues and working locations. As the project has moved along he has altered our focus based on new and emerging requirements.

Figure 3.1 1 Development team with that illustrates teamwork required [3]

3.2 Test Cases

Testing is perhaps one of the most overlooked aspects of software development. It is easy to fall into the trap of testing around the way an application is built instead of testing based on requirements. Since a developer knows exactly how his or her program operates, they will often test based on this knowledge. Many times scenarios are missed because if the developer didn't think of it during the design phase, why would they think of it during the testing phase? To avoid this pitfall it is a good practice to have a separate test team or have developers test each other's code instead of their own.

Because software is prone to bugs, it should be tested as thoroughly as possible. Software testing can provide objective, independent information about the quality of software and risk of its failure to users and/or sponsors [4]. Test all conceivable configurations and input scenarios to minimize the chance that a bug will go unnoticed. Regardless of how thorough we may think our testing process has been, we will inevitably miss something or a customer/user will do something that we did not foresee as a possibility.

For our project, we have the integration between hardware devices, software on the devices, the interaction between the devices when loaded with the software and the simulator software as well. Before we develop software to test on the hardware platform we needed to make sure the devices were working correctly. Below is our process for testing the hardware devices:

1. Visual inspection
2. Solder connections
3. Load Basic pre-developed software
4. Apply power
5. Run software and observe the results

Because these devices will be used as part of a large network of communication nodes, it is important to test each one out individually to save troubleshooting the entire network to isolate a faulty node once deployment and more advanced testing has begun. Once all hardware devices were tested, we could move on to software development. Our network structure and communication protocol goals are relatively complex. Because of this, we have taken the approach of developing features and behaviors in stages and testing them as we go along (this follows our agile methodology). By taking this approach we simplify the problem and are able to build to a more complex solution. Below are the basic steps we have taking in testing our software:

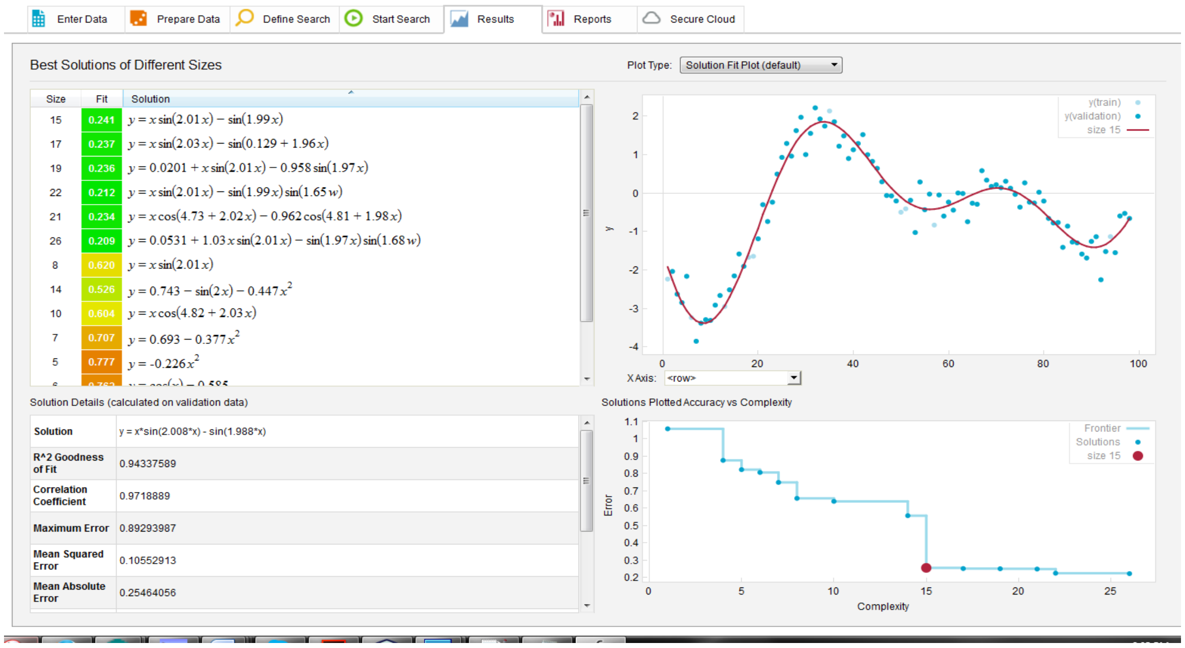
1. Produce minimal code for node to node communication
2. Test basic communication between 2 basic nodes
3. Expand communications test to 3x3 grid of nodes
4. Increase network complexity based on design requirements
5. Test new software on small grid
6. Increase grid size to 7x7 and test
7. Add more complex functionality based on requirements
8. Test on large grid
9. Repeat 7-9 as required

By taking this approach we were able to isolate problems quickly and focus on solutions on more solvable problems. This enabled us to be able to demonstrate different working phase to customers earlier than otherwise possible.

Given the nature of our project, not all desired behavior could be tested in the hardware or software that was deployed on the hardware. Since there is not a conventional user interface, and communication nodes exchange information through radio waves, it is a challenge to observe behavior under certain conditions. Part of this project utilizes agent based modeling software, NetLogo, to help test features that would be otherwise impossible or impractical to test in a laboratory environment. As it turns out, we actually used additional software called Eureqa to help formulize the results of the simulation software. Below are the basic steps we took in using the simulator to test network behavior:

1. Define problem and parameters to test on simulator
2. Configure software to meet conditions
3. Run simulation software
4. Collect results/analyze data
5. Implement software in hardware based on observed sim results

One particular problem that could not be tested in hardware was the ability to isolate event location based on triangulation to the node based on number of hops. Since our board was set up on a 4 foot by 4 foot piece of wood, it was not feasible to test the triangulation feature. Instead we implemented this feature in the simulation software and to fine tune it we used Eureqa software to help develop a fitness function to minimize triangulation error.



Best Solutions of Different Sizes

Size	Fit	Solution
15	0.241	$y = x\sin(2.01x) - \sin(1.99x)$
17	0.237	$y = x\sin(2.03x) - \sin(0.129 + 1.96x)$
19	0.236	$y = 0.0201 + x\sin(2.01x) - 0.958\sin(1.97x)$
22	0.212	$y = x\sin(2.01x) - \sin(1.99x)\sin(1.65w)$
21	0.234	$y = x\cos(4.73 + 2.02x) - 0.962\cos(4.81 + 1.98x)$
26	0.209	$y = 0.0531 + 1.03x\sin(2.01x) - \sin(1.97x)\sin(1.68w)$
8	0.620	$y = x\sin(2.01x)$
14	0.526	$y = 0.743 - \sin(2x) - 0.447x^2$
10	0.604	$y = x\cos(4.82 + 2.03x)$
7	0.707	$y = 0.693 - 0.377x^2$
5	0.777	$y = -0.226x^2$

Solution Details (calculated on validation data)

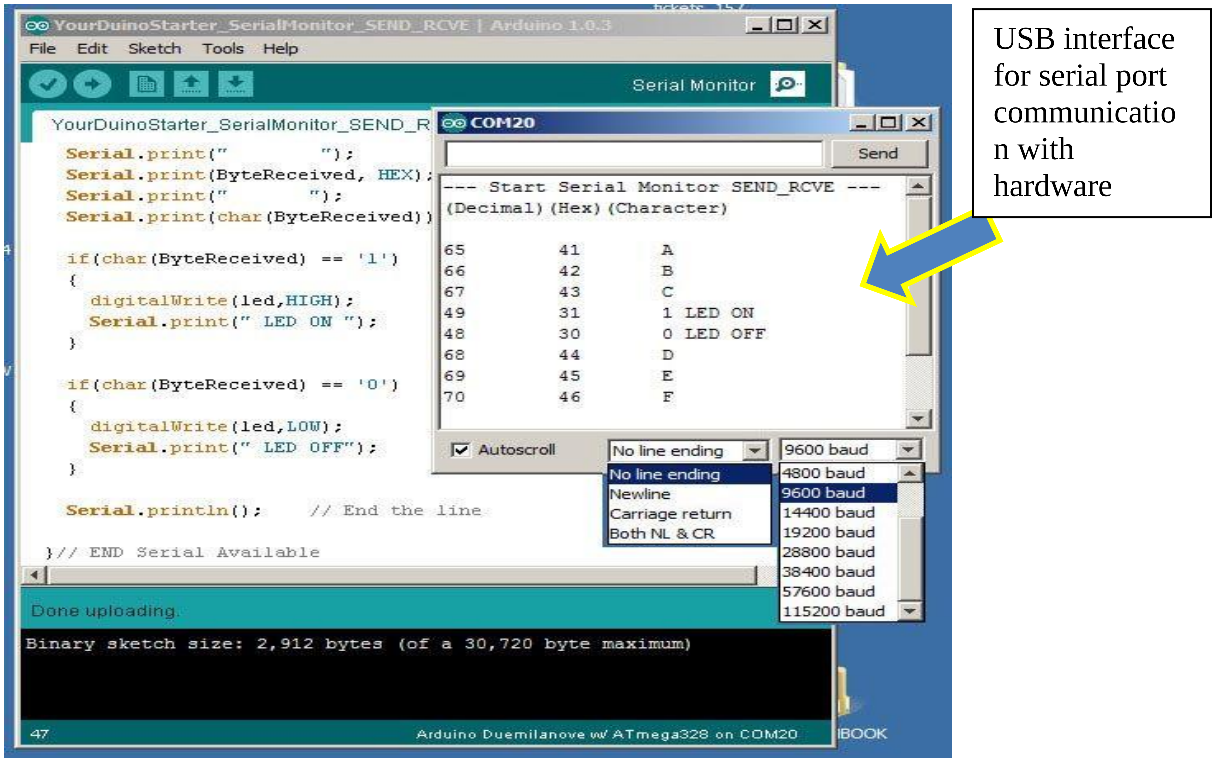
Solution	y = x*sin(2.008*x) - sin(1.988*x)
R^2 Goodness of Fit	0.94337589
Correlation Coefficient	0.9718889
Maximum Error	0.89293987
Mean Squared Error	0.10552913
Mean Absolute Error	0.25464056

Figure 3.2 Eurqa Software formulizing function to help reduce triangulation error

3.3 User Interface

A quality user interface can make average or even poorly written software desirable for people to use. People are visual creatures, and although all of the logic and real "work" of a software program is not seen by the user, it is the interface that makes the impression and in today's world that goes a long way. Conversely, an excellent piece of software can sometimes fail to be utilized to its potential if the user interface is not well designed. It is important to consider layout, ease of use (although this is a matter of opinion), complexity, and what the trends are (what has the user come to expect) such as button placement and design and expected behavior with GUI widgets.

Because this project includes a hardware based communications network, there is currently no user interface associated with it. As the project nears completion a web based interface will be developed so access to data collected by the network can be achieved. This interface will likely be in java or PHP and provide access to a MySql database. Although there is a lack of user interface for the network, we do use a provided USB interface that is used to observe communication to and from nodes connected to the computer via the USB. Additionally, the simulator software has a developer configurable interface that allows a researcher to set up network parameters for testing. This interface includes event based buttons, sliders, graphs, text boxes, labels and an output window.


YourDuinoStarter_SerialMonitor_SEND_RCVE | Arduino 1.0.3
File
Edit
Sketch
Tools
Help
Serial Monitor
YourDuinoStarter_SerialMonitor_SEND_R
COM20
Send
Autoscroll
USB interface for serial port communicatio n with hardware

```
  Serial.print("         ");
  Serial.print(ByteReceived, HEX);
  Serial.print("        ");
  Serial.print(char(ByteReceived))

  if(char(ByteReceived) == '1')
  {
    digitalWrite(led,HIGH);
    Serial.print(" LED ON ");
  }

  if(char(ByteReceived) == '0')
  {
    digitalWrite(led,LOW);
    Serial.print(" LED OFF");
  }

  Serial.println();    // End the line

}// END Serial Available
```

```
--- Start Serial Monitor SEND_RCVE ---
(Decimal)(Hex)(Character)

65        41        A
66        42        B
67        43        C
49        31        1 LED ON
48        30        0 LED OFF
68        44        D
69        45        E
70        46        F
```

No line ending
[✓ No line ending] Newline Carriage return Both NL & CR

9600 baud
4800 baud [✓ 9600 baud] 14400 baud 19200 baud 28800 baud 38400 baud 57600 baud 115200 baud

```
Done uploading.
Binary sketch size: 2,912 bytes (of a 30,720 byte maximum)
```

47   Arduino Duemilanove w/ ATmega328 on COM20

Figure 3.3 Arduino serial monitor interface [4]

Steup: The setup button clears the previous world if one exists, populates the board with the number of nodes specified in the grid size window, places the nodes in the world with a level of random irregularity.

clear: Clears all nodes and links from the world.

NetworkType: Establishes if the setup will use wiggle, no wiggle or the initial setup pattern.

Wiggle slider: used to variably select the wiggle offset to be used during world setup

Connectivity type: Allows the user to determine if radio range or max node degree will be used to determine connectivity between nodes for graph layout.
degree slider: Used in conjunction with connectivity type to set max node degree for graph setup.

