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

Peter G. Raeth
Independent Researcher and Educator

Proof of Concept

This document and its associated source code describe a proof of concept only. It has not been verified in an industrial environment. While the author has used good faith efforts to ensure that the information and instructions contained in this work are accurate, the author disclaims all responsibility for errors or omissions, including without limitation responsibility for damages from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code or other technology this work contains or describes is subject to open source licenses or intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

Abstract

This project is a proof-of-concept that builds a peer-to-peer network of LoRa transceivers to support remote sensing. 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 then add options to demonstrate the network's capabilities. Each section of this document takes a specific step toward the final goal 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. Lets dig in and peal 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.

Essential technical support to this project has been provided by:

- Bernd Giesecke of RAKwireless
- Carl Rowan of RAKwireless
- Martino Facchin of the Arduino IDE development team
- Rich Amparo of CalChip

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 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.

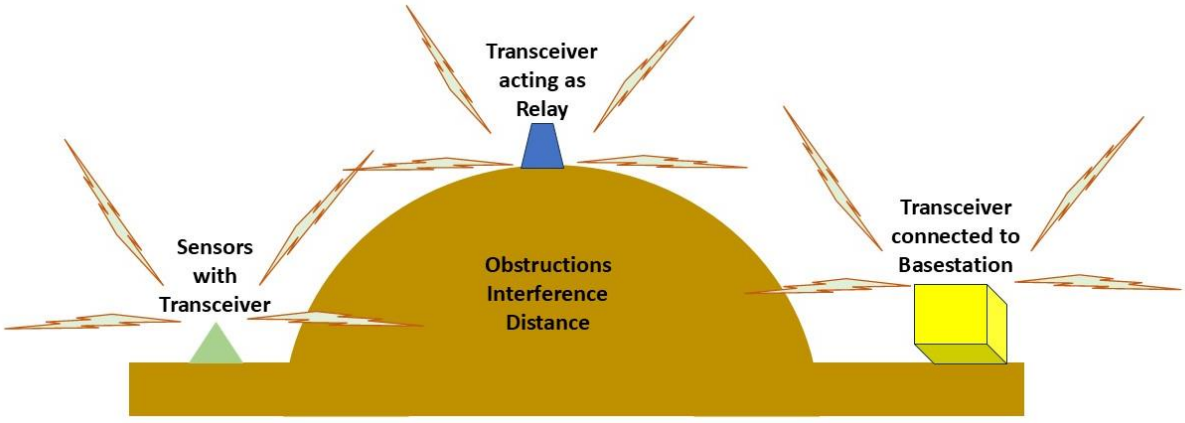


Figure 1. Fundamental idea behind LoRaP2P flood messaging.

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)
- RAKwireless 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.

Hardware

Originally, this project was completed and published while using Arduino Uno R3 microcontroller development boards and LoRaP2P transceiver plugins. However, at the beginning of 2024 the transceiver's manufacturer declared that it would no longer produce that particular transceiver. They did replace it with a different plugin. After ordering one unit and working with it, I was unable to get it to respond properly even to simple AT queries with no transmit/receive involved. Persistence with the device did not help and I decided that it was not appropriate for this project.

From there I began looking for something new. Many 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 of low quality

All things considered, I decided to go with RAKwireless' product line.

It is true Rakwireless is based in China but they have distributors in many nations. Their components can communicate with Arduino microcontroller development boards and RPi headless computers. They can be programmed using the Arduino IDE. Here is a good introductory video on RAKwireless' products. More detail is given in this excellent introductory book.

It is not necessary to purchase all the hardware at once. It can be purchased as this document proceeds since we build a basic 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 Transceiver Node

RAKwireless' 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. I like to start with one having several options. This allows ideas to grow without buying additional baseboards. For our purposes, we will want to add plugins to the board as we move forward.

Assembling Hardware

We start with one RAK19001 baseboard, coupled with a RAK11310 computational core. From the devices' documentation, you will see that RAKwireless products are quite 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. Screws are provided in the package to firmly mount components to the baseboard. As you can imagine, these screws are even smaller. They require an equally-small Phillips-head screwdriver. You may well have one small enough in your toolkit. If not, a Manual Precision Screwdriver (**SKU:**920430) works well. Figure 2 and Figure 3 show my own setup.

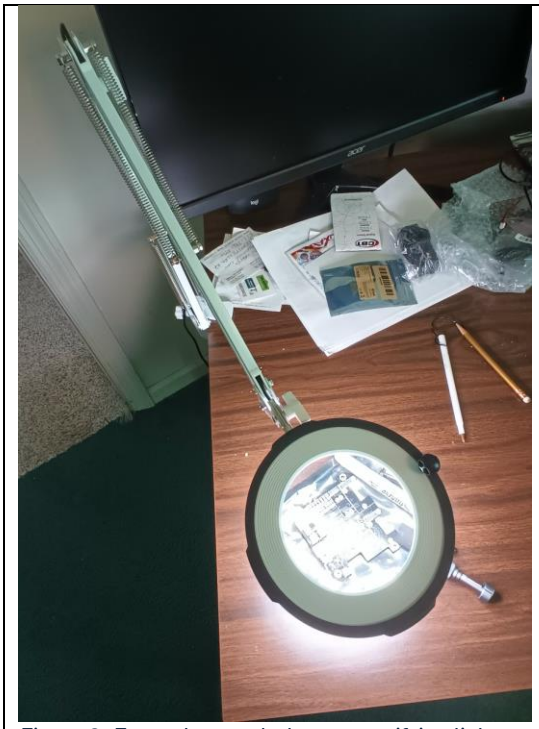


Figure 2: Example stand-alone magnifying light

Figure 3: Component size comparison

Read the documentation for the RAK19001. It explains that component in detail and explains how to mount modules to it. Follow that guidance to mount the RAK11310. Be sure to mount the antenna before applying power. As you look closely at the RAK19001, near the center, you will see a connector marked "CPU SLOT". That is where the RAK11310 gets mounted. Figure 4 Illustrates.

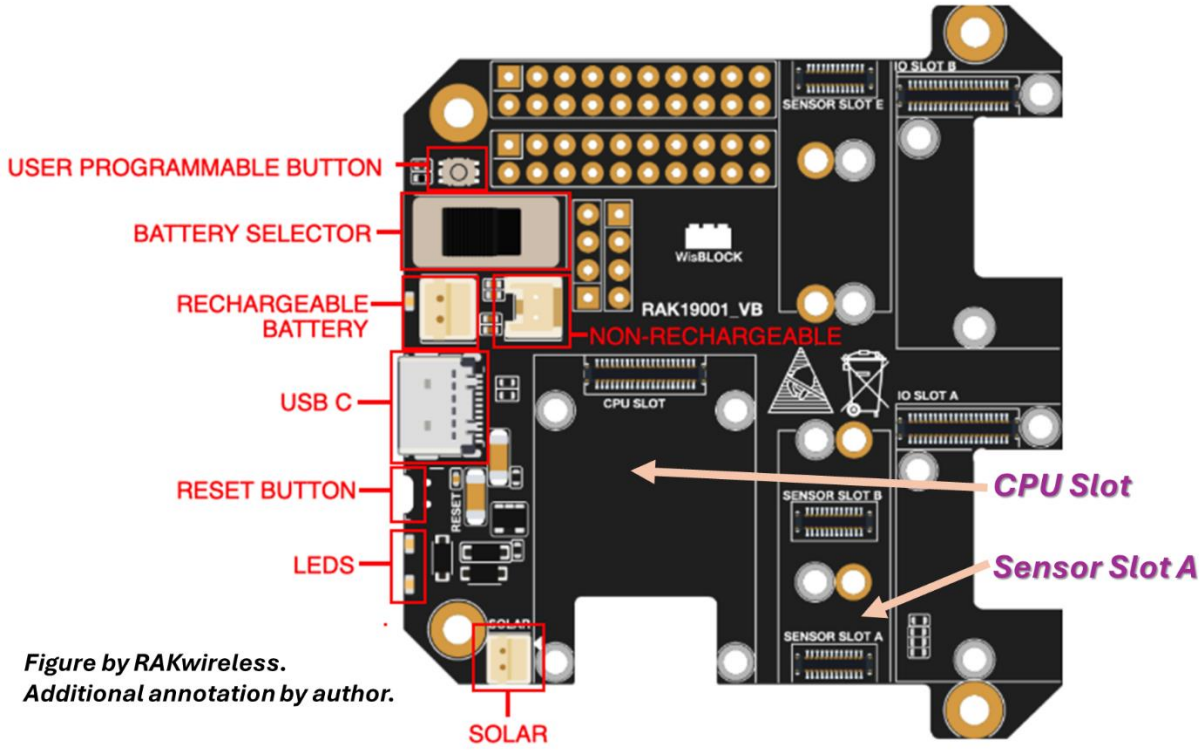


Figure 4. RAK19001 baseboard top

DO NOT force-fit any modules. For myself, I found that the best approach is to line up the screw holes and then lightly provide firm-and-even downward pressure. If you have to force it, you are doing something wrong. Be sure to mount the antenna to the provided connector on the upper-right of the RAK11310 before applying power. The provided antenna is good for a start. RAKwireless offers better ones if your application requires them.

