Scalable Point-to-Point LoRa P2P Sensor Network Via Flood-Messaging

Peter G. Raeth
Independent Researcher and Educator

Proof of Concept

Abstract

This project is a proof-of-concept that builds a peer-to-peer network of LoRa transceivers to support remote sensing and data transport. It uses flood-messaging to eliminate the need for more-costly LoRaWAN and third-party services. We begin with basics and gradually build the network. After having a working network, we add options to demonstrate the network's capabilities. Each section of this document takes a specific step forward and specifies the required hardware, associated documentation, and reliable sources of purchase. Project material is posted at the GitHub Repository. This present document is meant to be active in the sense that links are provided where necessary for additional details. While this pdf form of the document can be read without an internet connection, getting full value requires one. Let's dig in and peel back the onion.

Table of Contents

Table of Figures

Project Contributors

A paper by these three professors inspired the approach used by this project:

- Professor Philip Branch, School of Science, Computing and Engineering Technologies, Swinburne University of Technology, Melbourne, Australia
- Professor Binghao Li, Faculty of Engineering, University of New South Wales, Sydney, Australia
- Professor Kai Zhao, Faculty of Engineering, University of New South Wales, Sydney, Australia

Their well-written paper discussed the design, implementation, and testing of their LoRa flood-messaging network. The citation and link to their paper is: Branch, P., Li, B., Zhao, K. (2020) A LoRa-Based Linear Sensor Network for Location Data in Underground Mining. MDPI, Telecom, 1(2), 68-79, https://www.mdpi.com/2673-4001/1/2/6.

Introduction

Flood-Messaging, in its simplest form, is like Ethernet's UDP networking in that delivery of messages is highly likely but not absolutely guaranteed. Also, there is no effort to maintain message order. A source broadcasts messages. Those messages are received and rebroadcast by relay nodes. Destination (Basestation) nodes make use of message contents. Over time, messages age out of the system. Figure 1 Illustrates the basic idea and the three types of network nodes involved.

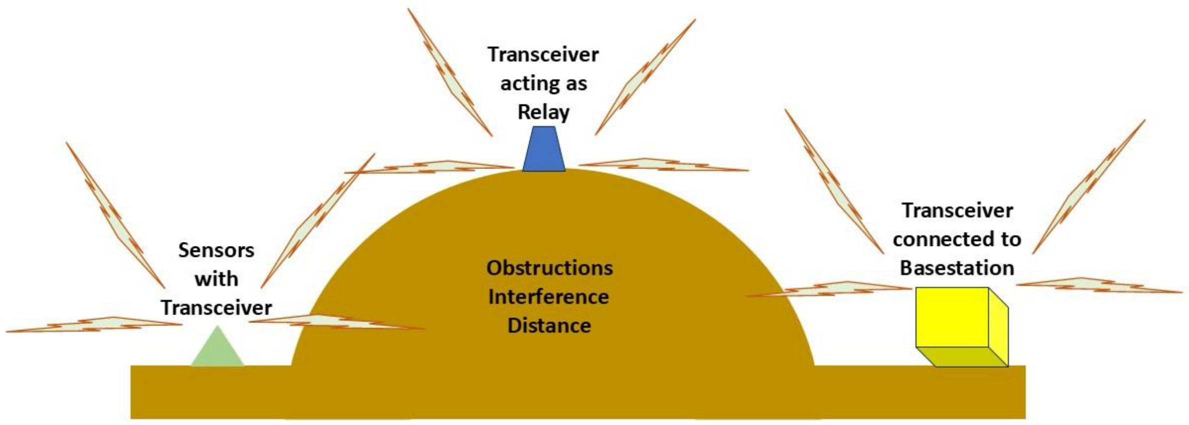
Transceiver acting as Relay
Sensors with Transceiver
Obstructions Interference Distance
Transceiver connected to Basestation

Figure 1. Fundamental idea behind LoRaP2P flood-messaging.

To dig deeper into the concepts and practical implementation of LoRaP2P flood-messaging, start by reading the paper by Branch, Li, and Zhao. Then read about its evolution in the paper by Raeth and Branch. In this present document, we get down into the weeds of realization via software and hardware. To begin, we put together modules that provide basic functionality. Then we build an inexpensive grassroots network. We continue by adding options that illustrate what can be done with the network. The Appendix contains a spreadsheet that cites each piece of hardware needed and reliable sources. Part-specific documentation is on each product page or linked from there.

Lets begin by laying the groundwork.

Laying the Groundwork

We start by examining the learning background one needs to proceed beyond plug-and-play. Then we put components together to build a simple functional transceiver that broadcasts sensor data. Be sure to read the entire section of interest before proceeding with a given step. Nothing beats understanding before doing. You will gain insights that help you as you proceed.

What You Need to Know

New projects require us to learn more than we may already know. They also lead to hardware purchases as well as writing and installing software. We talk now about those topics.

Education

Although we start with simple steps and little hardware, and hope for plug-and-play, understanding the guts of the software and modifying it is not for beginners. Where appropriate, reference is made to courses at Saylor Academy. Course completion earns an online certificate.