GridSize: Variable used to determine the two dimensional size of the world. Number of nodes = gridSize ^2

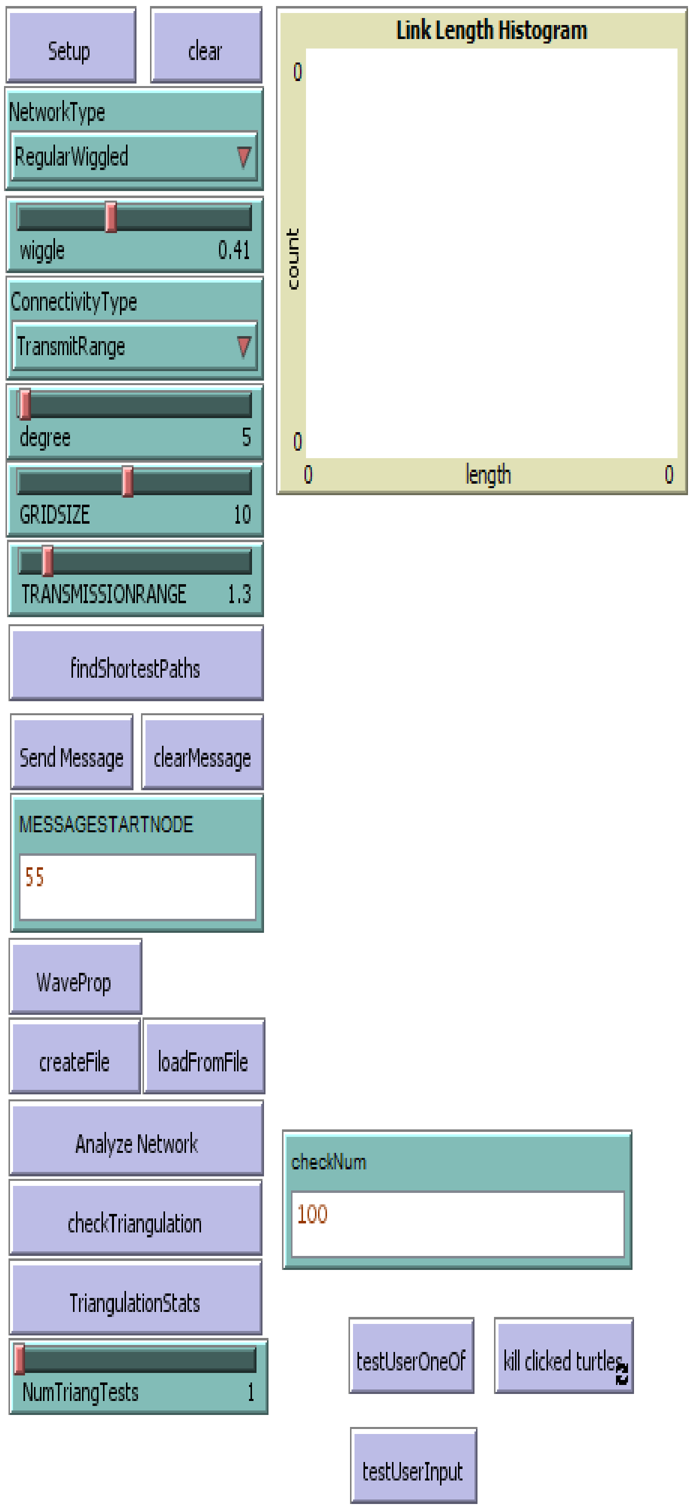
TRANSMISSIONRANGE: Establishes radius for node connectivity.

findShortestPath: This calls a procedure for each node to establish the shortest path based on number of hops to each of the four readout nodes.

sendMessage: Triggers an event from the node specified in the MESSAGESTARTNODE window.
create file: generates a csv file that can be used to transfer connectivity and world parameters to hardware devices

Analyze Network: Triggers an event from each node in a random manner to test connectivity and world behavior. Prints some information about the results to the screen for user interpretation.

Check Triangulation: Triggers an event from each node and records/displays results relevant to the triangulation method



NetworkType: RegularWiggled

wiggle 0.41

ConnectivityType: TransmitRange

degree 5

GRIDSIZE 10

TRANSMISSIONRANGE 1.3

MESSAGESTARTNODE: 55

checkNum: 100

NumTriangTests 1

Figure 3.4 Simulator software user interface and basic descriptions

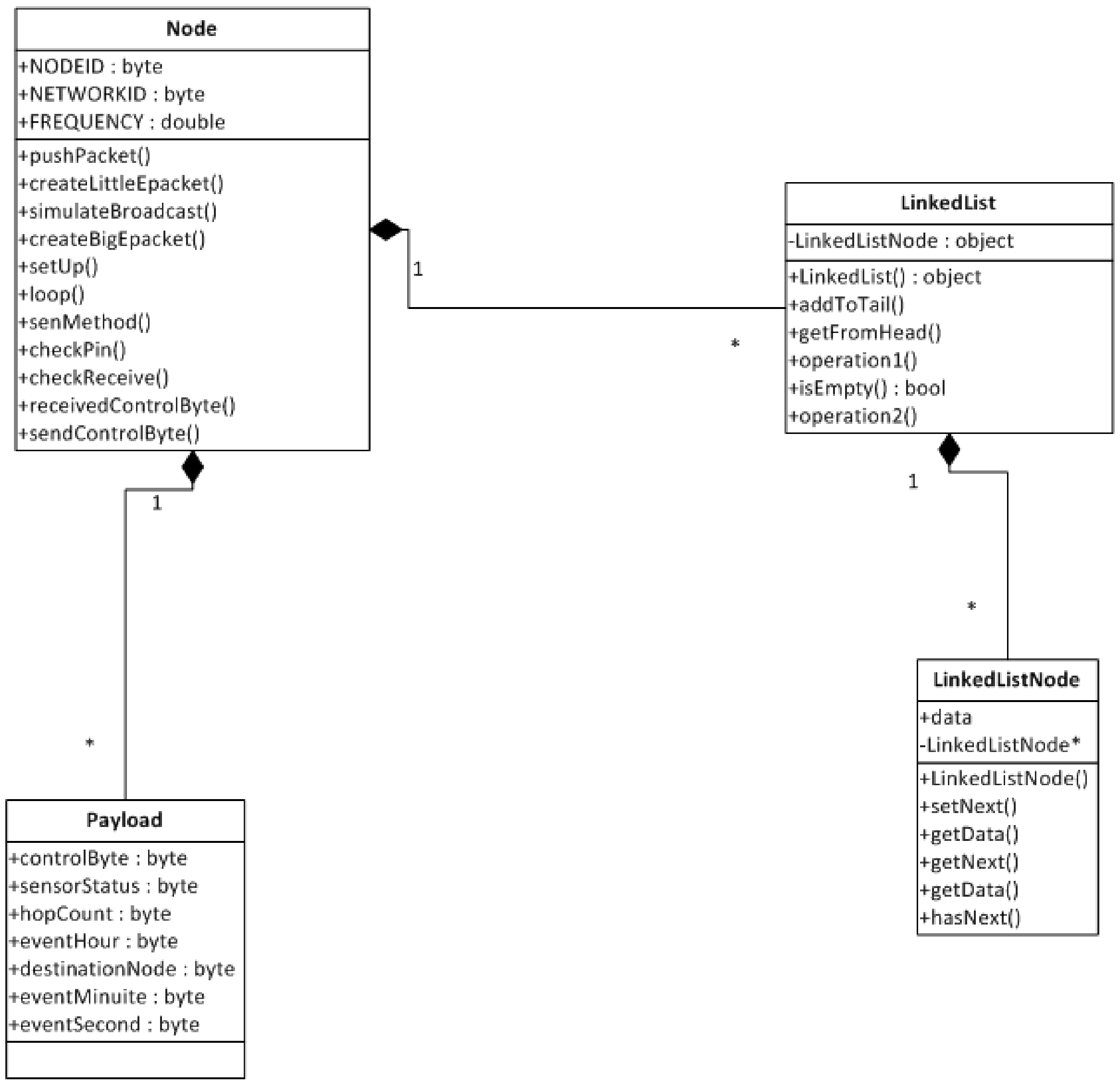


Figure 3.5 Sensor UML

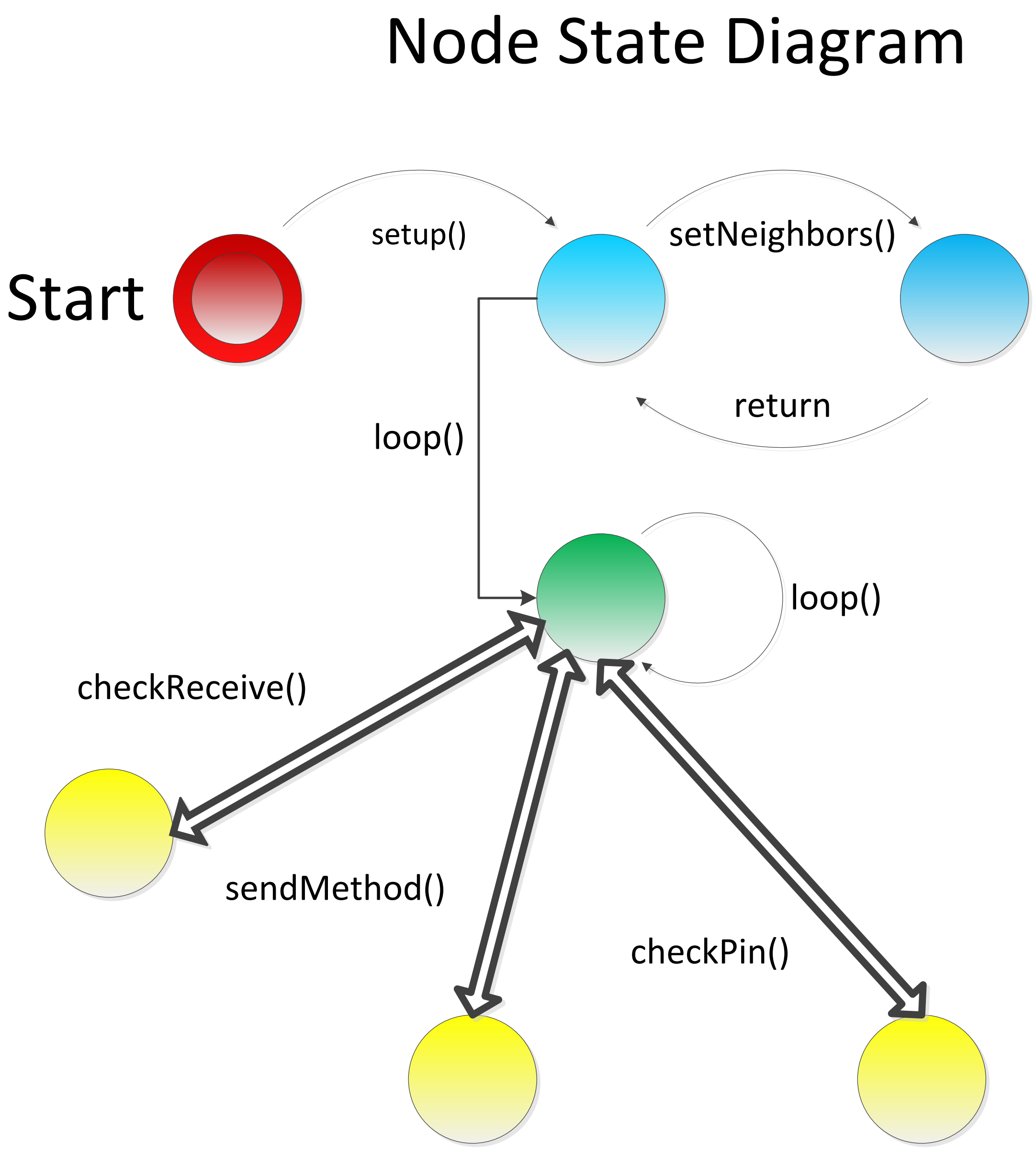


Figure 3.6 Node State Diagram

Chapter 4

Results and Discussion

4.1 Introduction

Our research team led by Dr. Martin Cenek began this project with a concept of creating a sensor platform that will be cost effective, robust, reliable and self-healing. From selecting a hardware platform, learning its capabilities, and initial design, we have taken this project from concept to tangible. While field testing has yet to begin, we have done extensive lab testing and successful demonstrations that show the concept is plausible and well on its way to implementation.

4.2 Results

4.2.1 Hardware

Nodes:

The hardware selected for this project, the Moteino with embedded RFM69 UHF transceiver, has proven quite capable. As mentioned before, we have not subjected these units to field conditions; however, they have proven to be highly reliable. Of the 60+ units we have been using only one has failed to my knowledge. That is a reliability rate of 98.3% in the lab environment. Furthermore, while somewhat limited computing capacity with 2 KB memory and a 16 MHz processor; they have handled everything we have thrown at them up to this point with no signs of sluggish performance. The onboard radio does have some range limitations given the low output power coupled with RF wavelength behavior but this is overcome by increasing the deployment density.

Microphones:
We decided to test our network using microphones as the sensor output into the nodes for audio detection. Research and configuration of the microphones are ongoing but initial indications show good selectivity and sensitivity in the 300 – 3 KHz ranges. The goal is to be able to create audio profiles that are characteristically similar to the sounds we are trying to detect. When the microphone picks up the sound we would then compare that sound to the profile and determine if it fits the profile we are looking for.