Some people are very handy with tools. I know how to use a screwdriver without impaling myself but that is about the extent of it. So, forgive me if I add some thoughts. There are two different sizes of screws provided with the RAK19001. The smallest of these are used to finish mounting the RAK11310. The Manual Precision Screwdriver is designed to work with all RAKwireless screws. Fit the head of the screw onto the head of the screwdriver (which is magnetized). Then use the screwdriver to aim the screw into the hole. If your hand is unsteady at close distances, use one hand to hold the top of the screwdriver and a finger from the other hand on the screwdriver's blade to guide the screw into the hole. Turn the screw carefully into the hole so that the screw is finger-fast only. Do this for all for all four holes.

Figure 5 shows what the result should look like.

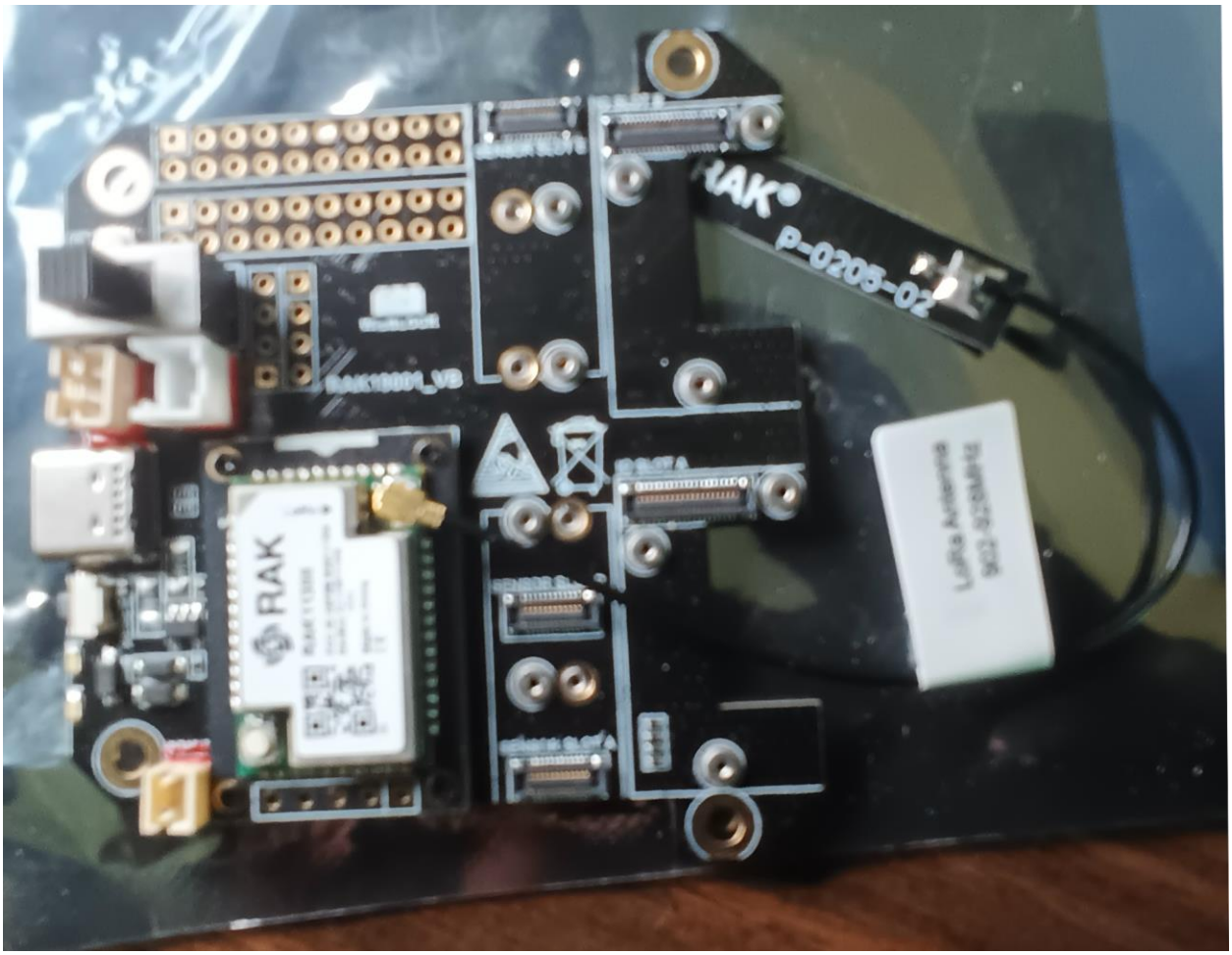


Figure 5: RAK11310 fully mounted on RAK19001

Installing Software

Once fully mounted, connect the provided USB-C cable to the RAK19001. This cable provides power and serial communication. The port is on the left side, near the middle. Then plug the other end of the cable into a standard USB port on your computer. You will notice a red blinking LED on the right side of the baseboard, near the middle.

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 RAKwireless is a bit out of sync with v2.x such that blanks in the PC's user name causes compilation failure. For those who want to dig into this matter and try a local solution, see this thread.

Install the latest version of "RAKwireless RP Boards" (RAK11310 is an update of RAK11300). Ensure you have the correct port selected, Tools/Port/RAK11300. 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 gives a reading on battery voltage. In the IDE, go to File/Examples/RAK Wiseblock examples/RAK11300/Power/ RAK11300_Battery_Level_Detect. That sketch will come up. Notice the link on Line 14. Click on that link. The IDE's library manager will come up. EiMOS_U8X8 will be on top. Install that library and all its dependencies. Close the library manager. After pressing Verify, the build should complete successfully

> *Sketch uses 17916 bytes (0%) of program storage space. Maximum is 16777216 bytes.*
>
> *Global variables use 46056 bytes (17%) of dynamic memory, leaving 224280 bytes for local variables. Maximum is 270336 bytes.*

Now press the Load button. The program will recompile and install itself in the RAK11310. Now use Tools/Serial Monitor. You will see that the program periodically polls and reports battery voltage. Notice that battery voltage is reported in millivolts, despite the mistaken nomenclature.

> *The ADC value is: 2765*
> *The battery voltage is: 4113.98 V*
> *LIPO = 4113.98 mV (87%) Battery 222*
> . . .

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

Adding a Sensor

As one considers the future of this project, one realizes that the baseboard and its plugins may ultimately be enclosed in some sort of casing to protect it from weather and other environmental conditions. All electronics have minimum and maximum operating temperatures and moisture tolerance. So, we may want to mount one temperature/humidity sensor (RAK1901) onto the baseboard, and then report its values. (If we want sensors like this for weather-station applications, the sensors have to be mounted outside the baseboard's enclosure. Note that, to deliver value, weather stations have to be properly installed and positioned. But that is a topic for another conversation.)

The process of adding the RAK1901 temperature/humidity sensor is rather straightforward. Do follow the component's documentation and apply the same cautions as we did during the installation of the computational core. Be sure to unplug the baseboard from your computer before mounting or dismounting any components.

Immediately to the right of the computational core we have already mounted, you will see Sensor Slot A and Sensor Slot B. Further up on the baseboard, you will see Sensor Slot E. Sensor Slot C and Sensor Slot D are on the underside of the baseboard. We are going to use Sensor Slot A. Notice that the sensor slots are smaller than the I/O slots further to the right as shown in Figure 4.