> *Saylor Academy is a nonprofit initiative working since 2008 to offer free and open online courses to all who want to learn. We offer nearly 100 full-length courses at the college and professional levels, each built by subject matter experts. All courses are available to complete — at your pace, on your schedule, and free of cost.*

One needs background in the following:

- Python programming language (CS105: Introduction to Python)
- C++ programming language (CS107: C++ Programming)
- C++ as implemented via Arduino IDE (Arduino Language Reference)
- LoRa itself (Introduction to LoRa Technology, LoRa and LoRaWAN Introduction)
- Arduino hardware (general link, see the appendix for specific devices)

Integrated Development Environments

There are many IDEs one may use. These two are the ones I have found to be most useful. Both tools are well-documented for installation and employment.

- PyCharm Community is an excellent open-access tool for working with Python. (See the bottom of the linked page.) Versions exist for Windows and Linux.

- Arduino IDE is an excellent open-access tool for working with Arduino C++ and associated microcontroller boards. Versions exist for Windows and Linux. (We use the latest 1.x.)

Hardware

Suitable hardware for this project has not been obvious. It is important to look beyond advertising and hype to find reliable hardware and suppliers.

The present thrust relies on the Arduino product line and their MKR WAN 1310 microcontroller board and its embedded LoRa transceiver. Many other options were rejected for various reasons:

* requires rats-nest of jumper wires to implement even simple applications
* poor or disorganized documentation
* lack of versatility
* limited to the infrastructure and components of one company
* unresponsive tech support and inactive user community
* dysfunctional website or out-of-stock on most products
* unable to get basic examples to function correctly
* limited to LoRaWAN, unable to work with LoRaP2P
* shipping increases overall cost by more than 50%
* products are fragile or otherwise of low quality

All things considered, I decided to go with Arduino's product line. Their distribution is global and they have many distributors. They have warehouses in many countries such that devices ordered directly from Arduino ship from the nearest warehouse. Their robust products are relatively expensive, when compared to other options. But they work well, manufacturing is consistent, and they survive many scenarios of physical and electrical application. Documentation is extensive and their user community is active. There are any number of less-expensive clones but my personal experience with clones has not been good. Using original Arduino devices, it is possible to build a grassroots network for around $US170.

It is not necessary to purchase all the hardware at once. It can be purchased as this document proceeds since we build a grassroots LoRaP2P network piece-by-piece and then add options. The Appendix contains a table of all hardware components used in this project, prices (at time of publication), and reliable purchase sources. Item-specific documentation is either on the product page or linked from there.

An Elementary Microcontroller

Arduino's product line is highly modular. It begins with a baseboard and then adds plugin modules. Wire-In points are provided if third-party components are desired. There are several options for baseboards. These vary in price and capability. For this project, we work with the MKR WAN 1310 since it has an embedded LoRa transceiver and so offers the best chance to build a useful network and connect to various sensors.

Assembling Hardware

We start with one MKR WAN 1310, it's antenna, and its USB cable. From the devices' documentation, you will see that the MKR WAN 1310 is somewhat small in physical size. Some people will need a magnifying light so they can use both hands while working with small components. It all depends on your eyesight

Read the product documentation carefully. It explains the MKR WAN 1310 in detail. For this first task, we use the Pins 5v, GND, and A1. We also use the LED_BUILTIN. These are illustrated in Figure 2. The asterixed "pins" accommodate male jumper wires.