Figure 4.1 Moteino capable microphone

4.2.2 Software

Nodes:
The software development has been the most robust and challenging part of this project. After several iterations, we have developed a software package that operates in a high traffic RF hostile environment while maintaining reliable payload delivery to the readout destination. The most effective way for us to do this was to implement a payload structure that could be stored in a circularly linked list so that we can perform several checks to ensure data integrity. Of those checks we look for duplicate event data to avoid loops, stale data to prevent sending old information, and event reliability to minimize triggering false positive events. For both efficiency and triangulation purposes we keep track of shortest hop data in a 2D array within each node. Most times only the first or second column of data is ever retrieved in this array so data access is very fast. This software package has proven reliable and effective in the lab environment. Click here to view source code→ ***Appendix A***

Code repository shifted from GitHub to Moodle for legacy reasons:
http://137.229.141.184/moodle/course/view.php?id=14

Node Source Code:

Simulator:

The decision to utilize agent based simulation software to model network behavior and implement features for testing has proven to be a valuable evaluation tool. By providing a visual context for both network connectivity and layout, we were able to monitor behavior for node failures, hop patterns, and develop a potential triangulation technique without the aid of coordinates or synchronization between nodes. Additionally, the simulator is able to take an input file and configure nodes based on the data provided in the file. This allows dynamic testing based on a real world deployment scenario.

4.2.3 Event Detection

Event detection began with simple triggering of an event by applying a signal to an input pin. When this occurred, the node that detected the event would access its hop matrix and determine the best neighbor to transmit the event to. We used a handshake protocol to ensure the message was successfully received. In the event an acknowledgement could not be obtained, the sensing node would continue through its hop matrix until it got message delivery confirmation. The event node would continue this until the data was transmitted to each of the four readout nodes. After this detection technique was perfected we transitioned to an advanced detection algorithm to help reduce false positives and provide potential event directionality. This new technique uses a reinforcement of signals from adjacent nodes within a time domain window to either ignore the event or to verify and propagate the event through the network. While this method is still being perfected it has shown some early promise.

4.2.4 Triangulation

The ability to isolate event detection to a geographic area without coordinates, synchronized clocks or even knowledge of node id relative position is a challenge. With the help of the simulation software and some mathematical experimentation, we were able to come up with triangulation concept that was over 97% effective at event location using the simulator. This technique does require some network statistical information to be effective and the more accurate the data the better the triangulation results would be. This technique uses the number of hops to each readout and an average hop distance to get an estimate radius from each readout node positioned 90 degrees apart. With the concentric circles overlapping we find the intersections mathematically and then calculate the midpoint between the intersections to further isolate the potential location. Because we have not deployed these sensors yet, we have not been able to evaluate the results using the actual sensors and real events.

4.2.5 Database

We are in the early stages of database and front end development so that event data can be retrieved and analyzed. We chose PHP and MySql for its cross platform functionality. The database will keep track of nodes, node types, big E and little E events.

Chapter 5

Conclusion

5.1 Introduction

Designing and implementing a sensor network that can operate reliably in the arctic region is a significant challenge. As the arctic becomes more and more accessible to people, the need for a system or method of collecting and analyzing data becomes more profound. The challenges associated with such a project include the environment, lack of infrastructure, cost, versatility, and reliability just to name a few. Our team set out to find an inexpensive flexible platform that we could use to design a network that meets these demands. While this research will continue for some time, we have made significant strides in the right direction.

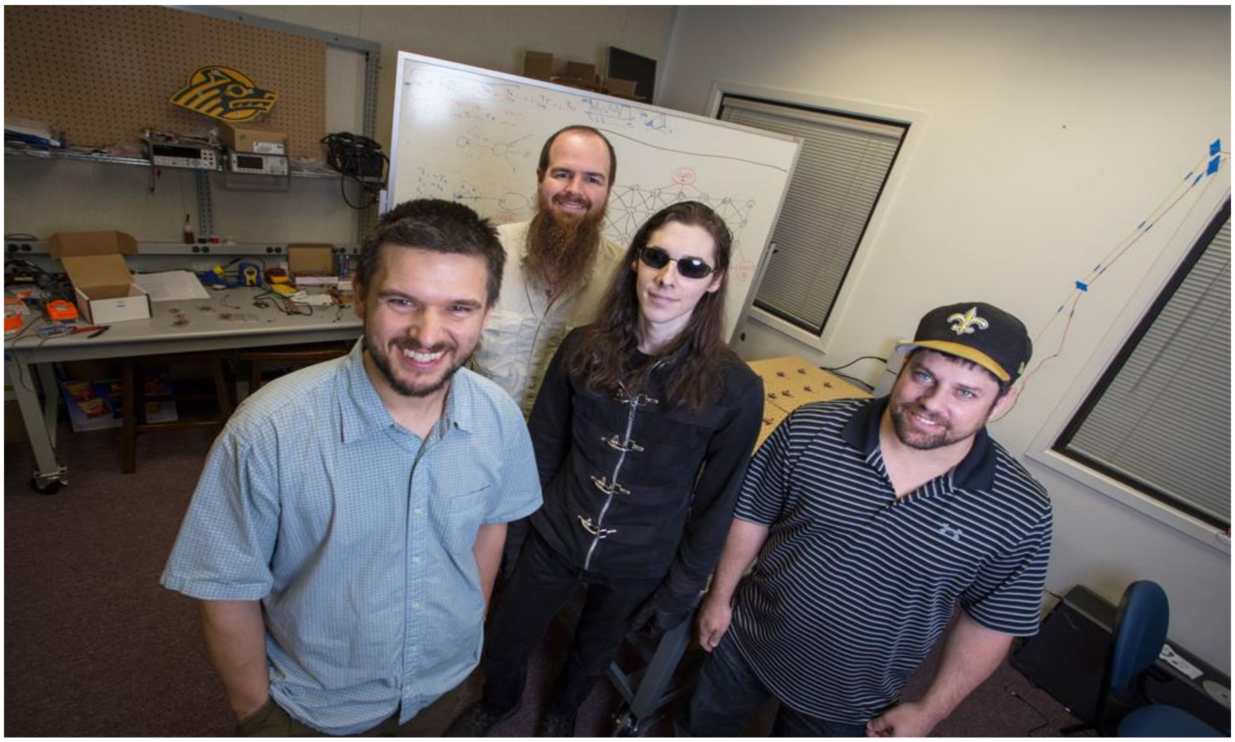


Figure 5.1 Remote Sensing research team in the ICAN lab. Left to right: Dr. Martin Cenek, Lance Leber, Matthew Devins, Mike Mobley

5.2 Conclusion

As I wrap up my official involvement with this research project pending my forthcoming graduation, I reflect on what has transpired from the beginning until now. I was fortunate to be involved in this project from the ground floor and have seen it grow through trial, error and unforeseen challenges. My goal was to have begun field testing by the end of the semester but that did not happen due to a variety of issues such as equipment ordering and scheduling issues that come along with a busy semester. Being a part of a team, where each member brings their own set of skills and personalities has been quite rewarding. I have learned a great deal about problem solving and how to approach problem solving from perspectives other than my traditional approach.

I have learned several valuable lessons from this project. 1) Project Management: Being able to observe Dr. Cenek manage his team and their expectations was educational. When I found myself questioning a direction he was taking the project, I only had to wait and see that his methods lead us to more advanced and challenging designs. By giving us smaller pieces to digest and work on at a time, we were able to march towards more advanced design without biting off more than we could chew at a time. His comprehensive view of the end goal kept us on track and avoided inevitable tangents that would have stifled or even destroyed progress. 2) Teamwork: While I have many years of experience in working in teams, this was my first opportunity to work as a member of a research team. It was interesting to see the different strengths each team member brought with them as well as the different perspectives around software development. 3) Testing and Simulation: I already had some appreciation for testing that was strengthened by this project. Thorough testing enabled us to find limitations with our design. By testing each new feature before moving on, we were able to prevent large scale refactors and avoid substantial problems during the more advanced design phases. This was my first introduction to simulation of this kind. I was quite skeptical in the beginning and felt we were spending a lot of time and effort on a part of the project that will not produce any results. However, as the design became more complex and the testing limitations were realized, I became quite convinced that the simulation aspect was one of our most important design pieces. Without the simulation aspect, our triangulation technique could not have been developed.

5.2 Future Work

Because this research project is part of a large Department of Homeland Security funded project, research will continue for some time. For the students that carry on this research, further advancements still need to be made in both audio and video processing, enhanced event detection algorithm design, network connectivity to base/home computing center through either Wi-Fi or node USB interface. Additional advancements need to be made with the database and front end application to work with event archives. Finally, field testing needs to be accomplished. This is perhaps the most important milestone facing this project in the near future. The nodes need to be configured in their weather tight containers, deployed and monitored. Because this network is intended to be flexible, experimentation needs to be accomplished with a variety of sensors such as motion, vibration, atmospheric, etc.

Appendix A

Code repository shifted from GitHub to Moodle for legacy reasons:

http://137.229.141.184/moodle/course/view.php?id=14

Node Source Code:

```
//5.10

//setNeighbors(): CREATED, SETS UP SIMULATED NEIGHBOR LIST BASED ON NODEID
//checkPin(): FIXED, NOW LISTENS FOR A PIN TRIGGER + TRANSMITS. ALSO MADE TO NOT TR
ANSMIT TWICE IF THE SAME SENSOR READING IS PRESENT BECAUSE THE HUMAN TRIGGERING IT
PUT THE JUMPER BACK ON TOO SLOWLY.
//----
sendData(): FIXED, NOW LOOPS THROUGH EACH NEIGHBOR AS BEFORE BUT TRANSMITS UNTIL AL
L NEIGHBORS ARE REACHED OR THE NODES THAT HAVE NOT ACKED HAVE BEEN DOWN LONGER THAN
 THE MAX SLEEP PERIOD LENGTH. ALSO DOES NOT RETRANSMIT TO THE SAME NEIGHBOR THAT HA
S ACKED TWICE LIKE BEFORE, IF THIS PACKET WAS SENT BY THIS NODE TO THAT NODE IN THI
S LOOP ALREADY IT ALREADY HAS UP TO DATE INFO.
//checkReceive(): COPIED TO CheckReceiveM() FOR MODIFICATION
//----relay(): FIXED, MODIFIED TO WORK LIKE sendData()
//checkReceiveSave(): CREATED, IF ANOTHER NODE IS BROADCASTING IT WILL NOT SEND AN
ACK. THIS METHOD SAVES INBOUND PACKETS IN THE MIDDLE OF BROADCASTING FOR POSSIBLE R
ELAYING LATER.

//#include <Time.h>
#include <RFM69.h>
#include <SPI.h>
#include <SPIFlash.h>
#include <stdlib.h>

#define NODEID      99
#define NETWORKID   100
#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
#define ACK_TIME    30  // # of ms to wait for an ack
#define GATEWAYID   255

RFM69 radio;
```

```
SPIFlash flash(8, 0xEF30); //EF40 for 16mbit windbond chip
//May not need this if TX based on time
int TRANSMITPERIOD = 500; //transmit a packet to gateway so often (in ms)
byte sendSize=0;
boolean requestACK = false;
boolean promiscuousMode = false; //set to 'true' to sniff all packets on the same
network
boolean clockSet = false;

byte * neighbors;
byte * readouts;

byte arraySize = 10;
byte sensePin = 7;
byte ackCount=0;
bool * hasACKed;
unsigned long transStart;
bool timeoutReset = true;

byte numNeighbors = 0;//TOTAL NUMBER OF NEIGHBORS OF THIS NODE
byte currentNode = 0;
byte currentTime = 0;

int MAXSLEEPWAIT = 1000; //TEMPORARILY SET TO 1 SECOND, NEEDS TO BE SYNCED TO THE
MAX SLEEP TIME
int SLEEPTIME;

byte MAXWAVEHOPS = 2;

byte NUMREADOUTS = 4;
byte** hopArray;
boolean sensorTriggered = false;

int lightTimeout = 3000;
unsigned long lightOff = 0;

const byte DATAARRAYSIZE = 30;

struct Payload
{
  int           nodeId; //store this nodeId
  //may be able to eliminate this
  unsigned long uptime; //uptime in ms
  //
  byte          controlByte;
  byte          sensorStatus;
  byte          hopCount;
  byte          destinationNode;
  byte          eventHour;
  byte          eventMinute;
  byte          eventSecond;
  byte          dataArray[DATAARRAYSIZE];
};
```

```
class LinkedList
{

  private:

  class LinkedListNode
  {
    private:
      Payload data;
      LinkedListNode *next;

    public:

      LinkedListNode(Payload dataIn)
      {
        data = dataIn;
        next = 0;
      }

      LinkedListNode(Payload dataIn, LinkedListNode* nextIn)
      {
        data = dataIn;
        next = nextIn;
      }

      LinkedListNode* getNext()
      {
        return next;
      }

      void setNext(LinkedListNode* linkIn)
      {
        next = linkIn;
      }

      Payload getData()
      {
        return data;
      }

      boolean hasNext()
      {
        if(next == 0)
        {
          return false;
        }
        else
        {
          return true;
        }
      }

  };
```

```
LinkedListNode *head;
LinkedListNode *tail;

public:

LinkedList()
{
  head = 0;
  tail = head;
}

void addToTail(Payload dataIn)
{

  //LinkedListNode *temp;// = new LinkedListNode();

  if(head == 0)
  {
    head = new LinkedListNode(dataIn);
    tail = head;
  }
  else
  {
    /*LinkedListNode *temp = head;
    while(temp->hasNext())
    {
      temp = temp->getNext();
    }
    temp->setNext(new LinkedListNode(dataIn));*/

    tail->setNext(new LinkedListNode(dataIn));
    tail = tail->getNext();
  }
}

Payload getFromHead()
{

  Payload temp = head->getData();
  LinkedListNode* LLTemp = head;
  head = head->getNext();

  delete LLTemp;
  return temp;

}

boolean isEmpty()
{
  if(head == 0)
  {
    return true;
  }
```

```
    else
    {
      return false;
    }
  }

  /*void dumpLL()
  {
    int counter = 0;
    LinkedListNode *temp = head;
    while(temp != 0)
    {
      Serial.println((String)counter + ":" + temp->getData());
      temp = temp->getNext();
    }
  }*/

  ~LinkedList()
  {
    LinkedListNode *temp;
    tail = 0;
    while(head != 0)
    {
      temp = head->getNext();
      delete head;
      head = temp;
    }
  }

};

Payload tempPacket;

Payload theData;
Payload rxData;

const byte QUEUESIZE = 10;
Payload packetQueue[QUEUESIZE];//ARRAY TO BE USED AS A CIRCULAR LIST IMPLEMENTING A
QUEUE OF PACKETS TO ALLOW INTERLEAVED RECEIVE/SEND/SENSING AND TO PREVENT
RETRANSMISSION BY KEEPING A LIST OF THE LAST QUEUESIZE PACKETS SEEN.
byte currentPacket = 0;
byte numPackets = 0;

LinkedList PacketLL;
byte TRANSMISSIONCOUNT = 10;
byte remainingTransmissions = 0;
unsigned int MAXDELAY = 4000000000;

unsigned int randomDelay;

void pushPacket(struct Payload& packetIn)
{

  packetQueue[(currentPacket + numPackets) % QUEUESIZE].nodeId = packetIn.nodeId;
  packetQueue[(currentPacket + numPackets) % QUEUESIZE].uptime = packetIn.uptime;
  packetQueue[(currentPacket + numPackets) % QUEUESIZE].controlByte = packetIn.cont
rolByte;
  packetQueue[(currentPacket + numPackets) % QUEUESIZE].destinationNode = packetIn.
destinationNode;
  packetQueue[(currentPacket + numPackets) % QUEUESIZE].sensorStatus = packetIn.sen
sorStatus;
  packetQueue[(currentPacket + numPackets) % QUEUESIZE].eventHour = packetIn.eventH
our;
  packetQueue[(currentPacket + numPackets) % QUEUESIZE].eventMinute = packetIn.even
tMinute;
  packetQueue[(currentPacket + numPackets) % QUEUESIZE].eventSecond = packetIn.even
tSecond;

  numPackets++;
}

void createLittleEPacket()
{
  tempPacket.nodeId = NODEID;

  tempPacket.uptime = millis();
  //tempPacket.eventHour = hour();
  //tempPacket.eventMinute = minute();
  //tempPacket.eventSecond = second();
}

void createBigEPacket(byte destinationIn)
{

  tempPacket.nodeId = NODEID;
  tempPacket.controlByte = 3;
  tempPacket.destinationNode = destinationIn;
  tempPacket.uptime = millis();
  tempPacket.hopCount = 0;

}


void setup()
{
  Serial.begin(SERIAL_BAUD);
  delay(10);
  radio.initialize(FREQUENCY,NODEID,NETWORKID);
  radio.setPowerLevel(31);
  radio.encrypt(KEY);
  radio.promiscuous(promiscuousMode);
  pinMode(sensePin, INPUT_PULLUP);
  pinMode(LED, OUTPUT);
  //pinMode(LED, OUTPUT);
  Serial.println((String)"\nListening at %d Mhz..." + (FREQUENCY == RF69_433MHZ ?
433 : FREQUENCY == RF69_868MHZ ? 868 : 915));
  //sets time to 00:00:00 Jan 1, 2015 as an arbitrary starting point
  //setTime(0, 0, 0, 1, 1, 2015);
  setNeighbors();
  hasACKed = new bool[numNeighbors];

  //test neigbor array
  Serial.print((String)"Neighbors of node " + NODEID + ": ");
  for(int i = 0; i < numNeighbors; i++)
  {
    Serial.print(neighbors[i]);
    Serial.print(" ");
  }

  srand(NODEID);
  //PacketLL.addToTail(rxData);
  Serial.println();

  //for(int i = 0; i < 50; i++)
  //{
  //  Serial.println((String)"RANDOM IS " + random());
  //}
  //end test
}

void loop()
{

  //listen
  randomDelay = random();// % MAXDELAY;

  //Serial.println((String)"Listening " + randomDelay + " times");

  while(randomDelay > 0)
  {
    //Serial.println((String)randomDelay);
    checkReceive();
    checkReceive();
    checkReceive();
    randomDelay--;
  }

  //send
  sendMethod();

  //sense
  checkPin();

  if(millis() > lightOff)
  {
    digitalWrite(LED, LOW);
  }

}

void lightOn()
```

```
{
  digitalWrite(LED, HIGH);
  lightOff = millis() + lightTimeout;
}

void checkReceive()
{
  //IF THERE IS A PACKET IN THE BUFFER
  if (radio.receiveDone())
  {
    //AND THE PACKET IS VALID
    if (radio.DATALEN == sizeof(Payload))
    {
      //SEND AN ACK IF ONE WAS REQUESTED
      if (radio.ACKRequested())
      {
        radio.sendACK();
        Serial.println((String)"ACK requested, ACK sent.");
      }

      //PUT THE PACKET INTO RXDATA
      rxData = *(Payload*)radio.DATA; //assume radio.DATA actually contains our
struct and not something else

      Serial.print("Received Packet from ["); Serial.print(radio.SENDERID, DEC);
Serial.print("] ");
      Serial.println((String)" [RX_RSSI:" + (radio.readRSSI()) + "]");

      displayPacket(rxData);

      //HANDLE THE PACKET APPROPRIATELY, AND PUSH IT TO THE QUEUE
      switch(rxData.controlByte)
      {
        //CONTROL BYTE 0: SET NEIGHBOR ARRAY FROM SIMULATION DUMP FILE (DEMO
PURPOSES)
        case 0:
          //receivedControlByte0();
          break;
        //CONTROL BYTE 1: SET UP NEIGHBOR HOP ARRAY
        case 1:
          receivedControlByte1();
          break;
        //CONTROL BYTE 2: LITTLE E WAVE PACKET
        case 2:
          //receivedControlByte2();
          break;
        //CONTROL BYTE 3: BIG E TRANSMISSION PACKET
        case 3:
          receivedControlByte3();
          break;
        //CONTROL BYTE 4: NEIGHBOR UPDATE PACKET
        case 4:
          receivedControlByte4();
          break;
      }

    }
    else
    {
      Serial.print("BAD PACKET FROM ["); Serial.print(radio.SENDERID, DEC);
Serial.print("] ");
    }
  }
}

void receivedControlByte1()
{

  //find row in hop array to compare against this packet's hop count
  byte readoutIndex;
  for(byte i = 0; i < NUMREADOUTS; i++)
  {
    if(readouts[i] == rxData.nodeId)
    {
      readoutIndex = i;
      break;
    }
    //if the nodeId is not found in the list, this must be a faulty packet
    if(i == NUMREADOUTS - 1)
    {
      return;
    }
  }

  //search for the column in the hop array for this neighbor
  for(byte i = 0; i < numNeighbors; i++)
  {
    //if this packet's last sender was at index i in the neighbor list, then its
place in the hop array is also i
    if(neighbors[i] == (byte)radio.SENDERID)
    {

      Serial.println((String)(hopArray[readoutIndex][i]) + " --- " +
rxData.hopCount);

      //if this new packet's hop count is lower (better) than the previous best for
this neighbor, save it as the neighbor's new best
      if(hopArray[readoutIndex][i] > rxData.hopCount)
      {
        hopArray[readoutIndex][i] = rxData.hopCount;

        //rebroadcast or push
        rxData.hopCount++;

        Serial.println((String)"Pushing packet to LL:");
        displayPacket(rxData);
        PacketLL.addToTail(rxData);
        //Serial.println("Sending:");
        //displayPacket(rxData);
        //radio.send(255, (const void*)(&rxData), sizeof(rxData));


      }

      displayHopArray();

      break;
    }
  }

}

//THIS METHOD IS A SIMPLE RELAY, IT DOES NOT NEED TO DO ANY THINKING UNTIL IT TRIES
 TO SEND. THEN IT NEEDS TO DECIDE WHICH NEIGHBOR SHOULD RECEIVE THE PACKET.
void receivedControlByte3()
{

  //if (radio.ACKRequested())
  //{
  //  radio.sendACK();
  //  Serial.println((String)"Big E Relay packet received - ACK sent.");
  //}

  lightOn();

  rxData.hopCount++;

  Serial.println((String)"Pushing packet to LL:");
  displayPacket(rxData);
  PacketLL.addToTail(rxData);

}

//Control byte 4: Neighbor update packet
//packet contains:
//NodeId: the node that receives it will have this ID
//dataArray[0]: The number of neighbors in the array
//dataArray[]: The rest of dataArray holds the neighbor list
void receivedControlByte4()
{
  //delete the old neighbor list
  delete[] neighbors;
  //reinitialize the list with size [dataArray[0]]
  numNeighbors = rxData.dataArray[0];
  neighbors = (byte*)malloc(numNeighbors);

  //fill the list with the rest of dataArray[]
  for(int i = 0; i < numNeighbors; i++)
  {
    neighbors[i] = rxData.dataArray[i+1];
  }
  showNeighborList();
}

void sendMethod()
{
  //Serial.println((String)"SM: remaining " + remainingTransmissions + " isEmpty? "
+ PacketLL.isEmpty());
  //byte TRANSMISSIONCOUNT = 6;
  //byte remainingTransmissions = 0;
  if(remainingTransmissions > 0)
  {

    Serial.println((String)"Sending packet:");
    displayPacket(theData);

    //radio.send(255, (const void*)(&theData), sizeof(theData));
    //remainingTransmissions--;
    //Serial.println((String)(remainingTransmissions + " transmissions left"));

    switch(theData.controlByte)
    {
      case 0:
        break;