Line up the connection slot on the sensor with Sensor Slot A on the baseboard. Ensure its mounting hole lines up with the associated left screw-hole on the baseboard. Gently and evenly press the sensor's connection slot into the baseboard's connection. Put in a screw to keep the sensor firmly mounted to the baseboard. Here again, use the smallest screw.

Plug the baseboard back into your computer. Bring up your IDE and load File/Examples/ RACK Wiseblock examples/RAK11300/Sensors/ RAK1901_Temperature_Humidity_SHTC3. Notice Line 11 in the code, use the supplied link to load the required library and its dependencies. The required library will be at the top of the list. To check that all is ready, press the IDE's verify button. You will see the following compiler messages:

> *C:\Users\Peter Raeth\AppData\Local\Arduino15\packages\rakwireless\hardware\mbed_rp2040\0.0.6\libraries\RAK_examples\examples\RAK11300\Sensors\RAK1901_Temperature_Humidity_SHTC3\RAK1901_Temperature_Humidity_SHTC3.ino: In function 'void shtc3_read_data()':*
>
> *C:\Users\Peter Raeth\AppData\Local\Arduino15\packages\rakwireless\hardware\mbed_rp2040\0.0.6\libraries\RAK_examples\examples\RAK11300\Sensors\RAK1901_Temperature_Humidity_SHTC3\RAK1901_Temperature_Humidity_SHTC3.ino:36:8: warning: variable 'Temperature' set but not used [-Wunused-but-set-variable]*
>
> *float Temperature = 0;*
>
> *C:\Users\Peter Raeth\AppData\Local\Arduino15\packages\rakwireless\hardware\mbed_rp2040\0.0.6\libraries\RAK_examples\examples\RAK11300\Sensors\RAK1901_Temperature_Humidity_SHTC3\RAK1901_Temperature_Humidity_SHTC3.ino:37:8: warning: variable 'Humidity' set but not used [-Wunused-but-set-variable]*
>
> *float Humidity = 0;*
>
> *Sketch uses 16372 bytes (0%) of program storage space. Maximum is 16777216 bytes.*
>
> *Global variables use 44644 bytes (16%) of dynamic memory, leaving 225692 bytes for local variables. Maximum is 270336 bytes.*

Compilation is complete. The warnings are of no concern. Many programmers automatically initialize all variables, whether those initial values are ever used or not. Appropriate values are assigned later in the code. This practice helps prevent certain hard-to-find errors later in the software development cycle that are caused by random memory content.

Once loaded, the following should appear on the serial monitor:

> *shtc3 init*
> *Beginning sensor. Result = Nominal*
> *ID Passed Checksum. Device ID: 0b100010000111*
> *RH = 39.66% (checksum: pass), T = 28.89 deg C (checksum: pass)*
> *RH = 39.66% (checksum: pass), T = 28.89 deg C (checksum: pass)*
> *RH = 39.67% (checksum: pass), T = 28.87 deg C (checksum: pass)*
> *. . .*

The result of adding this step is a node that can report on its ambient situation regarding battery voltage, temperature, and relative humidity. We may well ask what good is all that if we have to keep going out to the node to check on it. In the next section we start to address that issue.

A Prelude to Broadcasting Messages

As we move forward, we will want to compose and send LoRa messages that contain readings from sensors connected to the baseboard. We are now able to sense humidity, temperature, and battery voltage. We have code that demonstrates all three. Now we want to consolidate and simplify that code so that reporting on each only occurs once and all other reportings are removed. That other data is good for demonstration and diagnostics but not strictly necessary for what we ultimately want to accomplish. (If you want the ability to turn diagnostics on or off, use #ifdef/#endif pairs.)

This code consolidation is best performed in a modular fashion where each group of code appears in its own subroutine. The loop() function calls those subroutines in their proper sequence to produce the desired result. The code block humidity_temperature_battery contains that consolidation. When loaded into the computational core, the serial monitor shows:

%RH = 37.70
dC = 27.64
Battery mV = 4337.16
%RH = 37.65
dC = 27.63
Battery mV = 4301.45
%RH = 37.67
dC = 27.61
Battery mV = 4222.59
. . .

Let's start to do something with that data.

A First Cut at LoRaP2P

Now that we have learned how to assemble and test the hardware as basic sensor nodes, let's experiment with sending messages between nodes. You will need one RAK19007/RAK11310 pair, along with the unit previously constructed and tested. Put the new pair together now, just as we did before. Test with the same RAKwireless battery example. You should get the same results as before.

We now have two physical nodes. Let's program them for sending messages from one to another. In this first trial, one node acts as sender and the other as receiver. There is no reason a node cannot act both as a sender and as a receiver but that will come later.

Plug both nodes into your PC, each node into a different USB port. Keep each node physically separated and not too close together. Certainly, do not have the antennas touching each other or anything metallic. Within the IDE, go to Tools/Ports. You will see your two nodes listed there. Each will be on a different port. Think of the port with the sensor module as sender and the port without the sensor module as receiver.

Sender Example

Use Tools/Port and Tools/Board to select the sender node. Select File/Examples/ RACK Wiseblock examples/RAK11300/Communications/LoRa/LoRaP2P/ LoRaP2P_Tx. Notice Line 13. Use that link to install the SX126x-Arduino library and all its dependencies. Press Verify. If successful so far, you will see:

D:\Program_Files\Arduino-IDE-1.x\Projects\libraries\SX126x-Arduino\src\boards\mcu\rak11300\SimpleTimer.cpp: In member function 'void SimpleTimer::run()':

D:\Program_Files\Arduino-IDE-1.x\Projects\libraries\SX126x-Arduino\src\boards\mcu\rak11300\SimpleTimer.cpp:67:40: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]

if (current_millis - prev_millis[i] >= delays[i])

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^~~~~~~~~~~~

D:\Program_Files\Arduino-IDE-1.x\Projects\libraries\SX126x-Arduino\src\boards\mcu\rak11300\SimpleTimer.cpp: In member function 'bool SimpleTimer::check()':

D:\Program_Files\Arduino-IDE-1.x\Projects\libraries\SX126x-Arduino\src\boards\mcu\rak11300\SimpleTimer.cpp:139:42: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]

if ((current_millis - prev_millis[i]) >= delays[i])

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^~~~~~~~~~~~

D:\Program_Files\Arduino-IDE-1.x\Projects\libraries\SX126x-Arduino\src\mac\LoRaMacHelper.cpp: In function 'void McpsConfirm(McpsConfirm_t)':*

D:\Program_Files\Arduino-IDE-1.x\Projects\libraries\SX126x-Arduino\src\mac\LoRaMacHelper.cpp:387:46: warning: suggest braces around empty body in an 'if' statement [-Wempty-body]

LOG_LIB("LMH", "Timeout TX + RX finished");

^

D:\Program_Files\Arduino-IDE-1.x\Projects\libraries\SX126x-Arduino\src\mac\LoRaMacHelper.cpp:403:93: warning: suggest braces around empty body in an 'if' statement [-Wempty-body]

LOG_LIB("LMH", "Timeout Conf TX finished %s", mcpsConfirm->AckReceived ? "SUCC" : "FAIL");

^

Sketch uses 14120 bytes (0%) of program storage space. Maximum is 16777216 bytes.

Global variables use 45644 bytes (16%) of dynamic memory, leaving 224692 bytes for local variables. Maximum is 270336 bytes.

Despite the warnings, the code does compile successfully. To say the warnings do not matter would be false. Such warnings are not acceptable, even though the code will function correctly under these specific circumstances. The warnings come from library code so I have not tried to repair them. When teaching courses, I do not accept code containing warnings or errors. Warnings tend to obscure what else might be wrong. Using compiler options to eliminate warnings does not solve that problem.

Look now at Line 30. That is your operating frequency. It is different for each country. Be sure you replace what is there with the correct frequency for your country. Being in the USA, I used 915mhz. Thus, for me, Line 30 needs to read "#define RF_FREQUENCY 915000000 // Hz".

Press Upload. On the Serial Monitor, you will see:

```
====================================
LoRap2p Tx Test
====================================
OnTxDone
OnTxDone
OnTxDone
OnTxDone
OnTxDone
OnTxDone
. . .
```

Receiver Example

Use Tools/Port and Tools/Board to select the receiver node. Select File/Examples/ RACK Wiseblock examples/RAK11300/Communications/LoRa/LoRaP2P/ LoRaP2P_Rx. Change Line 30 to the frequency appropriate to your country. Press Verify. The code should compile successfully, albeit with the same warnings as before.