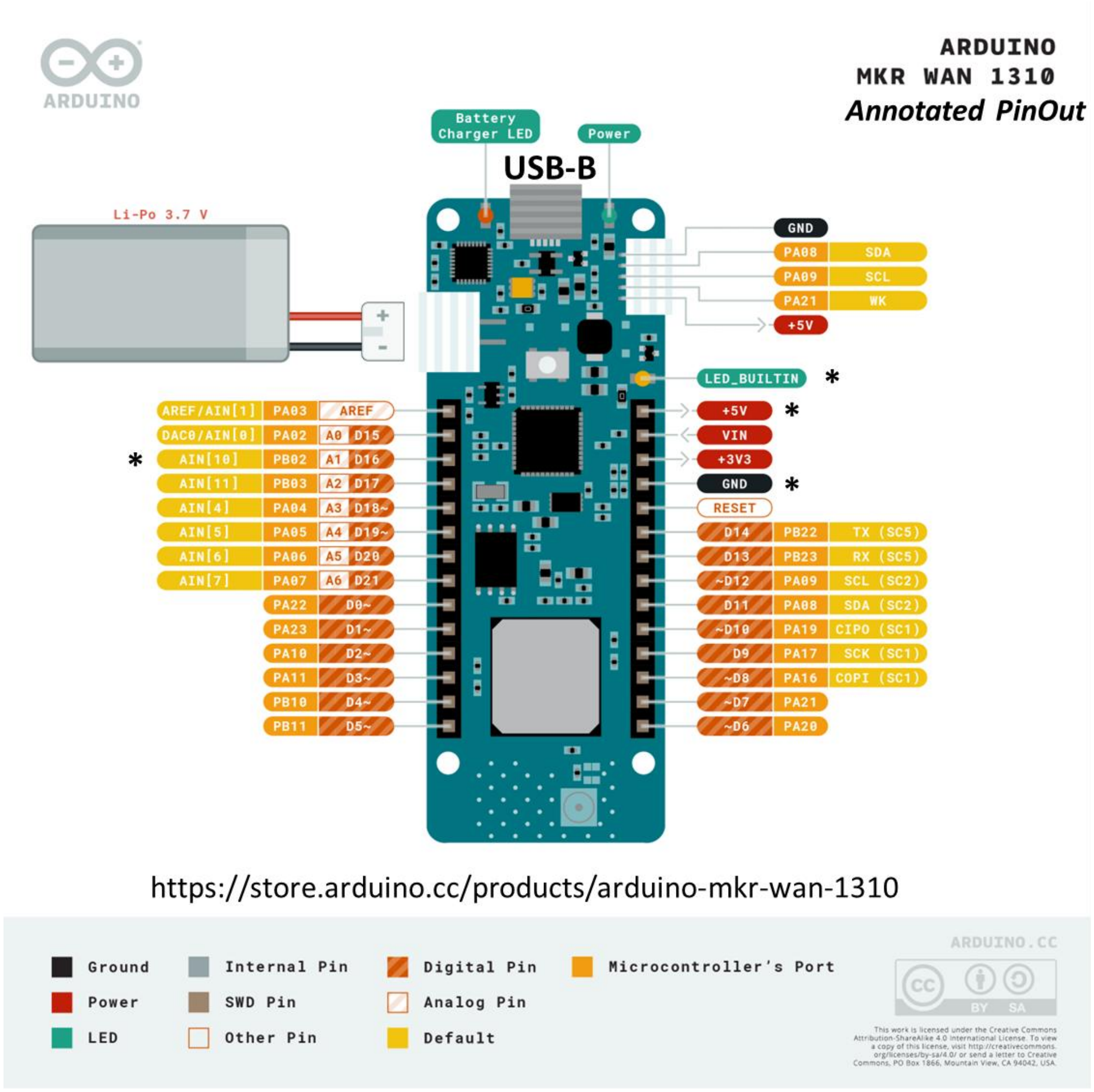

Figure 2. Annotated MKR WAN 1310 pinout diagram.

Installing Software

Once the antenna has been mounted, connect a USB-A/USB-B cable to the device. This cable has to provide power and serial communication. Then plug the other end of the cable into a USB-A port on your computer. You will notice the activated charging and power LEDs near the USB-B port.

Follow these directions to install and configure the Arduino IDE. Notice that we are installing v1.x, not the very latest version, 2.x. The reason is that v2.x is still in beta.

Install the latest version of "Arduino SAMD Boards". Within the IDE, use Tools/Board/Boards Manager. Search on "MKR". Select the library and press Install. Once the library is installed, ensure you have the correct port selected, Tools/Port/Arduino (MKR WAN 1310). Once you reach that point, try a blank sketch and press the "Verify" button. Compilation should succeed if everything has been done correctly.

Now load a relatively simple example, one that just blinks the built-in LED, File/Examples/Basics/Blink. After pressing Verify, the build should complete successfully.

Sketch uses 11964 bytes (4%) of program storage space. Maximum is 262144 bytes.

Global variables use 2988 bytes (9%) of dynamic memory, leaving 29780 bytes for local variables. Maximum is 32768 bytes.

Now press the Load button. The program will recompile and install itself in the device. Once that process completes, you will notice LED-BUILTIN is blinking. This step seems trivial but it is important to verify IDE installation and cable connection. Some USB cables provide only power and do not facilitate data communication. Others provide only data communication. The one specified in the appendix' parts list provides both. We want to be sure everything is working as expected.

As we end this section, we have demonstrated that the device is installed and configured correctly. We have programmed a basic operation and so are ready for something more.

Adding a Simple Sensor

Sensors do nothing more than provide voltage that has to be interpreted according to the sensor type. For our simple sensor, we will just supply a voltage and verify that we are reading it correctly.

The device has an integrated Analog-to-Digital Converter (ADC). ADC's do not give a direct voltage reading. Rather, they give an integer level value according to the specified resolution. See this article by Arrow Electronics for more detail.

The 5v pin is a regulated 5vdc voltage supply (see product documentation for specifics). As the voltage supplied to the device declines below the specified input voltage, the 5v pin output declines and becomes erratic. In that way, we can keep an eye on battery-powered devices power supply.

Run a jumper wire between the 5v and A1 pins. Then load the SimpleSensor code. On the serial monitor, you will see:

```
12:21:07.522 -> =====================================
12:21:07.522 -> Arduino MKR 1310 read-voltage test
12:21:07.522 -> =====================================
12:21:07.522 -> 5.00
12:21:08.507 -> 5.00
12:21:09.527 -> 5.00
12:21:10.508 -> 5.00
12:21:11.533 -> 5.00
12:21:12.513 -> 5.00
12:21:13.544 -> 5.00
. . .
```

That is exactly what we would expect to see. If you want to experiment with the idea, turn off the Serial Monitor and turn on the Serial Plotter. Let the program run for a bit. Then unplug the A1 side of the jumper wire. Follow up by plugging one end of the jumper wire into A1 and the other into GND. Finally, restore the jumper wire to A1 and 5v. You will see a plot similar to Figure 3.