      case 1:
        sendControlByte1();
        break;
      case 2:
        break;
      case 3:
        sendControlByte3();
        break;
    }
  }
  //Serial.println((String)(!(PacketLL.isEmpty())) + " " + remainingTransmissions);
  if(!PacketLL.isEmpty() && remainingTransmissions == 0)
  {
    Serial.println((String)("Moving new packet into temp"));
    theData = PacketLL.getFromHead();
    remainingTransmissions = TRANSMISSIONCOUNT;//THIS IS THE NUMBER OF BROADCAST
ATTEMPTS, IN THE CASE OF NON-BROADCAST THE NUMBER DOES NOT MATTER AS LONG AS IT IS
CHANGED TO ZERO ONLY WHEN AN ACK IS RECEIVED
  }
}

void sendControlByte1()
{

  //Serial.println((String)"Sending packet:");
  //displayPacket(theData);

  radio.send(255, (const void*)(&theData), sizeof(theData));
  remainingTransmissions--;
  Serial.println((String)(remainingTransmissions + " transmissions left"));
```

```
}

void sendControlByte3()
{
  //Serial.println((String)"Sending packet:");
  //displayPacket(theData);

  bool noACK[numNeighbors];
  byte currNeighbor;
  byte readoutIndex;

  for(byte i = 0; i < NUMREADOUTS; i++)
  {
    //Serial.println((String)readouts[i] + " == " + theData.destinationNode + "?");
    if(readouts[i] == theData.destinationNode)
    {
      readoutIndex = i;
      //Serial.println((String)readouts[readoutIndex] + " @ " + readoutIndex + " ==
" + theData.destinationNode);
      break;
    }
  }

  Serial.println((String)"Sending to " + readouts[readoutIndex] + " at index " +
readoutIndex);

  for(int i = 0; i < numNeighbors; i++)
  {
    noACK[i] = false;
  }

  //resetHasACKed();

  //TRY TO SEND TO EACH NEIGHBOR ONCE
  for(int i = 0; i < numNeighbors; i++)
  {
    //FIND THE CORRECT NEIGHBOR TO SEND
    //FIRST FIND THE FIRST NEIGHBOR THAT HAS NOT BEEN TRIED
    for(int j = 0; j < numNeighbors; j++)
    {
      //IF THE NEIGHBOR [j] HAS NOT FAILED TO ACK YET, START WITH THIS ONE AND
BEGIN COMPARING
      if(noACK[j] == false)
      {
        currNeighbor = j;
        break;
      }
    }

    //LOOP TO FIND THE NEIGHBOR WITH THE LOWEST HOP COUNT THAT HAS NOT BEEN TRIED
YET
    for(int j = (currNeighbor + 1); j < numNeighbors; j++)
    {
      //IF THE NEIGHBOR HAS NOT BEEN TRIED YET
      if(noACK[j] == false)
      {
        //AND THE HOP COUNT FOR THIS NEIGHBOR IS LESS THAN THE CURRENT BEST
        if(hopArray[readoutIndex][j] < hopArray[readoutIndex][currNeighbor])
        {
          //THIS IS THE NEW BEST
          currNeighbor = j;
        }
      }
    }
    //TRY TO SEND TO THE CURRENT NEIGHBOR
     Serial.println((String)"Sending to nodeID " + neighbors[currNeighbor]);
    if(radio.sendWithRetry(neighbors[currNeighbor], (const void*)(&theData),
sizeof(theData), 5, ACK_TIME))
    {
      Serial.println((String)"ACK received from " + neighbors[currNeighbor]);
      //IF IT ACKED THEN SET REMAINING TRANSMISSIONS TO ZERO (THE PACKET WAS
SUCCESSFULLY TRANSMITTED SO THIS PACKET IS DONE) AND EXIT
      remainingTransmissions = 0;
      return;
    }
    //OTHERWISE, SET NOACK[] AND REPEAT
    else
    {
      Serial.println((String)"ACK NOT received from " + neighbors[currNeighbor]);
      noACK[currNeighbor] = true;
    }

  }
}

//RECEIVE METHOD THAT PUSHES RECEIVED PACKETS INTO A CIRCULAR ARRAY TO BE USED BOTH
 AS A QUEUE FOR TRANSMISSION AND AS A LIST OF PREVIOUSLY SEEN PACKETS (TO AVOID WAS
TE ASSOCIATED WITH RETRANSMISSION)
void checkReceivePush()
{
  if (radio.receiveDone())
  {
    Serial.print("Received Packet from ["); Serial.print(radio.SENDERID, DEC);
Serial.print("] ");
    Serial.print((String)" [RX_RSSI:" + (radio.readRSSI()) + "]");
    /*if (promiscuousMode)//look here
    {
      Serial.print((String)"to [" + (radio.TARGETID, DEC) + "]");
    }*/

    if (radio.DATALEN == sizeof(Payload))
    {
        rxData = *(Payload*)radio.DATA; //assume radio.DATA actually contains our
struct and not something else

        //ACK SHOULD BE SENT IF THE PACKET WAS VALID AND UNCORRUPTED, BUT SHOULD
NOT BE DEPENDENT UPON WHETHER OR NOT THIS PACKET HAS BEEN SEEN BEFORE
        if (radio.ACKRequested())
        {
          byte theNodeID = radio.SENDERID;
          radio.sendACK();
          Serial.println((String)" - ACK sent.");
        }

        //displayPacket(rxData);

        //CHECK TO SEE IF THIS PACKET IS NEW OR OLD
        bool isNewPacket = true;
        for(int i = 0; i < QUEUESIZE; i++)
        {
          //IF THIS PACKET'S NODEID AND UPTIME MATCH A PACKET IN MEMORY, IT IS OLD
          if((rxData.nodeId == packetQueue[i].nodeId) && (rxData.uptime ==
packetQueue[i].uptime))
          {
            isNewPacket = false;
            break;
          }
        }

        if(isNewPacket)//IF THE PACKET IS NEW, PUSH IT INTO MEMORY
        {
          Serial.println("Pushing Packet");
          digitalWrite(LED, HIGH);
          displayPacket(rxData);
          pushPacket(rxData);
          displayPacketQueue();
        }

    }
    else
      Serial.print("Invalid payload received, not matching Payload struct!");
    Serial.println();
    //Blink(LED,3);
  }
}


void checkPin()
{

  if(digitalRead(sensePin) == HIGH)//MODIFIED SO ONLY 1 PACKET IS SENT PER SENSOR
TRIGGER
  {
    if(sensorTriggered == false)
    {
      Serial.println((String)"Sensor triggered");
      sensorTriggered = true;
      lightOn();

      //createPacket();
      //Serial.println((String)"ISEMPTY? " + );
      for(int i = 0; i < NUMREADOUTS; i++)
      {
        createBigEPacket(readouts[i]);
        Serial.println((String)"Pushing packet: ");
        displayPacket(tempPacket);

        PacketLL.addToTail(tempPacket);
      }

      /*createBigEPacket(readouts[1]);

      Serial.println((String)"Pushing packet: ");
      displayPacket(tempPacket);

      PacketLL.addToTail(tempPacket);*/


      //displayPacketQueue();

    }
  }
  else
  {
    sensorTriggered = false;//RESET THE TRIGGER FOR THE INNER IF ONCE THE JUMPER IS
REPLACED
  }

}

void randomSleep()
{
  SLEEPTIME = random();

  for(int i = 0; i < SLEEPTIME; i++)
  {
  }

}

//NEIGHBOR DEFINING METHOD FOR DEMO1, DYNAMICALLY SIZES DOES NOT USE THE FIXED SIZE
 10 ARRAY LIKE BEFORE. ASSUMES ONLY CALLED ONCE, IF DONE FOR UPDATES SHOULD CALL RE
ALLOC TO AVOID LEAKS!
void setNeighbors()
{

  //create3by3_fourReadouts();
  create7by7_fourReadouts();

  //sets up a lookup table for each readout's row in hopArray and its nodeID
  readouts = (byte*)malloc(NUMREADOUTS);
  for(int i = 0; i < NUMREADOUTS; i++)
  {
    readouts[i] = (254 - i);
  }

  hopArray = new byte*[NUMREADOUTS];
  for(int i = 0; i < NUMREADOUTS; i++)
  {
    hopArray[i] = new byte[numNeighbors];
    for(int j = 0; j < numNeighbors; j++)
    {
      hopArray[i][j] = 255;
    }
  }

  displayHopArray();

  displayPacketQueue();

}

void displayHopArray()
{
  for(int i = 0; i < NUMREADOUTS; i++)
  {
    Serial.println(String(i) + ": ");
    for(int j = 0; j < numNeighbors; j++)
    {
      Serial.println(hopArray[i][j]);
    }
  }
}

//METHOD TO PRINT THE PACKET QUEUE
void displayPacketQueue()
{
  Serial.println("Packet Queue contains: ");
  for(int i = 0; i < QUEUESIZE; i++)
  {
    //Serial.print((String)" Packet contains: nodeId= " + packetQueue[i].nodeId + "
uptime= " + packetQueue[i].uptime + " sensor status= " +
packetQueue[i].sensorStatus);
    //Serial.println((String)" Destination: " + packetQueue[i].destinationNode + "
Event Time: " + packetQueue[i].eventHour + ":" + packetQueue[i].eventMinute + ":" +
packetQueue[i].eventSecond);
    displayPacket(packetQueue[i]);
  }
}

//METHOD TO PRINT A PACKET. PASSES BY REFERENCE, SO NOT PARTICULARLY COSTLY COMPARE
D TO DOING IT WITHOUT A METHOD.
void displayPacket(struct Payload& packetIn)
{

  Serial.print((String)"PACKET CONTAINS: nodeId= " + packetIn.nodeId + " control
byte= " + packetIn.controlByte + " uptime= " + packetIn.uptime + " hop Count= " +
packetIn.hopCount);
```

```
  Serial.println((String)" Destination: " + packetIn.destinationNode + " Event
Time: " + packetIn.eventHour + ":" + packetIn.eventMinute + ":" +
packetIn.eventSecond);

  Serial.print((String)"Data: [ ");
  for(int i = 0; i < DATAARRAYSIZE; i++)
  {
    Serial.print((String)packetIn.dataArray[i] + " ");
  }
  Serial.println("]");

}

void printTheData()
{
  Serial.println((String)"Destination: " + theData.destinationNode);
  Serial.println((String)"Data: ");
  for(int i = 0; i < DATAARRAYSIZE; i++)
  {
    Serial.print((String)theData.dataArray[i] + " ");
  }
  Serial.println();
}

void showNeighborList()
{
  Serial.print((String)"Neighbor List: < ");

  for(int i = 0; i < numNeighbors; i++)
  {
    Serial.print((String)neighbors[i] + " ");
  }
  Serial.println(">");
}

void resetHasACKed()
{
  for(int i = 0; i < numNeighbors; i++)
  {
    hasACKed[i] = false;
  }
}

void create3by3_oneReadout()
{
    switch(NODEID)
  {
    case 1:
      numNeighbors = 4;
      neighbors = (byte*)malloc(numNeighbors);
      neighbors[0] = 2;
      neighbors[1] = 3;
      neighbors[2] = 4;
      neighbors[3] = 5;
```

```
break;
case 2:
  numNeighbors = 5;
  neighbors = (byte*)malloc(numNeighbors);
  neighbors[0] = 1;
  neighbors[1] = 3;
  neighbors[2] = 4;
  neighbors[3] = 5;
  neighbors[4] = 6;
break;
case 3:
  numNeighbors = 4;
  neighbors = (byte*)malloc(numNeighbors);
  neighbors[0] = 1;
  neighbors[1] = 2;
  neighbors[2] = 5;
  neighbors[3] = 6;
break;
case 4:
  numNeighbors = 5;
  neighbors = (byte*)malloc(numNeighbors);
  neighbors[0] = 1;
  neighbors[1] = 2;
  neighbors[2] = 5;
  neighbors[3] = 7;
  neighbors[4] = 8;
break;
case 5:
  numNeighbors = 8;
  neighbors = (byte*)malloc(numNeighbors);
  neighbors[0] = 1;
  neighbors[1] = 2;
  neighbors[2] = 3;
  neighbors[3] = 4;
  neighbors[4] = 6;
  neighbors[5] = 7;
  neighbors[6] = 8;
  neighbors[7] = 9;
break;
case 6:
  numNeighbors = 5;
  neighbors = (byte*)malloc(numNeighbors);
  neighbors[0] = 2;
  neighbors[1] = 3;
  neighbors[2] = 5;
  neighbors[3] = 8;
  neighbors[4] = 9;
break;
case 7:
  numNeighbors = 5;
  neighbors = (byte*)malloc(numNeighbors);
  neighbors[0] = 4;
  neighbors[1] = 5;
  neighbors[2] = 8;
```

```
      neighbors[3] = 9;
      neighbors[4] = 252;
    break;
    case 8:
      numNeighbors = 6;
      neighbors = (byte*)malloc(numNeighbors);
      neighbors[0] = 4;
      neighbors[1] = 5;
      neighbors[2] = 6;
      neighbors[3] = 7;
      neighbors[4] = 9;
      neighbors[5] = 252;
    break;
    case 9:
      numNeighbors = 5;
      neighbors = (byte*)malloc(numNeighbors);
      neighbors[0] = 5;
      neighbors[1] = 6;
      neighbors[2] = 7;
      neighbors[3] = 8;
      neighbors[4] = 252;
    break;
  }


}

void create3by3_fourReadouts()
{
    switch(NODEID)
  {
    case 1:
      numNeighbors = 6;
      neighbors = (byte*)malloc(numNeighbors);
      neighbors[0] = 2;
      neighbors[1] = 3;
      neighbors[2] = 4;
      neighbors[3] = 5;
      neighbors[4] = 254;//NORTH
      neighbors[5] = 251;//WEST
    break;
    case 2:
      numNeighbors = 6;
      neighbors = (byte*)malloc(numNeighbors);
      neighbors[0] = 1;
      neighbors[1] = 3;
      neighbors[2] = 4;
      neighbors[3] = 5;
      neighbors[4] = 6;
      neighbors[5] = 254;//NORTH
    break;
    case 3:
      numNeighbors = 6;
      neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 1;
    neighbors[1] = 2;
    neighbors[2] = 5;
    neighbors[3] = 6;
    neighbors[4] = 254;//NORTH
    neighbors[5] = 253;//EAST
  break;
  case 4:
    numNeighbors = 6;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 1;
    neighbors[1] = 2;
    neighbors[2] = 5;
    neighbors[3] = 7;
    neighbors[4] = 8;
    neighbors[5] = 251;//WEST
  break;
  case 5:
    numNeighbors = 8;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 1;
    neighbors[1] = 2;
    neighbors[2] = 3;
    neighbors[3] = 4;
    neighbors[4] = 6;
    neighbors[5] = 7;
    neighbors[6] = 8;
    neighbors[7] = 9;
  break;
  case 6:
    numNeighbors = 6;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 2;
    neighbors[1] = 3;
    neighbors[2] = 5;
    neighbors[3] = 8;
    neighbors[4] = 9;
    neighbors[5] = 253;//EAST
  break;
  case 7:
    numNeighbors = 6;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 4;
    neighbors[1] = 5;
    neighbors[2] = 8;
    neighbors[3] = 9;
    neighbors[4] = 252;//SOUTH
    neighbors[5] = 251;//WEST
  break;