Press Upload. If no messages are received from the sender node, you will see this display on the receiver's Serial Monitor:

```
====================================
LoRaP2P Rx Test
====================================
Starting Radio.Rx
OnRxTimeout
OnRxTimeout
OnRxTimeout
OnRxTimeout
OnRxTimeout
OnRxTimeout
. . .
```

That is what you will see if you start the receiver before the sender. So, always start the sender first. If you start the sender first, the receiver will display:

```
====================================
LoRaP2P Rx Test
====================================
13:10:14.350 -> Starting Radio.Rx
13:10:16.761 -> OnRxDone
13:10:16.761 -> RssiValue=-22 dBm, SnrValue=13
13:10:16.761 -> 48 65 6C 6C 6F
13:10:19.752 -> OnRxTimeout
13:10:22.768 -> OnRxTimeout
13:10:25.772 -> OnRxTimeout
13:10:28.767 -> OnRxTimeout
. . .
```

We have established basic LoRa P2P operation. Let's expand on that.

Goal-Oriented LoRaP2P Send/Receive

There are two issues with the earlier send/receive example: 1) The sender must be started before the receiver. 2) Only one message is received. We need to improve on that.

Let's modify the transmitter's program so that messages are sent every second. Load LoRaP2P_TX_modified onto your sender node. You will see the same display as before on the serial monitor.

We also make modifications to the receiver so that it delivers an understandable message. LoRaP2P_RX_modified rewrites LoRaP2P_RX so that it repeats the message as text characters, instead of as just numbers in base_16 (hex). (Later, we will examine those numbers as byte values, 0..255. This turns out to be very facilitating when interpreting complex messages.) We also ensure that the order of starting sender or receiver is not a factor. Load LoRaP2P_RX_modified onto your receiver node. The receiver will time-out as long as there are no incoming messages. But, when there are messages, the receiver node will display:

```
====================================
LoRaP2P Rx Test
====================================
Starting Radio.Rx
OnRxDone
RssiValue=-29 dBm, SnrValue=13
48 65 6C 6C 6F  : H e l l o
OnRxDone
RssiValue=-29 dBm, SnrValue=13
48 65 6C 6C 6F  : H e l l o
. . .
```

In the next section, we push this idea even further as we build a grassroots flood-messaging network.

Building a Grassroots Network

We describe a grassroots network. The devices involved 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 RAK11310.

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 6 offers a view of our grassroots network and its components.

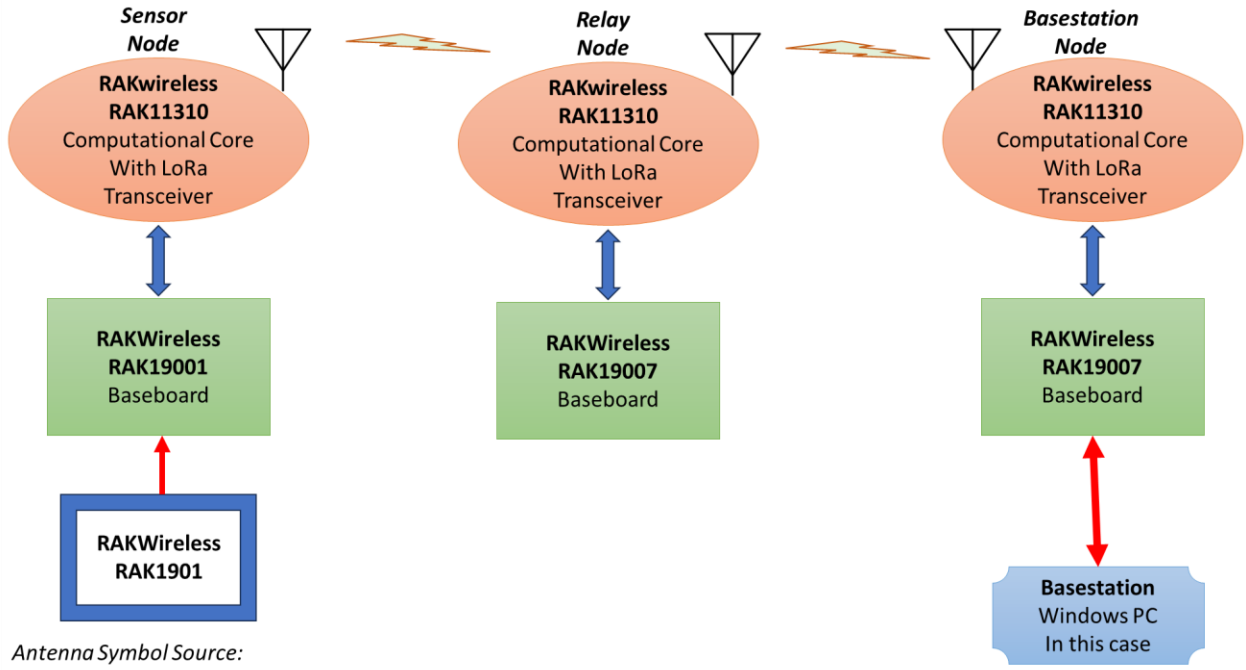


Figure 6. Component relationships in grassroots network.

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 sent by a relay node. It is necessary to account for that possibility without limiting the network to only one basestation.

Forming Messages for Broadcasting

From a network perspective, it is necessary for messages to contain certain basic information.

Byte	Meaning
0	Message vector's ending Index (0 .. 255 infers 256 max bytes)
1	System ID
2	Source Node ID
3	Destination Node ID
4	Source Message ID - High Byte
5	Source Message ID - Low Byte
6	Message Type
7	Source Sensor ID
8	Number of Rebroadcasts Remaining
9..255	Message Contents

Notice that we are dealing in bytes, although our messages, for now, will only contain text. Bytes are unsigned eight-bit integer values in the range 0..255. Text allows for values in the range 0..127 only. 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. For now, text will do. To that end, we use a MessageConstruction class that packs text data into 256-byte messages. After a message is constructed, it is broadcast. We do have to take care that the text of our message does not become so long that the overall message length exceeds its maximum. Other nodes use that same message format to unpack and interpret messages.

Sensor Node

Recall what we did when we added sensors to our baseboard. We assumed that the board would be inside an enclosure and wanted to keep an eye on the container's environmental conditions. We also wanted to monitor the battery. Let's now broadcast that data using a sender node. This amounts to merging code we have already written (LoRaP2P_TX_modified and humidity_temperature_battery). The new code can be found in FloodMessaging_sensor. Upload that code to the RAK19001/RAK11310/RAK1901 component we built and demonstrated earlier. On the sender's Serial Monitor you should see something similar to:

```
5:39:10.661 -> ================================================================
15:39:10.661 -> Initializing LoRaP2P Tx for sending sensor data via text messages
15:39:10.661 -> ================================================================
15:39:10.661 -> Serial port initialized
15:39:11.208 -> shtc3 initialized
15:39:11.208 -> Beginning sensor. Result = Nominal
15:39:11.208 -> ID Passed Checksum. Device ID: 0b100010000111
15:39:11.208 -> The ADC value is: 2654
15:39:11.250 -> The battery voltage is: 3948.82 mV
15:39:11.250 -> ================================================================
15:39:11.250 -> Initialization completed
15:39:11.250 -> ================================================================
15:39:11.250 ->
15:39:11.250 -> %RH = 28.00
15:39:11.250 -> dC = 22.64
15:39:11.250 -> The ADC value is: 2849
15:39:11.250 -> The battery voltage is: 4238.96 mV
15:39:11.250 -> Message: Batt_V   4.24, Temp_C   22.64, Humid_%   28.00 |
15:39:11.250 -> Header: 57 111  3  2  0  1  3  0  5
15:39:11.250 -> Sending message
15:39:11.337 -> OnTxDone
15:39:13.161 ->
15:39:13.161 -> %RH = 27.98
15:39:13.204 -> dC = 22.59
15:39:13.204 -> The ADC value is: 2850
15:39:13.204 -> The battery voltage is: 4240.45 mV
15:39:13.204 -> Message: Batt_V   4.24, Temp_C   22.59, Humid_%   27.98 |
15:39:13.204 -> Header: 57 111  3  2  0  2  3  0  5
15:39:13.204 -> Sending message
15:39:13.297 -> OnTxDone
. . .
```

As the serial monitor displays its debugging output, messages are being broadcast into the network as byte vectors. These messages look like:

Header: 57 111 3 2 0 2 3 0 5

- 57 is the message's last cell index
- System ID is 111
- Source Node ID is 3
- Destination Node ID is 2
- Message ID is 2
- Message Type is 3 since this is a text message
- Source Sensor ID is 0 since message is text interpreted for several sensors
- There are five rebroadcasts remaining for this message

The header is followed by the text message itself. For instance:

Batt_V   4.24, Temp_C   22.59, Humid_%   27.98 |

Observe the format of this particular text message. It can be parsed and so is interpretable. In this case, we are using commas to separate the major components and a <sensor name><space><reading> sub-format. A '|' is a unique character that appears at the end. This indicates a complete message. If that character does not appear as the last character of a text message then the message has been truncated as too long. Clarity in message format is important if messages are to be useful. Keep that in mind as you compose messages of any kind.