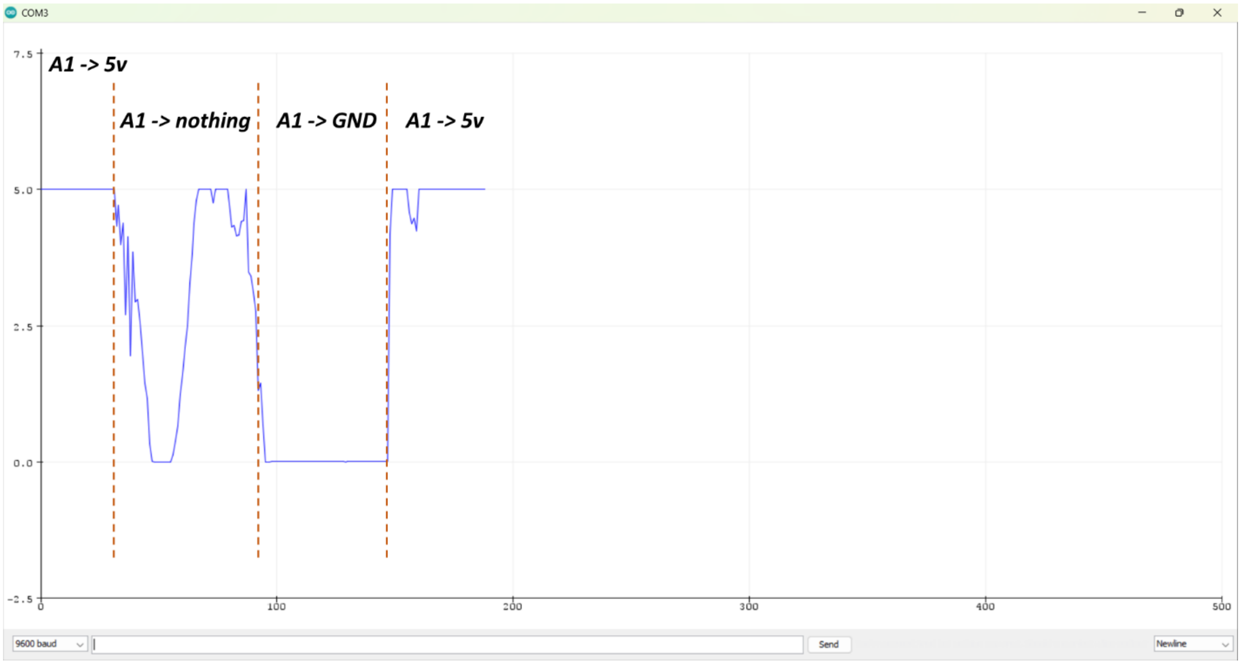

Figure 3. Plot of voltage reading at A1 pin.

A First Cut at LoRaP2P

We are now able to sense voltage inputs. Let's learn how to send and receive that data. We begin by composing and sending simple LoRa messages that contain readings from sensors connected to the device. A second node receives, parses, and displays the contents of those messages. You need two MKR WAN 1310, their antennas, and their associated USB cables.

Which LoRa library to use is an important consideration. This project uses a modified version of the LoRa library maintained by Logsnath Logu and Sandeep Mistry. That library is still in development but is a good foundation. Fixes recommended within that library's pull requests have been incorporated. These resolve various issues encountered as project demands advanced. Use the provided version of the library, the one in the LoRa directory. That entire directory goes into your Arduino libraries directory.

Sender

This first cut is based on an article by Karl Soderby. Merged into Soderby's code is our program for reading sensor data. Connect the 5v and A1 pins. Load BasicSenderNode. You will see the following on the Serial Monitor:

```
14:00:14.636 -> ========================================
14:00:14.636 -> Arduino MKR 1310 basic LoRa sender test
14:00:14.636 -> ========================================
14:00:14.636 ->
14:00:14.636 -> Sending packet: 1
14:00:14.683 -> Packet 1 : Value 4.998779
14:00:15.710 ->
14:00:15.710 -> Sending packet: 2
14:00:15.756 -> Packet 2 : Value 4.998779
14:00:16.781 ->
14:00:16.781 -> Sending packet: 3
14:00:16.829 -> Packet 3 : Value 4.998779
14:00:17.806 ->
14:00:17.806 -> Sending packet: 4
14:00:17.899 -> Packet 4 : Value 4.998779
. . .
```

Receiver

If we take a second MKR WAN 1310 and load BasicReceiverNode, we see that the LoRa packets are being transmitted accurately.

```
16:53:26.545 -> ==========================================
16:53:26.545 -> Arduino MKR 1310 basic LoRa receiver test
16:53:26.545 -> ==========================================
16:53:41.624 -> Received packet 'Packet 1 : Value 4.998779' with RSSI -44
16:53:42.696 -> Received packet 'Packet 2 : Value 4.998779' with RSSI -44
16:53:43.769 -> Received packet 'Packet 3 : Value 4.998779' with RSSI -44
16:53:44.842 -> Received packet 'Packet 4 : Value 4.998779' with RSSI -44
16:53:45.912 -> Received packet 'Packet 5 : Value 4.998779' with RSSI -44
16:53:45.912 -> Received packet 'Packet 5 : Value 4.998779' with RSSI -44
16:53:46.979 -> Received packet 'Packet 6 : Value 4.998779' with RSSI -43
. . .
```

Ensuring Proper Use

Now that we have a basic sender/receiver example, we should now consider proper use. While this author is not able to speak to exact legal matters, all nations have their requirements for proper use of LoRa radio frequencies. If you are communicating with some third-party service, they may have deeper requirements. For example: Which radio frequency to use is set by each nation. This table shows frequency by nation. Notice that some nations do not allow unlicensed radio broadcasts.