  case 8:
    numNeighbors = 6;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 4;
    neighbors[1] = 5;
```

```
      neighbors[2] = 6;
      neighbors[3] = 7;
      neighbors[4] = 9;
      neighbors[5] = 252;//SOUTH
    break;
    case 9:
      numNeighbors = 6;
      neighbors = (byte*)malloc(numNeighbors);
      neighbors[0] = 5;
      neighbors[1] = 6;
      neighbors[2] = 7;
      neighbors[3] = 8;
      neighbors[4] = 252;//SOUTH
      neighbors[5] = 253;//EAST
    break;
  }
}

void create7by7_fourReadouts()
{
  //NORTH (254) should connect to  3,  4,  5
  //EAST  (253) should connect to 21, 28, 35 neighbor[] = 253;
  //SOUTH (252) should connect to 45, 46, 47 neighbor[] = 252;
  //WEST  (251) should connect to 15, 22, 29 neighbor[] = 251;
  switch(NODEID)
  {
    case 1:
      numNeighbors = 3;
      neighbors = (byte*)malloc(numNeighbors);
      neighbors[0] = 2;
      neighbors[1] = 8;
      neighbors[2] = 9;
      break;
    case 2:
      numNeighbors = 5;
      neighbors = (byte*)malloc(numNeighbors);
      neighbors[0] = 1;
      neighbors[1] = 3;
      neighbors[2] = 8;
      neighbors[3] = 9;
      neighbors[4] = 10;
      break;
    case 3:
      numNeighbors = 6;
      neighbors = (byte*)malloc(numNeighbors);
      neighbors[0] = 2;
      neighbors[1] = 4;
      neighbors[2] = 9;
      neighbors[3] = 10;
      neighbors[4] = 11;
      neighbors[5] = 254;
      break;
    case 4:
      numNeighbors = 6;
      neighbors = (byte*)malloc(numNeighbors);
      neighbors[0] = 3;
      neighbors[1] = 5;
      neighbors[2] = 10;
      neighbors[3] = 11;
      neighbors[4] = 12;
      neighbors[5] = 254;
      break;
    case 5:
      numNeighbors = 6;
      neighbors = (byte*)malloc(numNeighbors);
      neighbors[0] = 4;
      neighbors[1] = 6;
      neighbors[2] = 11;
      neighbors[3] = 12;
      neighbors[4] = 13;
      neighbors[5] = 254;
      break;
    case 6:
      numNeighbors = 5;
      neighbors = (byte*)malloc(numNeighbors);
      neighbors[0] = 5;
      neighbors[1] = 7;
      neighbors[2] = 12;
      neighbors[3] = 13;
      neighbors[4] = 14;
      break;
    case 7:
      numNeighbors = 3;
      neighbors = (byte*)malloc(numNeighbors);
      neighbors[0] = 6;
      neighbors[1] = 13;
      neighbors[2] = 14;
      break;
    case 8:
      numNeighbors = 5;
      neighbors = (byte*)malloc(numNeighbors);
      neighbors[0] = 1;
      neighbors[1] = 2;
      neighbors[2] = 9;
      neighbors[3] = 15;
      neighbors[4] = 16;
      break;
    case 9:
      numNeighbors = 8;
      neighbors = (byte*)malloc(numNeighbors);
      neighbors[0] = 1;
      neighbors[1] = 2;
      neighbors[2] = 3;
      neighbors[3] = 8;
      neighbors[4] = 10;
      neighbors[5] = 15;
      neighbors[6] = 16;
      neighbors[7] = 17;
    break;
  case 10:
    numNeighbors = 8;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 2;
    neighbors[1] = 3;
    neighbors[2] = 4;
    neighbors[3] = 9;
    neighbors[4] = 11;
    neighbors[5] = 16;
    neighbors[6] = 17;
    neighbors[7] = 18;
    break;
  case 11:
    numNeighbors = 8;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 3;
    neighbors[1] = 4;
    neighbors[2] = 5;
    neighbors[3] = 10;
    neighbors[4] = 12;
    neighbors[5] = 17;
    neighbors[6] = 18;
    neighbors[7] = 19;
    break;
  case 12:
    numNeighbors = 8;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 4;
    neighbors[1] = 5;
    neighbors[2] = 6;
    neighbors[3] = 11;
    neighbors[4] = 13;
    neighbors[5] = 18;
    neighbors[6] = 19;
    neighbors[7] = 20;
    break;
  case 13:
    numNeighbors = 8;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 5;
    neighbors[1] = 6;
    neighbors[2] = 7;
    neighbors[3] = 12;
    neighbors[4] = 14;
    neighbors[5] = 19;
    neighbors[6] = 20;
    neighbors[7] = 21;
    break;
  case 14:
    numNeighbors = 5;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 6;
    neighbors[1] = 7;
    neighbors[2] = 13;
    neighbors[3] = 20;
    neighbors[4] = 21;
    break;
  case 15:
    numNeighbors = 6;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 8;
    neighbors[1] = 9;
    neighbors[2] = 16;
    neighbors[3] = 22;
    neighbors[4] = 23;
    neighbors[5] = 253;
    break;
  case 16:
    numNeighbors = 8;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 8;
    neighbors[1] = 9;
    neighbors[2] = 10;
    neighbors[3] = 15;
    neighbors[4] = 17;
    neighbors[5] = 22;
    neighbors[6] = 23;
    neighbors[7] = 24;
    break;
  case 17:
    numNeighbors = 8;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 9;
    neighbors[1] = 10;
    neighbors[2] = 11;
    neighbors[3] = 16;
    neighbors[4] = 18;
    neighbors[5] = 23;
    neighbors[6] = 24;
    neighbors[7] = 25;
    break;
  case 18:
    numNeighbors = 8;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 10;
    neighbors[1] = 11;
    neighbors[2] = 12;
    neighbors[3] = 17;
    neighbors[4] = 19;
    neighbors[5] = 24;
    neighbors[6] = 25;
    neighbors[7] = 26;
    break;
  case 19:
    numNeighbors = 8;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 11;
    neighbors[1] = 12;
    neighbors[2] = 13;
    neighbors[3] = 18;
    neighbors[4] = 20;
    neighbors[5] = 25;
    neighbors[6] = 26;
    neighbors[7] = 27;
    break;
  case 20:
    numNeighbors = 8;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 12;
    neighbors[1] = 13;
    neighbors[2] = 14;
    neighbors[3] = 19;
    neighbors[4] = 21;
    neighbors[5] = 26;
    neighbors[6] = 27;
    neighbors[7] = 28;
    break;
  case 21:
    numNeighbors = 6;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 13;
    neighbors[1] = 14;
    neighbors[2] = 20;
    neighbors[3] = 27;
    neighbors[4] = 28;
    neighbors[5] = 251;
    break;
  case 22:
    numNeighbors = 6;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 15;
    neighbors[1] = 16;
    neighbors[2] = 23;
    neighbors[3] = 29;
    neighbors[4] = 30;
    neighbors[5] = 253;
    break;
  case 23:
    numNeighbors = 8;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 15;
    neighbors[1] = 16;
    neighbors[2] = 17;
    neighbors[3] = 22;
    neighbors[4] = 24;
    neighbors[5] = 29;
    neighbors[6] = 30;
    neighbors[7] = 31;
    break;
  case 24:
    numNeighbors = 8;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 16;
    neighbors[1] = 17;
    neighbors[2] = 18;
    neighbors[3] = 23;
    neighbors[4] = 25;
    neighbors[5] = 30;
    neighbors[6] = 31;
    neighbors[7] = 32;
    break;
  case 25:
    numNeighbors = 8;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 17;
    neighbors[1] = 18;
    neighbors[2] = 19;
    neighbors[3] = 24;
    neighbors[4] = 26;
    neighbors[5] = 31;
    neighbors[6] = 32;
    neighbors[7] = 33;
    break;
  case 26:
    numNeighbors = 8;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 18;
    neighbors[1] = 19;
    neighbors[2] = 20;
    neighbors[3] = 25;
    neighbors[4] = 27;
    neighbors[5] = 32;
    neighbors[6] = 33;
    neighbors[7] = 34;
    break;
  case 27:
    numNeighbors = 8;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 19;
    neighbors[1] = 20;
    neighbors[2] = 21;
    neighbors[3] = 26;
    neighbors[4] = 28;
    neighbors[5] = 33;
    neighbors[6] = 34;
    neighbors[7] = 35;
    break;
  case 28:
    numNeighbors = 6;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 20;
    neighbors[1] = 21;
    neighbors[2] = 27;
    neighbors[3] = 34;
    neighbors[4] = 35;
    neighbors[5] = 251;
    break;
  case 29:
    numNeighbors = 6;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 22;
    neighbors[1] = 23;
    neighbors[2] = 30;
    neighbors[3] = 36;
    neighbors[4] = 37;
    neighbors[5] = 253;
    break;
  case 30:
    numNeighbors = 8;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 22;
    neighbors[1] = 23;
    neighbors[2] = 24;
    neighbors[3] = 29;
    neighbors[4] = 31;
    neighbors[5] = 36;
    neighbors[6] = 37;
    neighbors[7] = 38;
    break;
  case 31:
    numNeighbors = 8;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 23;
    neighbors[1] = 24;
    neighbors[2] = 25;
    neighbors[3] = 30;
    neighbors[4] = 32;
    neighbors[5] = 37;
    neighbors[6] = 38;
    neighbors[7] = 39;
    break;
  case 32:
    numNeighbors = 8;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 24;
    neighbors[1] = 25;
    neighbors[2] = 26;
    neighbors[3] = 31;
    neighbors[4] = 33;
    neighbors[5] = 38;
    neighbors[6] = 39;
    neighbors[7] = 40;
    break;
  case 33:
    numNeighbors = 8;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 25;
    neighbors[1] = 26;
    neighbors[2] = 27;
    neighbors[3] = 32;
    neighbors[4] = 34;
    neighbors[5] = 39;
    neighbors[6] = 40;
    neighbors[7] = 41;
    break;
  case 34:
    numNeighbors = 8;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 26;
    neighbors[1] = 27;
    neighbors[2] = 28;
    neighbors[3] = 33;
    neighbors[4] = 35;
    neighbors[5] = 40;
    neighbors[6] = 41;
    neighbors[7] = 42;
    break;
  case 35:
    numNeighbors = 6;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 27;
    neighbors[1] = 28;
    neighbors[2] = 34;
    neighbors[3] = 41;
    neighbors[4] = 42;
    neighbors[5] = 251;
    break;
  case 36:
    numNeighbors = 5;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 29;
    neighbors[1] = 30;
    neighbors[2] = 37;
    neighbors[3] = 43;
    neighbors[4] = 44;
    break;
  case 37:
    numNeighbors = 8;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 29;
    neighbors[1] = 30;
    neighbors[2] = 31;
    neighbors[3] = 36;
    neighbors[4] = 38;
    neighbors[5] = 43;
    neighbors[6] = 44;
    neighbors[7] = 45;
    break;
  case 38:
    numNeighbors = 8;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 30;
    neighbors[1] = 31;
    neighbors[2] = 32;
    neighbors[3] = 37;
    neighbors[4] = 39;
    neighbors[5] = 44;
    neighbors[6] = 45;
    neighbors[7] = 46;
    break;
  case 39:
    numNeighbors = 8;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 31;
    neighbors[1] = 32;
    neighbors[2] = 33;
    neighbors[3] = 38;
    neighbors[4] = 40;
    neighbors[5] = 45;
    neighbors[6] = 46;
    neighbors[7] = 47;
    break;
  case 40:
    numNeighbors = 8;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 32;
    neighbors[1] = 33;
    neighbors[2] = 34;
    neighbors[3] = 39;
    neighbors[4] = 41;
    neighbors[5] = 46;
    neighbors[6] = 47;
    neighbors[7] = 48;
    break;
  case 41:
    numNeighbors = 8;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 33;
    neighbors[1] = 34;
    neighbors[2] = 35;
    neighbors[3] = 40;
    neighbors[4] = 42;
    neighbors[5] = 47;
    neighbors[6] = 48;
    neighbors[7] = 49;
    break;
  case 42:
    numNeighbors = 5;
    neighbors = (byte*)malloc(numNeighbors);
    neighbors[0] = 34;
    neighbors[1] = 35;
    neighbors[2] = 41;
    neighbors[3] = 48;
    neighbors[4] = 49;
    break;
  case 43:
    numNeighbors = 3;
  neighbors = (byte*)malloc(numNeighbors);
  neighbors[0] = 36;
  neighbors[1] = 37;
  neighbors[2] = 44;
  break;
case 44:
  numNeighbors = 5;
  neighbors = (byte*)malloc(numNeighbors);
  neighbors[0] = 36;
  neighbors[1] = 37;
  neighbors[2] = 38;
  neighbors[3] = 43;
  neighbors[4] = 45;
  break;
case 45:
  numNeighbors = 6;
  neighbors = (byte*)malloc(numNeighbors);
  neighbors[0] = 37;
  neighbors[1] = 38;
  neighbors[2] = 39;
  neighbors[3] = 44;
  neighbors[4] = 46;
  neighbors[5] = 252;
  break;
case 46:
  numNeighbors = 6;
  neighbors = (byte*)malloc(numNeighbors);
  neighbors[0] = 38;
  neighbors[1] = 39;
  neighbors[2] = 40;
  neighbors[3] = 45;
  neighbors[4] = 47;
  neighbors[5] = 252;
  break;
case 47:
  numNeighbors = 6;
  neighbors = (byte*)malloc(numNeighbors);
  neighbors[0] = 39;
  neighbors[1] = 40;
  neighbors[2] = 41;
  neighbors[3] = 46;
  neighbors[4] = 48;
  neighbors[5] = 252;
  break;
case 48:
  numNeighbors = 5;
  neighbors = (byte*)malloc(numNeighbors);
  neighbors[0] = 40;
  neighbors[1] = 41;
  neighbors[2] = 42;
  neighbors[3] = 47;
  neighbors[4] = 49;
  break;
case 49:
        numNeighbors = 3;
        neighbors = (byte*)malloc(numNeighbors);
        neighbors[0] = 41;
        neighbors[1] = 42;
        neighbors[2] = 48;
        break;
    }

}
```