Relay Node

Relays rebroadcast messages from sensor nodes as they are received. After a given number of rebroadcasts, a message ages out and is no longer rebroadcast. Older messages are not rebroadcast. Later improvements reject older messages from the same node regarding the same sensor.

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

```
11:44:01.921 -> ======================================
11:44:01.921 -> LoRaP2P Relay Ready
11:44:01.921 -> ======================================
11:44:03.608 ->
11:44:03.608 -> OnRxDone
11:44:03.608 -> Size = 58 bytes, RssiValue = -34 dBm, SnrValue = 12
11:44:03.608 -> Batt_V   4.15, Temp_C   23.42, Humid_%   40.25 |
11:44:03.639 ->       Current/Previous Message ID: 399 / 0
11:44:03.639 -> Received true. Message rebroadcast.
11:44:05.554 ->
11:44:05.554 -> OnRxDone
11:44:05.554 -> Size = 58 bytes, RssiValue = -34 dBm, SnrValue = 13
11:44:05.554 -> Batt_V   4.24, Temp_C   23.42, Humid_%   40.20 |
11:44:05.554 ->       Current/Previous Message ID: 400 / 399
11:44:05.554 -> Received true. Message rebroadcast.
11:44:07.497 ->
. . .
```

It is possible that the receiver will time out between messages. That can happen if the receiver is checking for messages at a faster rate than the sender produces messages. This is of no concern. In the long run, it is not always necessary to report timeout events.

Basestation Node

There are two components to the basestation, a LoRaP2P transceiver/microcontroller and a personal computer (Linux or Windows, MAC may work too). The transceiver plugs into a serial port on the PC. The PC talks to the transceiver, processes the messages, and displays the results.

LoRaP2P Microcontroller/Transceiver

The microcontroller associated with the transceiver acts rather like a relay node but does not rebroadcast messages. Instead, it sends them to the PC via the serial port.

Load the code for FloodMessaging_Basestation_RAKwireless. Given our running example, you should see something like this on the serial monitor when the PC is not connected and DEBUG is set:

```
12:15:34.651 -> ====================================
12:15:34.651 -> LoRaP2P Basestation Ready
12:15:34.651 -> ====================================
12:15:35.753 ->
12:15:35.753 -> OnRxDone
12:15:35.753 -> Size = 58 bytes, RssiValue = -34 dBm, SnrValue = 13
12:15:35.753 -> Batt_V   4.28, Temp_C  23.62, Humid_%  39.69 |
12:15:35.753 -> Received true. Message passed forward.
12:15:35.753 -> 9oBatt_V   4.28, Temp_C  23.62, Humid_%  39.69 |
12:15:37.655 -> OnRxDone
12:15:37.655 -> Size = 58 bytes, RssiValue = -34 dBm, SnrValue = 13
12:15:37.655 -> Batt_V   4.16, Temp_C  23.61, Humid_%  39.72 |
12:15:37.701 -> Received true. Message passed forward.
12:15:37.701 -> 9oBatt_V   4.16, Temp_C  23.61, Humid_%  39.72 |
. . .
```

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

Personal Computer

Your knowledge of Python will come in handy when engaging the PC component of the basestation. That code is under FloodMessaging_Basestation_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.

Something similar to the following appears on the PC's console during this example:

```
Serial Ports:
Detected  4  total ports
COM8  :        COM8 :        Microsoft
COM1  :        COM1 :        (Standard port types)
COM6  :        COM6 :        Microsoft
COM4  :        COM4 :        Microsoft

Connecting to LoRaP2P microcontroller/transceiver (COM4)
Connected. Awaiting Messages...

Batt_V   4.16, Temp_C  23.40, Humid_%  39.39 |
Batt_V   4.19, Temp_C  23.41, Humid_%  39.38 |
Batt_V   4.20, Temp_C  23.41, Humid_%  39.37 |
Batt_V   4.24, Temp_C  23.40, Humid_%  39.41 |
. . .
```

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

Additional Thoughts

We see clearly now how to mount modules to a baseboard, how to query standard 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. In this case, we use commas to separate data items and use an <element><spaces><number> data format.

As we have concentrated on RAKwireless sensor modules, we may be curious about options that make a network more useful. Let's look into some possibilities in the next sections.

Data Security

Here we briefly address the issue of data security. We take the opportunity to implement the AES encryption/decryption standard.

Many people want to have secure communication. Although that can never be perfect, we have the Advanced Encryption Standard (AES) published by the US National Institute of Standards and Technology (NIST). AES is the same symmetric block cipher the US government uses to protect classified information. You can read more about AES in this posting by TechTarget: https://www.techtarget.com/searchsecurity/definition/Advanced-Encryption-Standard.

There are many implementations of AES posted on the internet. While I do not argue about their correctness, there was only one posting I could find that demonstrated how to use the underlying AES implementation to encrypt and decrypt variable-length messages. This library was written by Sergey Bel. You can download the library from his repository. We use this library to provide an example of how to implement AES encryption/decryption for LoRaP2P message broadcasts. There are many options for use that are discussed in the repository.

Cryptology and Cyber-Security are vast fields into which this project does not delve. You may find this open-access book on cryptology useful, https://cryptobook.nakov.com. You can read a discussion about the issues at hand in the well-written master's thesis by Priyansha Gupta.

Demonstrating Message Encryption/Decryption

Our overall project uses microcontroller/transceiver units. One unit has a sensor module. After working with the example in the selected AES library, AES_Baseline, the example's code was added to the code for message interchange. Each message to be transmitted is first passed through the encryption algorithm. On the receiver node, each message is decrypted after reception.

Running the Software

humidity_temperature_battery_send is expanded in humidity_temperature_battery_send_cypher. humidity_temperature_battery_receive is expanded in humidity_temperature_battery_receive_cypher. In this way, we expand our previous software to

include encryption/decryption. Loading and running this software shows that the receiver replicates the message sent by the transmitter.

If you merge encryption/decryption into the grassroots network, notice that applying AES may well extend the length of the message. That is normal to AES but has to be accounted for in the message to be transmitted by sensor nodes and interpreted by relay and basestation nodes. I have not yet decided on which way to go with this but it is not likely to be difficult.

Possible Extension

The employed AES implementation uses a static key but the key need not be static as long as both sender and receiver use the same key. Varying keys over time may enhance security. A message could be used to trigger a change in key without sending the key itself. As this project is not a study in communication security, the author has not attempted to extend what is demonstrated.

Third-Party Sensors

"There are ways and there are ways." That is good to remember. No approach is best for all applications. RAKwireless enables many ways to add third-party sensors to its modules. In this section we add a third-party sensor to our RAK19001/RAK11310.

Soil-Moisture Sensor

All sensors only deliver a voltage. (In the case of RAKwireless, that voltage must be limited to 0 .. 5.) Voltages have to be interpreted in terms of the type of sensor and the application to which it is applied. In our case, we convert voltages delivered by soil-moisture sensors to Volumetric Water content (VWC). "VWC is the ratio of the volume of water to the unit volume of soil. Volumetric Water Content can be expressed as a ratio, percentage, or depth of water per depth of soil (assuming a unit surface area), such as inches of water per foot of soil." (Ref: Oklahoma State University Agricultural Extension)

Capacitive soil-moisture sensors are far more reliable and accurate than resistives. Resistives deteriorate rapidly. In synergy, their accuracy and reliability does as well. For brief explorations, they may well be fine but I avoid them because of my other work. Voltage readings for all types of soil-moisture sensors have to be calibrated for the soil in which they will be used. Many companies that sell these sensors provide a table that shows calibration according to soil type. They have already carried out a laboratory-grade calibration procedure. You can find many procedures on the internet. A general approach is described by Daniel Fernandez in his video. A similar process is described by Vegetronix. These processes result in a VWC vs Voltage curve, an example of which is shown for Miracle Grow Potting Soil. The data used to generate such plots can be used to generate an equation that relates voltage output by the sensor to VWC. An implementation of code that employs a piecewise-linear equation for Arduino is offered by Michael Ewald.