Messaging duty cycle is also set by each nation (limitations on how often messages can be sent). For instance, the USA has no duty-cycle limitations but transmissions cannot last longer than 400 milliseconds. Europe has a duty cycle of 1%. Considerable technical detail is given in this document by Semtech. A good parameter calculator is posted by Arjan.

For my own case in the USA, from the calculator and assuming a 13 byte overhead:

Frequency: 915mhz; Spreading Factor: 7; Bandwidth: 125khz

Other factors shown in the linked calculator's table are in regard to The Things Network (a third-party service) and a 1% duty cycle.

We also have to be concerned about Channel Activity Detection (CAD) since we do not want a network node to try broadcasting at the same time as another node. This topic is discussed in Semtech's Introduction to Channel Activity Detection. Their application note speaks to different CAD uses. This author has been unable to confirm that the LoRa baseliine library we are using has a fully-functional CAD capability. However, something we can do is check for an existing signal, per Fujiura Toyonori's modification. His CAD method was integrated into the baseline library.

For this demonstration, the very same code is used for all nodes, except that each node has a unique address. This works because each node can send and receive. Note, however, that the transceiver cannot work in full duplex mode. It cannot send and receive at the same time. Thus, if a message arrives during a transmission, the receiver may not see it.

Load Duplex_Test_NodeA into one device and Duplex_Test_NodeB into another device. Something similar to the following will appear on either serial monitor:

```
13:36:04.065 -> Node is active
13:36:04.529 -> Transceiver is active
13:36:04.529 -> Frequency: 915000000
13:36:04.529 -> Spreading Factor: 7
13:36:04.529 -> Signal Bandwidth: 125000
13:36:04.529 -> ==========================================
13:36:04.529 -> Arduino MKR 1310 LoRa Send/Receive test
13:36:04.529 -> ==========================================
13:36:05.782 -> Received from: 0xff
13:36:05.782 -> Sent to: 0xbb
13:36:05.782 -> Message ID: 13
13:36:05.782 -> Message length: 17
13:36:05.782 -> Message: Hello LoRa World!
13:36:05.782 -> RSSI: -28
13:36:05.782 -> Snr: 9.75
13:36:05.782 ->
13:36:08.477 -> No signal detected. Could send something.
13:36:08.524 -> Sent: Hello LoRa World!
. . .
```

Building a Grassroots Network

Having established basic send/receive, we now build a grassroots network. The devices we are using can all be powered by batteries and encased for outdoor installation. Additional capability can be added as needed. Different antennas can be purchased if required but we begin with the one provided with the Arduino MKR WAN 1310.

Recall from Figure 1 that there are three types of nodes in a flood-messaging network: sensor, relay, basestation. Although any node could be made to send and receive as the ability of the network and its nodes grows, for now we restrict sensor nodes to broadcast only. Relays only receive and rebroadcast. Basestations only receive. Figure 4 offers a view of our grassroots network and its components.