Simulator Source Code:

```
;"Remote Sensing in Arctic Regions" Simulation" - M Devins
;  Simulation simulating:
;   -neighbor detection
;   -a method to find shortest paths to each readout node
;   -a method to send a sensor message to each readout
;   -the beginnings of a sensor verification method using overlapping waves
;   -the ability to export the generated network configuration for use on the
hardware implementation of this 7x7 network
;   -can work its way around dead nodes
;   -Files dumped by the simulator now follow the readout numbering convention, 254
and down.
;   -
Files also reflect the order of the nodeIDs on the board, right to left starting in
 the upper right corner

breed [sensors sensor]
breed [readouts readout]

globals
[
  COUNTER
  ;MESSAGEHOPS
  AVG_LINK_LENGTH
  AVG_DEGREE

  NORTH
  SOUTH
  EAST
  WEST
]

sensors-own
[
  neighborList
  distToN
  distToE
  distToS
  distToW
  distList
  hopDist
]

readouts-own[
  triangulation
  hopLength
  hops
]

to setup
  no-display
  ;;;;;;;;;;;;;;;;;;;;;;;;
  clear-all
  ;;;;;;;;;;;;;;;;;;;;;;;;
  set-patch-size 50
  resize-world 0 (GRIDSIZE + 1) 0 (GRIDSIZE + 1)

  ask patches [set pcolor (95 * ((pxcor + pycor) mod 2))]; make patches look like a
 checkerboard for clarity

  let gridYInc (((max-pycor - min-pycor) - 1) / GRIDSIZE)
  ;print gridYInc
  let gridXInc (((max-pxcor - min-pxcor) - 1) / GRIDSIZE)
  ;print gridXInc e
  create-sensors (GRIDSIZE * GRIDSIZE)
  [
    set color yellow
    set shape "dot"
    set size .4
    set heading 0
    if NetworkType = "Regular"[
      setxy (who mod GRIDSIZE + 1) (floor (who / GRIDSIZE + 1))
    ]
    if NetworkType = "RegularWiggled"[
      setxy (random-float wiggle - wiggle / 2 + who mod GRIDSIZE + 1 ) (random-
float wiggle - wiggle / 2 + floor (who / GRIDSIZE + 1 ))
    ]
    if NetworkType = "MattLayout" [
      setxy ((floor( ((GRIDSIZE * GRIDSIZE) - 1 - who) mod GRIDSIZE) * gridXInc) +
.5 + (random-float 1 * gridXInc)) ((floor(((GRIDSIZE * GRIDSIZE) - 1 - who) /
GRIDSIZE) * gridYInc) + .5 + (random-float 1 * gridYInc))
    ]
    set neighborList []
    set distList [[][][][]]
    set hopDist 9999;
  ]

  set COUNTER 0;
  ;set MESSAGEHOPS 0

  create-readouts 4 [
    set color red
    set shape "dot"
    set size 1
  ]

  set NORTH (readout (GRIDSIZE * GRIDSIZE))
  set EAST  (readout ((GRIDSIZE * GRIDSIZE) + 1))
  set SOUTH (readout ((GRIDSIZE * GRIDSIZE) + 2))
  set WEST  (readout ((GRIDSIZE * GRIDSIZE) + 3))

  ask NORTH[
    setxy (max-pycor / 2) (max-pxcor)
  ]
  ask EAST[
    setxy (max-pycor) (max-pxcor / 2)
  ]
  ask SOUTH[
    setxy (max-pycor / 2) (0)
  ]
  ask WEST[
    setxy (0) (max-pxcor / 2)
  ]

  set COUNTER 0;
  display

  ; This just cuts down on time by finding agent sets that are reachable by this
node. This way I only check the subsets in range that are linked.
    if ConnectivityType = "TransmitRange" [
      ask turtles[
        ask other sensors with [distance myself <= TRANSMISSIONRANGE] [
          create-link-with myself
        ]
      ]
    ]

    ;; this is messy since, the network is created asymetrically.
    ;; nodes might make degree connections, but they can be linked to from other no
des with my-links < min
    ;; for each node, get all sensors, order then by distance and connect a sensor
with up to degree
    ;; closest nodes
    if ConnectivityType ="DynamicRange"[
      ask turtles[
        let edgeCandidates[]
        ask other turtles[
          ;; get distances to all other sensors and order them by distance
          set edgeCandidates lput (list who distance myself) edgeCandidates
          set edgeCandidates sort-by [item 1 ?1 < item 1 ?2] edgeCandidates
        ]
        let i 0
        let f true
        ;; as long as you don't exceed degree, connect to the degree num of closest
sensors
        ;; unless edges are unreasonable far
        while [i < degree and f][
          let s1 item 0 item i edgeCandidates
          if (count my-links < degree)  [
            ;; this is the unreasonable check, if subsequent nodes are much far
away than current edge do not connect.
            ;; this should take care of sensors on the edges having to connect to
sensor way too far to satisfy the degree cond
            ifelse (item 1 item i edgeCandidates / item 1 item (i + 1) edgeCandidat
es > 0.5673)[
              create-link-with turtle s1][
              create-link-with turtle s1
              set f false
              ]
          ]
          set i i + 1
        ]
      ]
    ]
  ;print count links;
  set AVG_LINK_LENGTH mean [link-length] of links
  set AVG_DEGREE mean [count my-links] of turtles ;count links /  count turtles
  plot-histograms
end

to findShortestPaths
  ask NORTH[
    FSP self 0 0
  ]

  ask EAST[
    FSP self 1 0
  ]

  ask SOUTH[
    FSP self 2 0
  ]

  ask WEST[
    FSP self 3 0
  ]

  ;TELL EACH NODE TO SEARCH FOR 0s IN ITS DISTANCE TABLE, IF THEY ARE FOUND THE OTH
ER TABLES SHOULD BE CHANGED TO NOT READ -1
  ask sensors[
    foreach (n-values (count readouts) [?])[
      while[position -1 (item ? distList) != false][
        set distList replace-item ? distList (replace-item (position -1 (item ?
distList)) (item ? distList) (9999))
      ]
    ]
  ]
end

to FSP[sensorIn readoutNum hopCount]
```

```
  ask sensorIn[
    ask link-neighbors with [[breed] of self = sensors][

      ;IF THIS NODE'S ID IS NOT FOUND IN ITS NEIGHBOR'S LIST, TELL THE NEIGHBOR TO
ADD IT TO THE END OF ITS LIST
      if(position ([who] of myself) neighborList = false) [
        set neighborList lput ([who] of myself) neighborList

        ;SINCE THE NEIGHBOR HAD NOT BEEN SEEN BEFORE, ADD THE HOP COUNT SINCE THIS
MUST BE THE SHORTEST THAT HAS BEEN SEEN SO FAR
        ;A) INCREASE THE SIZE OF THE OTHER READOUT LISTS TO ACCOMODATE THIS NEW NEI
GHBOR
        foreach (remove readoutNum (n-values (count readouts) [?]))[
          set distList replace-item ? distList (lput -1 (item ? distList))
        ]
        ;B) APPEND THIS HOPCOUNT TO THE LIST FOR THIS READOUT
        set distList replace-item readoutNum distList (lput hopCount (item
readoutNum distList))

        FSP self readoutNum (hopCount + 1)
      ]

      ;IF THE COUNT IN DISTLIST[READOUTNUM][POSITION OF THE NODE THAT THE HOP CAME
FROM] IS GREATER THAN (hopCount + 1)
      let neighborIndex position ([who] of myself) neighborList

      if( (item neighborIndex (item readoutNum distList) > (hopCount)) or (item
neighborIndex (item readoutNum distList) = (-1))) [
        set distList replace-item readoutNum distList (replace-item neighborIndex
(item readoutNum distList) (hopCount))
        FSP self readoutNum (hopCount + 1)
      ]
    ]
  ]
end

to-report sendMessage[currSensor]

  ;reset the nodes' colors
  let errorList[]
  let returnError 0
  ask sensors [
    set color yellow
    if(shape = "square")[die]
  ]
  ask readouts[
    set color red
  ]
  ask links[
    set color 5
  ]

  ask currSensor
  [
    ;set the originating node's color to white
    set color white
    ;for loop, goes through each readout node
    foreach (n-values (count readouts) [?])
    [
      let tempNeighborList neighborList
      let currSubList (item ? distList);the hop array for the current readout

      carefully
      [
        let currBestIndex (position (min currSubList) currSubList);the index of the
initial best neighbor
        let nextNode (item currBestIndex tempNeighborList);the neighbor list and ho
p array are aligned so that the index is the same for neighbor ID and its
corresponding hop count
         ;; I don't know what this does, It might be a check if not graph, and path
cannot reach the readout node
         ;; needs testing o.w. leave out - seems to work for connected graphs just
fine
;        while[turtle nextNode = NOBODY][
;        ;remove the current neighbor from the temp copy of the list, because it is
invalid (NOBODY)
;        set currSubList remove-item currBestIndex currSubList
;        set tempNeighborList remove-item currBestIndex tempNeighborList
;
;        ;reset to the new best
;        set currBestIndex (position (min currSubList) currSubList)
;        set nextNode (item currBestIndex tempNeighborList)
;            ]

          ask (link-with turtle nextNode)
          [
            set color (13 + (? * 40))
          ]
          sendMessageTo (nextNode) (?) (list [link-length] of link-with turtle
nextNode)
        ]
      [
        show "Not connected to readouts"
      ]
    ]
    ;create list of readouts here that are within range of sending node then call e
rror function with this list****************************
      ask currSensor
      [
        ask readouts with [distance myself <= size / 2]
        [
          set errorList fput self errorList
        ]
      ]
      show(errorList)
      ifelse(length errorList > 1)
      [
        set returnError findError (errorList) (currSensor) ;***********************
******
      ]
      [
        set returnError 1000 ;#################issue with bad reporting using this
method and low connectivity
      ]
  ]
  report returnError
end

to sendMessageTo[currSensor destinationReadout path]
  ask turtle currSensor[
    set color (13 + (destinationReadout * 40))

    ifelse([breed] of turtle currSensor != readouts)[
      let tempNeighborList neighborList
      let currSubList (item destinationReadout distList);the hop array for the
current readout

      let currBestIndex (position (min currSubList) currSubList);the index of the
initial best neighbor
      let nextNode (item currBestIndex tempNeighborList);the neighbor list and hop
array are aligned so that the index is the same for neighbor ID and its
corresponding hop count

      ask (link-with turtle nextNode)[
        set color (13 + (destinationReadout * 40))
        set path lput link-length path
      ]

      ;set MESSAGEHOPS MESSAGEHOPS + 1
      sendMessageTo (nextNode) (destinationReadout) path
    ]
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    [
      ;;ask patches within radius of heuristic to set color yellow
      ask readOut currSensor [
        set shape "MyCircle"
        set hops length path
        set hopLength path
        set triangulation (2.37997896862097 * hops + 0.030071762086814 * hops ^ 3 -
 0.0481647006066521 - 0.00157569233056816 * hops ^ 4 -
 0.175050402785827 * hops ^ 2) + 0.2
        ;0.277029863730913 + 1.96667632290825 * hops + 0.000832137259970713 * hops
^ 3 * sin ( sin ( 0.288639574476076 * hops * sin ( 5.57735250744605 +
0.355845682087151 * hops ))) + 0.7
        ;min( list (2.07076631531666 * hops) 21.7660772395908) + 0.2 ;80%class +5%
Really tight
        ;max (list (0.431459327412312 + 1.87592726741594 * hops) (min (list
21.4108445807148 (2.07231525034307 * log ( floor ( exp ( hops))) 2 )))) ;; 100% +5%
very loose
        ;2.16504272785232 * hops + tanh ( 32.3146716158353 -
 2.97934869050464 * hops) - 0.955208702293253 ;; 100% +10%
```

```
        ;2 * (0.25256283423033 + 0.881612245287836 * Hops + 0.00225440775191647 * (
Hops ^ 3) - 1.54862701832959e-5 * Hops ^ 5) * 1.001
        ;2 * ( 0.486 * hops + 0.073 * (hops ^ 2) + 1.87 * exp(2.17E-8 * (hops ^ 8))
- 1.39 ) * 1.10
        ;2 * ( 0.526 * hops + 0.0655 * (hops ^ 2) + 1.57 * exp(6.05E-15 * (hops ^
17)) - 1.13 ) * 1.15
        set size triangulation
      ]
    ]
    ;;;;;;;;;;;;;;;;;;;;;;;;;;
  ]
  plot-histograms
end

;; take this out and put it into an aux file that will do triangulation and triang
error
;; load separately from external file
to-report tanh [x]
  let expPow2 exp ( 2 * x)
  report ( expPow2 - 1 ) / ( expPow2 + 1 )
end


to waveProp[currSensor hopDistance]
  ask currSensor[
    ask link-neighbors[
      if(color = yellow)[
        set color gray
      ]
      if((hopDistance - 1) > 0)[
        waveProp self (hopDistance - 1)
      ]
    ]
  ]
end

to createFile
  if(file-exists? "./viewDump.png") [
    file-delete "./viewDump.png"
  ]

  export-view "./viewDump.png"

  if(file-exists? "./worldDump.csv") [
    file-delete "./worldDump.csv"
  ]

  export-world "./worldDump.csv"

  if(file-exists? "./SensorNWDump.rtf") [
    file-delete "./SensorNWDump.rtf"
  ]
  file-open "./SensorNWDump.rtf"
  foreach sort-on [who] sensors [
    ;file-type [who] of ?
    ask ? [
      file-type who + 1
      foreach ([who] of link-neighbors) [
        file-type ","
        ;if the neighbor is less than (GRIDSIZE * GRIDSIZE) it is a sensor
        ifelse (? < (GRIDSIZE * GRIDSIZE)) [
          file-type (? + 1)
        ]
        ;Otherwise, it is a readout and should be listed from 254 downward for
hardware compatibility
        [
          file-type (254 - (? - (GRIDSIZE * GRIDSIZE)))
        ]
        ;file-type ?
      ]
    ]
    file-print ""
  ]

  file-flush
  file-close
end

to loadFromFile
  let inputFile user-file
  ifelse(inputFile != false) [
    ;User selected a valid file
    import-world inputFile
  ]
  [
    ;user did not select a valid file
    user-message "No file selected"
  ]

end

;Mouse kills nodes when clicked. I am no comedian, but I try...
to ratPoison
  if(mouse-down?) [
    ask turtles with [(distancexy mouse-xcor mouse-ycor) < [size] of self / 4][
      die
    ]
  ]
end

to testUserOneOf
  print word "User selected " (user-one-of "Select an option" ["A" "B" "C" "D"
"None"])
end

to testUserInput
  print word "User typed: " (user-input "Type something")
```

```
end


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
to analyze
  let numNodes count turtles
  let numLinks count links
  let linksLength 0
  let numDegrees 0
  let avgDegrees 0
  set AVG_LINK_LENGTH 0

  ask links[
    set linksLength linksLength + link-length
  ]

  ask turtles[
    set numDegrees numDegrees + (count my-links)
  ]

  set AVG_LINK_LENGTH linksLength / numLinks
  show word "total links / by number of nodes = " (linksLength / numNodes )
  show word "nodes * links / area = " ((numNodes * numLinks) / (GRIDSIZE *
GRIDSIZE))
  show word "nodes * average link length / area = " ((numNodes * (linksLength /
numLinks)) / (GRIDSIZE * GRIDSIZE))
  show word "nodes * average degree / area = " ((numNodes * avgDegrees) / (GRIDSIZE
* GRIDSIZE))
  ;show numDegrees
  ;show avgDegrees
  ;show linksLength

end


;this method checks each node individually to determine how well the triangulation
method works inside sendMessageTo method
to-report checkTriangulation

  carefully [file-delete "ErrorStats.txt"][]
  file-open "ErrorStats.txt"
  let statCount 0
  let success 0

  ;loop here
  while[statCount < checkNum]
  [
  setup
  findShortestPaths

  let cnt_rea count readouts

  let cnt_links 0
  let cnt_degree 0
  let tria_stats []
  ;file-print "sensorID xcor ycor avgLinkLen avgDegree PerfectLinks PefrectDegree
TransRange Hops1 E1 Hops2 E2 Hops3 E3 Hops4 E4"
  if GRIDSIZE = 10 [
    ;let cnt_links count links - sum [count link-neighbors] of READOUTS
    set cnt_links 342
    set cnt_degree 8 ]
  if GRIDSIZE = 7 [
    set cnt_links 156
    set cnt_degree 8]

  foreach [who] of sensors [
    let sen ?
    let tria_ok 0
    let tria_not 0
    let tria_cnt 0     ;send messages to readout nodes
    let errorVal sendMessage(sensor sen)
    ;;;

     ;stats array here_____________________________________________________________
____________________________________________________________________________________
______________________________________________________
    set tria_stats fput errorVal tria_stats


    ;;;
    ;file-type (word [who] of sensor sen " " [xcor] of sensor sen " " [ycor] of
sensor sen " " AVG_LINK_LENGTH " " AVG_DEGREE " " cnt_links " " cnt_degree " "
TransmissionRange " " errorVal " ")
    foreach [who] of readouts [
      let rea ?
      let rea_size (([triangulation] of readout rea) / 2)
      let rea_hops [hops] of readout rea

      ;make links between sensors to readouts
      ask sensor sen [
        let dif distance readout rea - rea_size
        ;file-type (word rea " " rea_hops " " distance readout rea " " [sum
hopLength] of readout rea " " dif " " ); get the difference between triangultion
and distance if <0 triangulated, >0 o.w.
        ifelse (dif < 0) [

          set tria_ok tria_ok + dif
          set tria_cnt tria_cnt + 1
        ][
          set tria_not tria_not + dif
        ]
      ]
    ]
    ifelse tria_cnt > 2 [
      ;file-type (word tria_cnt " " (tria_ok / tria_cnt) " ")
      set success success + 1
    ][
      ;file-type (word tria_cnt " " (tria_not / (cnt_rea - tria_cnt)) " ")
    ]
    ;file-print " "
  ]
  plot-histograms
  show (word "min error " min tria_stats " max error " max tria_stats " Avg error "
mean tria_stats)
  file-print(word min tria_stats " " max tria_stats " " mean tria_stats " ")
  ;file-print("")

  set statCount statCount + 1
]

  ;show length tria_stats
  report success
end

; This sets up 100 grphs and calls checkTriangulation for each of the graphs
to TriangulationStats
  no-display
  carefully [file-delete "TriangualationDump.txt"][]
  file-open "TriangualationDump.txt"
  file-print (word wiggle "sensorID xcor ycor avgLinkLen avgDegree PerfectLinks
PefrectDegree TransRange errorVal ReaID1 Hops1 Dist1 PathDist1 E1 ReaID2 Hops2
Dist2 PathDist2 E2 ReaID3 Hops3 Dist3 PathDist3 E3 ReaID4 Hops4 Dist4 PathDist4 E4
CorTria AvgTriaError")
  let hist[]
  let cnt 10
  let statLoop 0
  let totalGood 0
  let averageGood 0
  let badSetup 0
  let myLinks 0

  let TxRange TransmissionRange

  while [statLoop < NumTriangTests][
    set TransmissionRange TxRange
    repeat 5 [
      setup
      ;linkNeighbors
      findShortestPaths
      ;repair nodes w/ disconnected nodes and link w/ closest sensor
      ask turtles with [ count my-links < 2][
        create-link-with min-one-of other sensors [distance myself]
      ]
      set totalGood totalGood + checkTriangulation
      set statLoop statLoop + 1
      ;set hist sentence pathsHist hist
      plot-histograms
      file-flush

      set TransmissionRange TransmissionRange + 0.1
    ]
    set TransmissionRange TxRange
  ]
  set averageGood totalGood / NumTriangTests
  show (word "Average good triangulations: " averageGood)
  show (word "Average Percent of good triangultions: "  (averageGood / count
sensors))
  file-close-all
  display
  ;set pathsHist hist ; / n-values NumTriangTests [NumTriangTests]
  plot-histograms
end

to plot-histograms
  ;set-histogram-num-bars 20
  ;set-current-plot-pen "linksLength"
  ;histogram [link-length] of links
  ;set-current-plot-pen "pathsLength"
  ;histogram pathsHist
  ;set-current-plot-pen "degree"
  ;histogram [count my-links] of turtles
end

to-report findError[errorList currSensor] ;???pass list as parameter
  ;***************APPROACH 1***************
  ;****vaiables**********
  ;let circle1_x
  ;let circle1_y
  ;let circle1_r
  ;let circle2_x
  ;let circle2_y
  ;let circle2_r
  ;let y
  ;let x
  ;let point1_x
  ;let point1_y
  ;let point2_x
  ;let point2_y

  ;******terms**********
  ;let term1
  ;let term2

  ;******equation for terms****
  ;set term1 ((circle1_x - circle2_x) / (circle1_y - circle2_y)) ;
  ;set term2 ((circle1_r ^ 2 - circle2_r ^ 2) + (circle2_x ^ 2 -
 circle1_x ^ 2) + (circle1_y ^ 2 - circle2_y ^ 2)) / (2(circle1_y - circle2_y)))

  ;TODO
  ;    y = (term1 * x) + term2 ***solve for x by plugging this in to one of the
circle equations
  ;                              ***then plug the two x values from that quad equatio
n to solve for y
  ;                              ***TODO figure out the equation of plugging this int
on of the circle
```

```
 ;                             ***equations and then solve that quadriatic equation
plug those two
 ;                             ***into formula for y to get the two points....


 ;***************APPROACH 2***************
 let done false
 let start1 0
 let currentIndex 1
 let stop1 length errorList - 1
 let minError []
 let minErrorNode[]
 let currError 100
 ;let circle1 item start1 errorList
 ;let circle2 item start1 errorList
 let Dist 0    ;distance between circles(nodes)
 let greek 0;some crazy greek variable
 let r0 0   ;circle1_r
 let r1 0   ;circle2_r
 let a 0    ;circle1_x
 let b 0    ;circle1_y
 let c 0    ;circle2_x
 let d 0    ;circle2_y
 let point1_x 0
 let point2_x 0
 let point1_y 0
 let point2_y 0

 foreach errorList
 [
   let circle1 ?
    foreach errorList
    [
      let circle2 ?
      if(circle1 != circle2)
      [
        ;show "im here"
        ask circle1
        [
          set a xcor
          set b ycor
          set r0 size / 2
        ]
        ask circle2
        [
          set c xcor
          set d ycor
          set r1 size / 2
        ]
        ask circle1
        [ create-link-with circle2
          ask (link-with circle2)[set Dist link-length]
        ]
```

```
         ;do math... add calculations to list or set min error to
variable*************
         set greek (1 / 4) * (sqrt abs(((Dist + r0 + r1) * (Dist + r0 - r1) * (Dist
- r0 + r1) * ((Dist * -1) + r0 + r1 ))))
         set point1_x ((a + c) / 2) + (((c - a) * (r0 ^ 2 -
 r1 ^ 2)) / (2 * Dist ^ 2)) + 2 * ((b - d) / (Dist ^ 2)) * greek
         set point2_x ((a + c) / 2) + (((c - a) * (r0 ^ 2 -
 r1 ^ 2)) / (2 * Dist ^ 2)) - 2 * ((b - d) / (Dist ^ 2)) * greek
         set point2_y ((b + d) / 2) + (((d - b) * (r0 ^ 2 -
 r1 ^ 2)) / (2 * Dist ^ 2)) + 2 * ((a - c) / (Dist ^ 2)) * greek
         set point1_y ((b + d) / 2) + (((d - b) * (r0 ^ 2 -
 r1 ^ 2)) / (2 * Dist ^ 2)) - 2 * ((a - c) / (Dist ^ 2)) * greek
         show (word "pt1_xy pt2_xy " point1_x " " point1_y " " point2_x " "
point2_y)

         hatch 1
         [
           carefully
           [
             ;set xcor abs (point1_x + point2_x) / 2
             ;set ycor abs (point1_y + point2_y) / 2
             setxy point1_x point1_y
           ]
           []

           set color blue
           set shape "square"
         ]

         hatch 1
         [
           carefully
           [
             ;set xcor abs (point1_x + point2_x) / 2
             ;set ycor abs (point1_y + point2_y) / 2
             setxy point2_x point2_y
           ]
           []

           set color blue
           set shape "square"
         ]

         hatch 1
         [
           carefully ;is 0 error coming from here?
           [
             ;set xcor abs (point1_x + point2_x) / 2
             ;set ycor abs (point1_y + point2_y) / 2
             setxy ((point2_x + point1_x) / 2) ((point2_y + point1_y) / 2)
           ]
           [
             setxy 0 11
           ]
           set color yellow
           set shape "square"
           ;ask currSensor [show xcor]

           ;set minError fput (currSensor distance myself) minError
           ;perhaps an if statement here if xcor = 0 and ycor = 11 set currError
1000
else...***************************************************************************
*******************************************************
           set currError distancexy [xcor] of currSensor [ycor] of currSensor
           set minError fput currError minError
           set minErrorNode fput who minErrorNode
          ]

         ;**************TODO measure dist between points and add calculations to li
st or set min error to variable*************

         ask links with [link-length > TRANSMISSIONRANGE] [ die ]
       ]
     ]
  ]
  ;carefully
  ;[
    let errorID item (position (min minError) minError) minErrorNode
    ask turtle errorID [set color green]
    show (word min minError  " :Error  Node: "  errorId)
    ask turtles with [shape = "square"][die]
    report min
minError;/////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////
////////////////
  ;]
  ;[report 1000]
end
```