In the next sub-sections, we add a soil-moisture sensor to our RAK19001/RAK11310. Different soil-moisture sensors can certainly be used but their required voltages and calibrations have to be studied and implemented. Those will not be exactly the same as described here. As we go forward, we mount and test the basic arrangement. Then we calibrate for known input voltages. Finally, we add a Vegetronix VH400-2M capacitive soil-moisture sensor.

Setting Up

Begin with a RAK5811. Follow the documentation carefully, especially the figure that shows how to orient the module with the baseboard. The approach we took to mount the computational core offers good technique. RAK5811 provides attachment points for the soil-moisture sensor's wires. No soldering is required. Instead, one uses a small straight-edge screwdriver to depress and release crimp points.

You will find clear guidance on mounting the RAK5811 linked from its product page. Follow that guidance carefully to ensure good physical connection. The code we will use originated with this thread and was produced by Bernd Giesecke.

The first effort simply ran Giesecke'a code and injected various input voltages for analog inputs A0 and A1. Recall that these input voltages have to be limited to 0..5vdc. *Figure 7* illustrates.

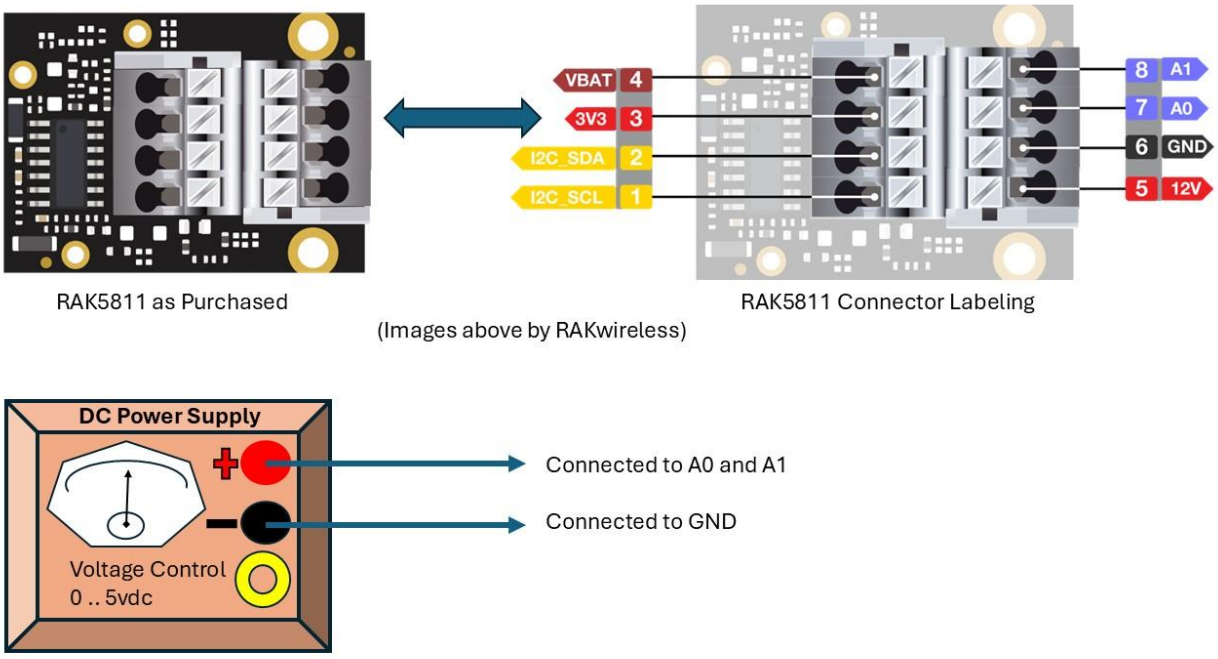


Figure 7. Voltage input to RAK5811.

The result of this trial matched the original exposition as the voltage increased from 0 to 5vdc.

10:04:43.375 -> A0 = 0.013431
10:04:43.375 -> A1 = 0.013431
10:05:36.260 -> A0 = 0.017460
10:05:36.260 -> A1 = 0.017460
10:05:49.680 -> A0 = 0.201465
10:05:49.680 -> A1 = 0.201465
10:06:13.732 -> A1 = 0.554701
10:06:14.701 -> A0 = 0.554701
1:35:23.3470 -> A0 = 0.831380
11:35:23.347 -> A1 = 0.832723
11:35:24.261 -> A0 = 0.905250
11:35:24.295 -> A1 = 0.906593
11:35:30.042 -> A0 = 1.117460
11:35:30.042 -> A1 = 1.118803
11:35:29.067 -> A0 = 1.147009
11:35:29.067 -> A1 = 1.147009
. . .

Producing a Calibration

If we want the displayed voltage to match that of the input, we need a calibration equation. To get that, we input a known voltage and record the value delivered by the code's variable mcu_ain_voltage. If we use a spreadsheet to do that, we arrive at *Figure 8*. (The spreadsheet applied a linear equation to produce the trendline.)

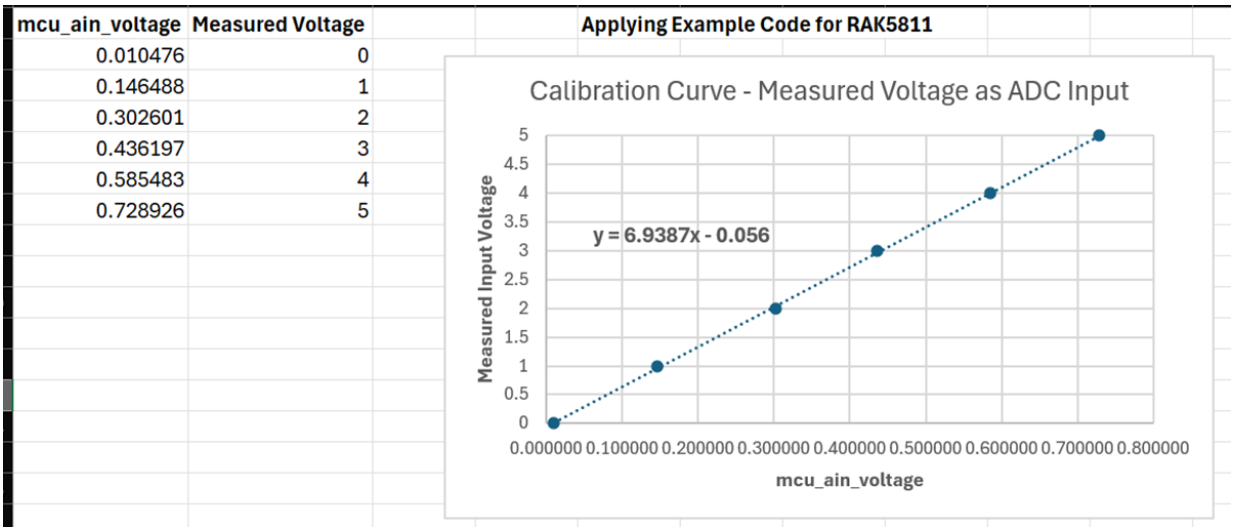


Figure 8. Trendline developed from measured inputs.

Replacing the calibration equation that produces the value for the variable voltage_sensor, we get the following when again inputting measured voltages:

```
13:09:51.442 -> mcu_ain_voltage = 0.726055  A0 =  5.0
13:10:22.213 -> mcu_ain_voltage = 0.600769  A1 =  4.1
13:11:30.448 -> mcu_ain_voltage = 0.451358  A0 =  3.1
13:12:04.048 -> mcu_ain_voltage = 0.320254  A1 =  2.2
13:12:08.853 -> mcu_ain_voltage = 0.158553  A0 =  1.0
13:15:08.557 -> mcu_ain_voltage = 0.010703  A0 =  0.0
```

Why are the results for A0 and A1 different from what the table in *Figure 8* shows?