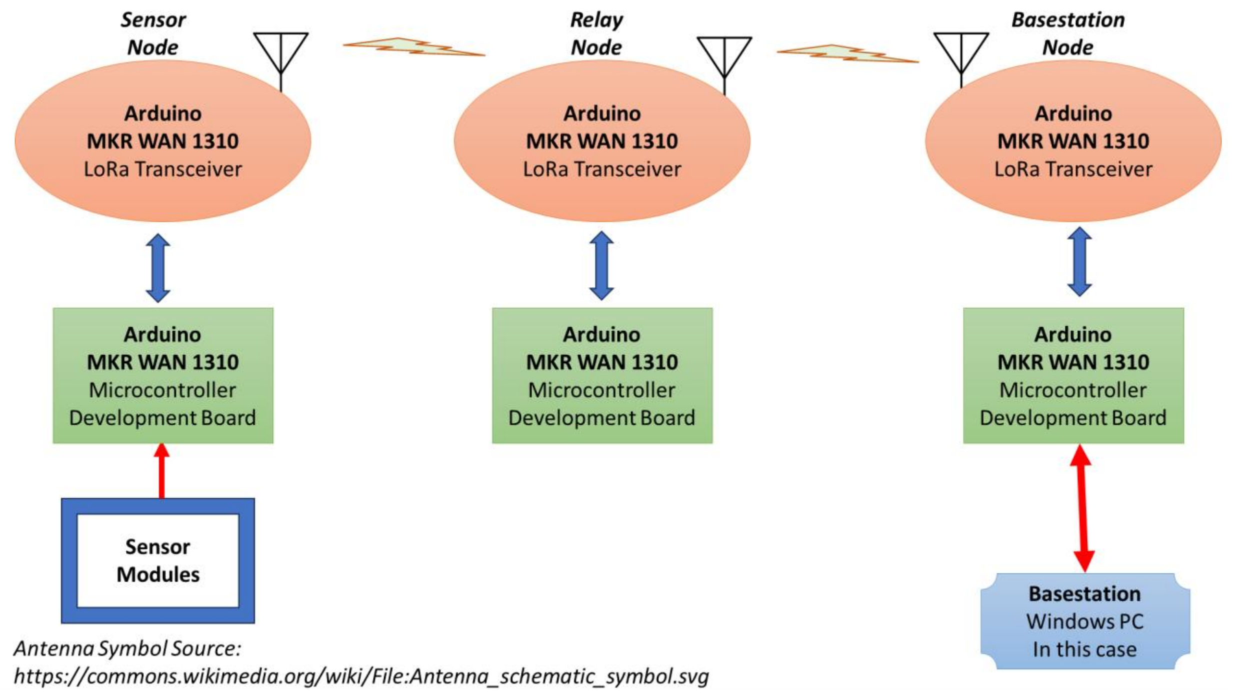
Sensor Node
Relay Node
Basestation Node
Arduino MKR WAN 1310 LoRa Transceiver
Arduino MKR WAN 1310 LoRa Transceiver
Arduino MKR WAN 1310 LoRa Transceiver
Arduino MKR WAN 1310 Microcontroller Development Board
Arduino MKR WAN 1310 Microcontroller Development Board
Arduino MKR WAN 1310 Microcontroller Development Board
Sensor Modules
Basestation Windows PC In this case
Antenna Symbol Source: https://commons.wikimedia.org/wiki/File:Antenna_schematic_symbol.svg

Figure 4. Component relationships in grassroots network.

Recall that we are talking about radio broadcast. There is no reason that any node could not receive a message sent any other node. For instance, the basestation could directly receive a message from a sensor node, as well as that same message rebroadcast by a relay node. It is necessary to account for that possibility without limiting the network to only one basestation. (A "basestation" is any node that receives data and does more than rebroadcast that data.)

Forming Messages for Broadcasting

From a network perspective, it is necessary for messages to contain certain basic information. We apply the following envelope:

Byte	Meaning
0	Message Length (255 max, could be less)
1	System ID
2	Source Node ID
3	Destination Node ID
4	Message ID - High Byte
5	Message ID - Low Byte
6	Message Type
7	Sensor ID
8	Number of Rebroadcasts Remaining
9..255	Message Contents

Notice that we are dealing with bytes, although one can still decide to send messages containing only ascii text. Bytes are unsigned eight-bit integer values in the range 0..255. Text allows for values in the range 0..127. As options are added to the network, we will see why we really do need to think in terms of bytes instead of just text.

We employ a standard byte-level messaging library that is evolving with this project, and which accesses the selected LoRa library. This approach maintains commonality between nodes, except for what appears in the ino file specific to each node type. Take the code in the MessageHandler directory and put that into your Arduino libraries directory.

MessageHandler.cpp contains a DEBUG variable at the very top. If you want to view the library's operational updates on the serial monitor, activate that variable. Otherwise, comment that variable. Each node's ino file has a comparable DEBUG variable so that it is not necessary to see MessageHandler's text as the network operates. During standard operation, both variables should be commented. This is especially true for basestation nodes since those communicate via the serial port with a PC.

MessageHandler's constructor sets LoRa parameters. As distributed, those settings are for this project in the USA. Use the links given earlier to determine appropriate values for your application and your nation. Ultimately, the project may evolve this approach into a parameter-entry file.

Sensor Node

Recall what we did when we added a sensor to our MKR and built a simple sender/receiver. We expand now on that idea by using the supplied standard messaging library. After loading Grassroots_SensorNode, on the serial monitor you should see something similar to:

```
13:45:59.891 -> Node is active
13:46:00.355 -> Transceiver is active
13:46:00.355 -> Frequency: 915000000
13:46:00.355 -> Spreading Factor: 7
13:46:00.355 -> Signal Bandwidth: 125000
```

```
13:46:00.355 -> ==============================================================
13:46:00.355 -> Arduino MKR 1310 Grassroots LoRa flood-messaging sensor node test
13:46:00.355 -> ==============================================================
13:46:00.355 ->
13:46:04.265 -> No signal detected. Could send something.
13:46:04.357 -> Sent message of length 20
13:46:04.357 -> Sent text message: Volts: 5.00 (Text Length: 11)
13:46:04.357 -> Volts: 5.00
13:46:04.357 ->
13:46:07.767 -> No signal detected. Could send something.
13:46:07.813 -> Sent message of length 20
13:46:07.813 -> Sent text message: Volts: 5.00 (Text Length: 11)
13:46:07.813 -> Volts: 5.00
13:46:07.813 ->
13:46:10.270 -> No signal detected. Could send something.
13:46:10.317 -> Sent message of length 20
13:46:10.317 -> Sent text message: Volts: 5.00 (Text Length: 11)
13:46:10.317 -> Volts: 5.00
13:46:10.317 ->
. . .
```

Notice how our earlier work is carried forward. Messages use the same LoRa access methods applied earlier.