References

[1] Remote Sensing and Networks. Retrieved February 6, 2016 from https://cnr.ncsu.edu/geospatial/frontiers/remote-sensing-sensor-networks/

[2] Homeland Security. Retrieved February 6, 2016 from http://www.dhs.gov/science-and-technology/centers-excellence

[3] All about Moteino. Moteino description and secifications. Retrieved February 9, 2016, from http://lowpowerlab.com/moteino/

[4] U.S. Immigration. Retrieved February 9, 2016 from https://www.usimmigration.com/illegal-immigrants-through-canada.html

[5] Recent Developments in High Performance Computing for Remote Sensing: A Review Retrieved February 10, 2016 from http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5981407

[6] NetLogo Modeling. (n.d.). Retrieved February 12, 2016, from http:// http://ccl.northwestern.edu/netlogo/

[7] Tutorial – Agent Based Modeling. (n.d.). Retrieved February 20, 2016, from www.mcs.anl.gov/~leyffer/listn/slides-06/MacalNorth.pdf

[8] Joseph Phillips (2003). PMP Project Management Professional Study Guide. McGraw-Hill Professional, 2003. ISBN 0-07-223062-2 p.354.

[9] http://www.seguetech.com/blog/2013/02/05/Project-manager-role-software-development

[10] Harold Kerzner (2003). Project Management: A Systems Approach to Planning, Scheduling, and Controlling (8th ed.). Wiley. ISBN 0-471-22577-0.

[11] Kaner, Cem (November 17, 2006). "Exploratory Testing" (PDF). Florida Institute of Technology, Quality Assurance Institute Worldwide Annual Software Testing Conference, Orlando, FL. Retrieved March 06, 2016.

[12] Instructables. Retrieved March 3, 2016 from http://www.instructables.com/id/HOW-TO-use-the-ARDUINO-SERIAL-MONITOR/