- The equation generating the trendline does not pass exactly through all points.
- The voltage generator is not laboratory grade. Its output voltages may not be constant.
- The voltage generator uses analog dials whose readings vary by eyeball position.
- RAK5811 may not generate the very same reading if the input is held constant.

Thus, one should not depend on more than one broad decimal point in the reading. So, for instance, do not make decisions based on 3.1 vs. 3.2. There may be some value in stretching to 3.1 vs. 3.5.

Adding a Soil-Moisture Sensor

Now we add a Vegetronix VH400-2M capacitive soil-moisture sensor. This is an industrial-grade sensor. If you are just experimenting, something less expensive can be used.

Figure 9 shows how to wire these two sensors into the RAK5811.

Vegetronix VH400-2M Soil Moisture Sensor Wiring Table

Bare	Ground: RAK5811 GND
Red	POWER: RAK5811 12v (3.5V to 20 VDC)
Black	OUT: RAK5811 A0 or A1 (0 to 3V related to soil's moisture content.)

Figure 9. Vegetronix wiring table.

Soil-moisture sensors need an additional calibration equation to convert voltage_sensor to volumetric water content (VWC) for the particular soil in question. There are simple and complex approaches to arriving at this equation. (Vegetronix documentation discusses that topic.)
For one particular soil, Vegetronix offers the volts-vs-VWC chart show in *Figure 10*.

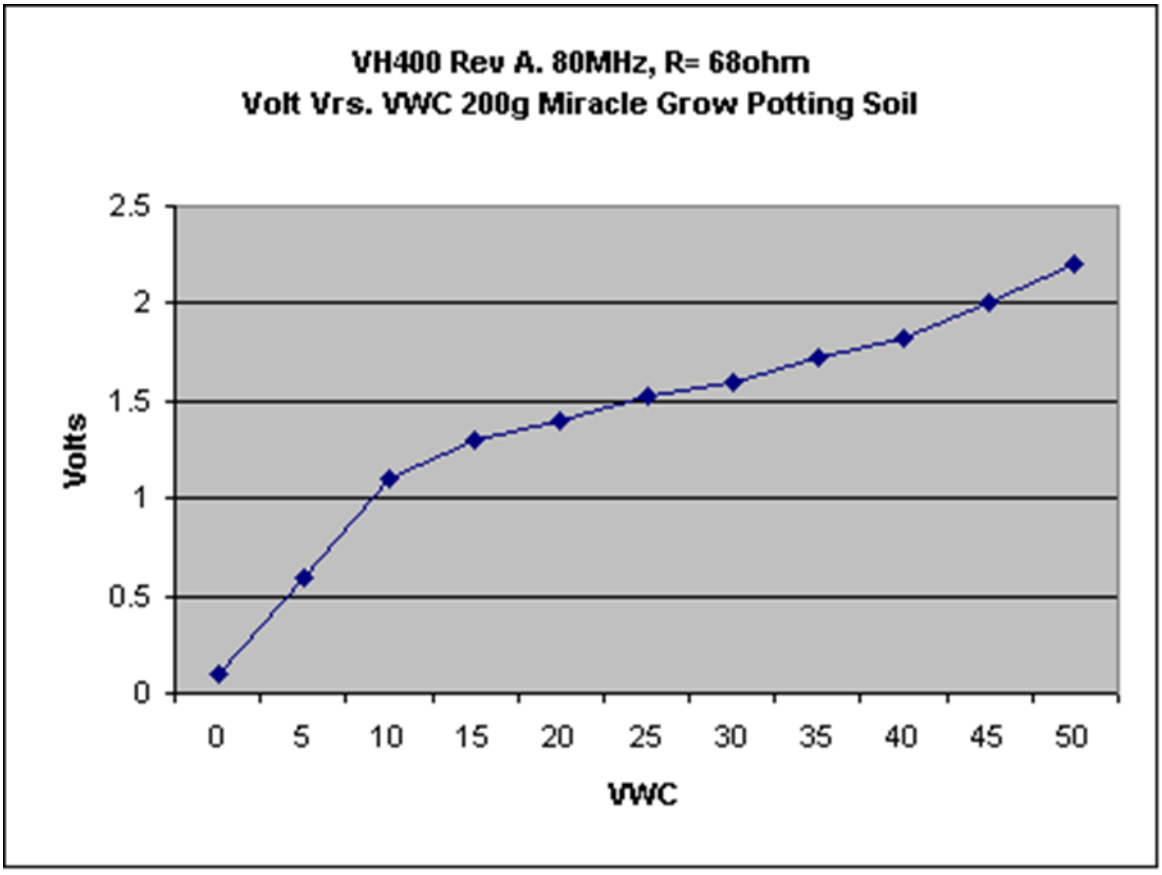


Figure 10. Vegetronix plot of volts-vs-VWC for a given soil.

They do not offer their data table but if we eyeball the chart and reverse the axes, we can arrive at an equation such as $\mathbf{y = -17.741x^4 + 77.926x^3 - 95.824x^2 + 47.161x - 3.6598}$, a fourth-order polynomial for converting x (voltage) to y (VWC). Or, we could use a set of piecewise linear equations. For all sensors, voltage_sensor calibration is needed to convert the sensor's output voltage to units meaningful to the application.

Image Sources

By far, most of the discussion I have seen on images within LoRa networks claims that LoRa is not designed for sending imagery. Others go so far as to claim it cannot be done. Yet, there are applications where single images are needed at long intervals. This section shows that images can indeed be sent through a LoRa network, and one way to do it. Fundamentally, LoRa is designed to send messages having 256 bytes or less. There is no reason a sensor's data has to fit within one message.

As it turns out, connecting a camera directly to RAKwireless modules is not easy, if it is possible at all. A solution is to host the camera on another device. For example, the Arduino Uno R3 readily hosts a Pixy2 camera via a simple cable. Uno can communicate serially with RAK via their serial ports. The same is generally true with connecting non-RAK devices to RAK units when those devices do not have accommodating plugs but do have serial ports. Yes, using serial ports is not necessary if one uses circuitry to connect for serial communication but that approach is not always feasible. To enable data-flow observation as part of a development/debugging process, we begin by establishing a programmable USB HUB. To make all this work, the following hardware is required:

- Arduino Uno R3 and its associated but separate USB cable.
  (It is possible that Arduino Uno R4 and Arduino Mega will also work. Have not tried that.)

- RaspberryPI v4 and its associated but separate Ethernet cable. Creating a programmable USB HUB does not necessarily require such a meaty SBC. It all depends on what you want the hub to do with incoming data before the data is passed on. It also depends on how many ports you need, and of which version. (This particular board has two v2 and two v3 USB ports.) For a specific implementation, choose the hardware, OS, and software most suited to that purpose. The device purchased can be used as part of numerous evolving proofs-of-concept. Note too that Raspberry PI has plugin modules for various applications.

- RAKwireless RAK19001 baseboard with RAK11310 computational core. Follow the documentation links for putting these together. Essentially, RAK11310 plugs into RAK19001 to make a compact unit. But care must be taken in making that happen.

- PixyCam v2 is the camera used in this proof-of-concept. It plugs easily into the Arduino Uno R3 SPI connector.

Setting up the Hardware

Here is what the project did to get the hardware ready to go. The RaspberryPi-v4 is programmed in Python-v3.10. That software runs just as well on a PC so testing is easy. RAKwireless and Arduino devices are programmed in the Arduino implementation of C++.

RaspberryPi-v4

Here are the steps followed get this device ready to use. This device operates just like any headless Linux device. There are also similarities with remotely-accessed Linux computers.

1. **Configured for headless operation (no desktop or keyboard).**
   Follow this documentation. To burn the OS, I employed a Windows PC and a USB SD Card adaptor. A 16gb micro-SD card proved sufficient. Used the imager's advanced-settings options to avoid having to modify the OS' files. Notice that there are many OS one can install. These depend on the purpose at hand. Selected OS/Other/Lite-64 since this device will take 64 bits and a desktop or keyboard was not needed. Using the imager's advanced-settings options greatly simplifies the configuration process.
   - Left hostname at raspberrypi.local to maintain synergy with the Pi device's documentation.
   - Enabled SSH and set password authentication plus username/password.
   - Did not set wireless LAN since that was not needed.