As you form text messages, be sure to follow a format that ensures accurate parsing of Sensor Type, Units, and Value. Multiple sensor values can be put that way into the same message. In this case, we are using text. But that is not essential. One could always just send a number as bytes and then specify the sensor type. Then, a table could be used to decode the value sent. There are a lot of messaging options as the need arises.

Basestation Node

There are two components to the basestation, an MKR WAN 1310 and a personal computer running a python program (Linux or Windows, Mac may work too). The MKR component plugs into a USB serial port on the PC. The PC talks to the transceiver, processes the messages, and displays results.

MKR Component

The MKR associated with the basestation receives messages and passes them to the PC via the USB serial port. It ignores messages not meant for the particular basestation to which it is attached.

Load the code for Grassroots_BasestationNode_MKR. When the example sensor node is running and DEBUG is set, you should see something like this on the serial monitor:

```
11:40:12.584 -> Node is active
11:40:13.049 -> Transceiver is active
11:40:13.049 -> Frequency: 915000000
11:40:13.049 -> Spreading Factor: 7
11:40:13.049 -> Signal Bandwidth: 125000
11:40:13.049 -> ============================================================
11:40:13.049 -> Arduino MKR 1310 Grassroots LoRa flood-messaging sensor node test
11:40:13.049 -> ============================================================
11:40:13.049 ->
11:40:16.974 -> No signal detected. Could send something.
11:40:17.021 -> Sent message of length 20
11:40:17.021 -> Sent text message: Volts: 5.00 (Text Length: 11)
11:40:17.021 -> Volts: 5.00
11:40:17.021 ->
11:40:20.449 -> No signal detected. Could send something.
11:40:20.541 -> Sent message of length 20
11:40:20.541 -> Sent text message: Volts: 5.00 (Text Length: 11)
11:40:20.541 -> Volts: 5.00
11:40:20.541 ->
11:40:22.938 -> No signal detected. Could send something.
11:40:22.984 -> Sent message of length 20
11:40:22.984 -> Sent text message: Volts: 5.00 (Text Length: 11)
11:40:22.984 -> Volts: 5.00
. . .
```

If DEBUG is not set, then only the raw passed-forward message will appear. That is what the PC component of the basestation will receive. (DEBUG should not be set if the PC component of the basestation is engaged.)

Personal Computer Component

Your knowledge of Python will come in handy when engaging the PC component of the basestation. That code is under Grassroots_BasestationNode_PC. For this grassroots demonstration, the personal computer simply repeats received messages. Later, we can add graphical displays as desired. We can also interact with external systems as devices are controlled or data is communicated.

DEBUG on the MAK component has to be disabled. (Comment DEBUG in the library and the ino file.) Further, the serial port on the Arduino IDE has to be turned off.

You can run the Python program from the console or the Python IDE. Something similar to the following appears on the PC during this example:

```
Serial ports:
Detected  3  total ports
Name    Device Manufacturer                         Description
COM1    COM1   (Standard port types)    Communications Port (COM1)
COM6    COM6   Arduino AG (www.arduino.cc)  Arduino MKR WAN 1310 (COM6)
COM3    COM3   Arduino AG (www.arduino.cc)  Arduino MKR WAN 1310 (COM3)

Connecting to LoRaP2P microcontroller/transceiver (COM6)
Connected. Awaiting Messages...

1740501250.9958715: Volts: 5.00
1740501251.0602365: Volts: 5.00
1740501254.7671747: Volts: 5.00
1740501258.823454: Volts: 5.00
1740501263.235231: Volts: 5.00
. . .
```

As described earlier, messages can be formed, interpreted, and acted upon according to the needs of the application.

Relay Node

Relays use a third MKR WAN 1310. They rebroadcast messages as they are received. After a given number of rebroadcasts, a message ages out and is no longer rebroadcast.

Load the code for Grassroots_RelayNode. On the receiver's Serial Monitor you should see something similar to:

```
11:13:05.882 -> Node is active
11:13:06.361 -> Transceiver is active
11:13:06.361 -> Frequency: 915000000
11:13:06.361 -> Spreading Factor: 7
11:13:06.361 -> Signal Bandwidth: 125000
11:13:06.361 -> Node Address: 0
11:13:06.361 -> ================================================================
11:13:06.361 -> Arduino MKR 1310 Grassroots LoRa flood-messaging relay node test
11:13:06.361 -> ================================================================
11:13:06.361 ->
11:13:08.646 -> Received from: 0x1
11:13:08.646 -> Sent to: 0x3
11:13:08.646 -> Message length: 20
11:13:08.646 -> RSSI: -35   Snr: 9.75
11:13:08.646 ->
11:13:08.646 -> Rebroadcasting
11:13:08.646 -> No signal detected. Could send something.
11:13:08.743 -> Sent message of length 20
11:13:08.743 ->
. . .
```

Some Thoughts

We see clearly now the basics of Arduino's MKR WAN 1310, how to query sensors, and how to send and receive messages that contain sensor data. Messages should be composed so that they are easily parsed, so that their essential contents can be extracted.

We may be curious about options that make a network more useful. Let's look into some possibilities in the next sections.

< more to come >

Appendix – Hardware Component Sources and Documentation

Here is a table of all hardware components used in this project, prices (at time of publication), and reliable purchase sources. Shipping, import fees, and taxes are not included since those vary according to your source, location, and volume purchased. Some source links are affiliate. Purchases through those links provides a bit of income to this project. Where a component is mentioned in the document, refer to this table for source of purchase. Complete documentation on the component in question is found directly on the product page or is linked from the product page.

Description (See product pages for documentation)	**Price (USD)**	**Total Number Required**	**Source** (2)
A suitable Linux or Windows computer is assumed to exist and to have three available USB ports.			
Basic Essentials - Grassroots Network			
Arduino MKR WAN 1310 Baseboard , with antenna	46.00	3	Arduino Official Store
Micro USB Cable (USB A / USB B)	6.90	3	Arduino Official Store
Male-Male jumper wires	4.80	1	Arduino Official Store
(1) Total Cost	163.50		
Optional Accessories			
Stand-alone magnifying light	40.00	1	Amazon
(3) Soil-Moisture Sensor	45.00	1	Vegetronix
Total if all purchased:	85.00		
For Exploring Image Transmission			
Arduino Uno R3	28.00	1	Amazon
USB 2.0 Cable	7.00	1	Amazon
RaspberryPi single-board computer	129.00	1	Amazon
Ethernet Cable	7.00	1	Amazon
Micro SD Card	8.00	1	Amazon
USB Micro SD Card Adaptor	9.00	1	Amazon
PixyCam + Connector Cable	70.00	1	Amazon
Total if all purchased:	258.00		

(1) Prices are at time of writing. They do not include shipping or taxes.
"Cost" refers to the sum of individual prices times the number required.

(2) "Arduino Official Store" refers to Arduino's global network.
See the bottom-right of the page to select your region.

(3) This is an industrial-grade sensor that happened to be already
on hand. For brief indoor experiments, cheaper sensors will do. For instance:
https://amzn.to/40DeuvC.
One has to study and understand the new sensor, and correctly integrate it
relative to the host device and the soil in question.

Appendix – LoRa Network Types

This project employs LoRa communications technology. LoRa (Long Range) employs unlicensed radio frequencies whose use and configuration are determined country-by-country. An important aspect of this means of radio communication is that LoRa broadcasts are very resistant to electromagnetic interference and physical obstacles (within reasonable limits). The result is improved signal penetration and propagation. This capability is due to the digital modulation employed, which is based on spread-spectrum technology. (Links to LoRa details are given earlier in this document.)

There are two major domains within LoRa application, LoRaWAN (LoRa for wide-area networks) and LoRaP2P (LoRa for direct peer-to-peer networks).

LoRaP2P operates without the need for a gateway. LoRaWAN employs a full network protocol and employs gateways as its means of providing network management and security. Since LoRaP2P does not employ a gateway, it is simpler and less expensive to implement for localized applications. LoRaWAN is meant for large-scale networks with a large number of devices. Like Ethernet UDP LoRaP2P messages have a high chance of delivery but delivery is not guaranteed. Nor is message order ensured.

Within LoRaP2P, there are two approaches to building a network when direct sender/receiver connections cannot be ensured, flood and mesh networks. Both create networks where data can hop between multiple devices through various routes. This enhances reliability and range within challenging environments by allowing data to be sent and received in spite of obstacles so that data has a better chance of reaching its destination.

Flood networks use very simple networking and messaging protocols, whereas mesh networks use more complex network management and message routing approaches. The mesh networks observed by this author to date depend on real-time operating systems and specific computational hardware. On the other hand, the flood network employed in this project has much less dependency on specific hardware and does not use a real-time operating system. Both network types have to be configured for the transceiver to be employed. The implementation of flood networking used in this project is easily transportable from one computational device to another.

Flood and mesh networks operate by connecting multiple devices (nodes) together, where each node can act as a message relay. This allows data to be transmitted across the network by "hopping" from one node to another until it reaches its destination, effectively creating a self-healing network that can reroute data if one node fails. Essentially, every device on the network can send and receive data directly with each other, not just through a central point or gateway, creating a robust and flexible communication system.

Both flood and mesh approaches have the following benefits:

- Ability to connect numerous devices, each acting as a network node that can both send and receive data.
- Transmitted data is forwarded so that it reaches the target node.
- Network congestion and node drop-outs are overcome.
- Multiple connection paths increases the chance of message delivery. If a node fails, data can still be transmitted through alternative routes, ensuring network resilience.
- There is nothing that prevents a network node from connecting to external systems.

This project employs flood networking. A good example of mesh networks is Meshtastic. But this author’s experience is that it is very hardware dependent. Still, international networks have been built using this approach. As this project is focused on local low-cost networks that are flexible regarding hardware, we continue to develop flood networks. The author has built rudimentary networks (one each: sensor, relay, basestation nodes) for under $175US. LoRaWAN begins with a gateway at a cost of around $300US.