- Set locale options per my geographical location.
- Turned off all persistent settings.

Burn took a few minutes. Process went smoothly. Dismounted the SD card from the PC and mounted it on the device. (Card faces outward from its socket, downward relative to the board, you see the face of the card as it is being inserted.)

2. **Connected device to ethernet network and plugged in the power source.**
   Which socket to use on the board is clearly marked. It does not appear possible to plug the power supply into the wrong socket. Plug the power supply's adaptor firmly into the socket. If you do it too lightly, it will not connect. However, as in all such things, if you have to force it, you are doing something wrong. The power supply's switch is not marked for on/off. For those with good eyes in good lighting, there appears to be a nub on the switch that marks the ON position. If you press the switch down on that nub, in the direction of the device, the power is on. Once the device is booted, a red LED will show. The green blinking LED will have stopped.

3. **Checked for presence on ethernet network.**
   Router's device-list showed two devices without names. That was the correct number. Brought up the PC's command console. Entered the command "ping raspberrypi". Ping confirmed the device's presence. Used Putty and Port 22 to connect to raspberrypi. Login console appeared. Entered username/password. Linux prompt appeared. "python --version" showed that v3.9.2 is installed. That is sufficient for our purposes.

4. **Once the device's operation was verified, the next step installs the device into its case.**
   Turned off the device and then disconnected the power supply and ethernet cable. Removed the SD Card from the device. Otherwise, it is likely the card will break. Followed the instructions that come with the case. All parts and the screwdriver are provided in the kit. Note: Pins on the device are not numbered but Pin-10 is marked. Look carefully at the diagram in the instructions. Count pins carefully when connecting the fan. After Step 2 in the instructions, notice the two heat sinks provided in the kit. There is no instruction regarding the heat sinks but have a look at this video. This video shows how to install the provided heat sinks. These appear meant for the processor and memory chips. Heat sinks for the USB and ethernet controllers are not included in the kit. Before Step 3 in the instructions, installed the two provided heat sinks. After Step 5 in the instructions, the two halves of the case were not tight together. There was still looseness. Tried using a better screwdriver but found that more tightness risked stripping the screws' bodies and heads. Once Step 5 is completed, notice the back of the case, the opposite side from the fan. There are two wall mounts. Be careful of how long the mounting screws are, and the size of the heads. It is possible to damage the board or short it out. The case is not designed to accommodate pin-mounted plugins. Notice too the four round indentations on the back of the case. These are for the feet provided in the same bag with the screws. A scissor serves to cut these to size for mounting. The material surrounding the round feet is designed to peal away. Installing the feet and laying the case down on the feet improves air circulation and thus cooling for both wall and table mounting. Finally, notice the carry-case that comes with the kit. This is a nice travel and storage case.

5. **Verified installation and operation.**
   Reinserted the SD Card containing the 64-bit Lite operating system. This is a Debian Linux OS extended for Raspberry Pi. Reattached the power supply and ethernet cable. Turned the power switch to ON. The fan came on. The red and ethernet LEDs came on. Used Putty to connect to the device. Logged in. "python --version" showed that v3.9.2 is already installed. This completed implementation and verification of the device. A concern is the loose case but perhaps a plastic tie strap would fix that.

6. **Moved files from PC to RaspberryPi**
   It is necessary to move Python software files from the PC to the RaspberryPi. An excellent open-access tool for that is FileZilla. After installation, go to File/SiteManager. Use SFTP as the protocol and raspberrypi as the host. Port remains blank. Connect and login. File transfers are drag-and-drop.

Arduino Uno R3

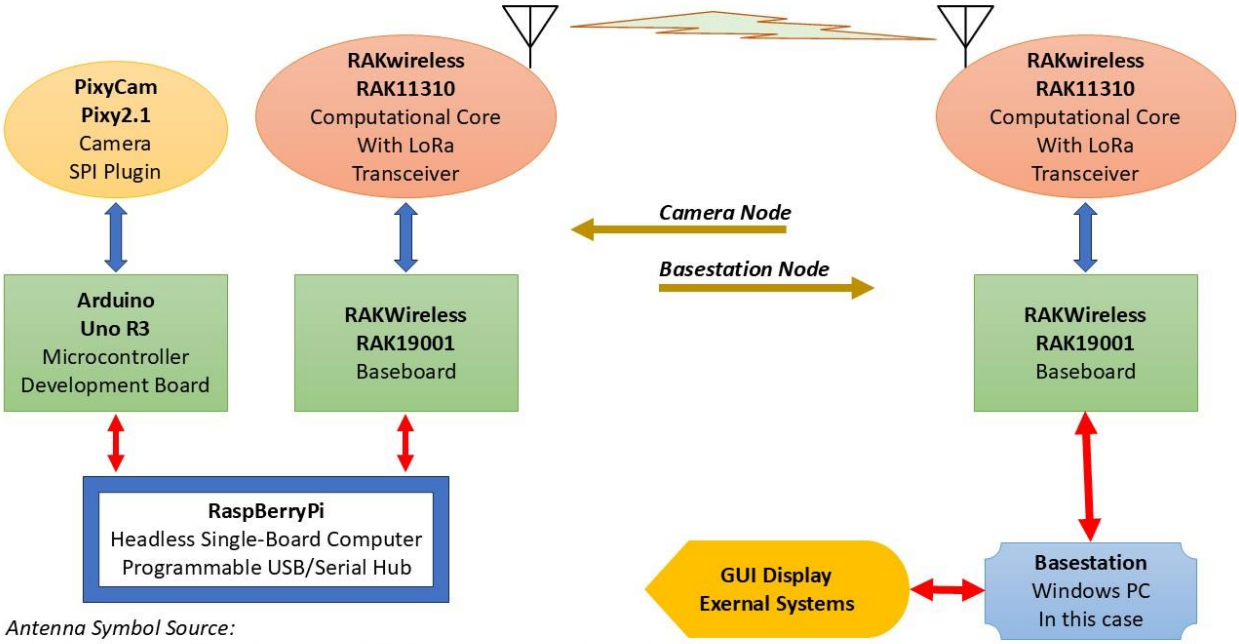
To learn how to set up this particular microcontroller development board, see this getting-started guidance. You will also want these details as you dig deeper. The board comes with various examples.

Pixy2.1 Camera

Very good documentation is provided. Follow it carefully. Depower the Uno and plug the Pixy into that board's SPI connector. Lots of good examples as you go along.

Top-Level Architecture

Figure 11 gives a block diagram of how the various hardware components operate together. From the figure, there are two nodes, the camera node and the basestation node. The camera communicates with its transceiver via a programmable USB HUB. Camera images are sent to the basestation via LoRaP2P. The basestation decodes messages and acts accordingly.



Antenna Symbol Source:
https://commons.wikimedia.org/wiki/File:Antenna_schematic_symbol.svg

Figure 11. Project's basic architecture.

We start with the idea of connecting two disparate devices via their USB serial ports, facilitated by a programmable USB HUB. This will engage the camera node only.

Bridge Two Devices - Text Messages

The USB HUB can be programmed to fulfill whatever data processing/transfer needs are required. For the application at hand, each connected device begins by exchanging a single-byte handshake with each device with which it wishes to communicate. For our first case, text messages are exchanged between devices. RAK requests the value of the next pixel. Arduino responds, cycling through available pixels. The USB HUB enables and reports the message interchange. Since the hub code is written in Python, it can also be run as-is on a PC. This facilitates the process of programming and debugging the code on the two different microcontrollers.

Look under the directory BridgeTwoDevices-TextMessages. main.py is the code for the USB HUB. Arduino-Server contains the code for the Arduino Uno R3 with Pixy2 camera. RAK-Client contains the code for the RAK19001/RAK11310. Loading and running the code on the appropriate device, the USB HUB displays messages between the two devices. These are similar to:

Serial ports:
Detected 3 total ports
COM1 : COM1 : (Standard port types)
COM6 : COM6 : Microsoft
COM7 : COM7 : Arduino LLC (www.arduino.cc)

Configuring serial ports...
Connected to PORT_A.
Port Name: COM6

Port Baudrate: 9600
Connected to PORT_B.
Port Name: COM7
Port Baudrate: 9600

Looking for PORT_A handshake: Success. Sending handshake to PORT_B.
Looking for PORT_B handshake: Success. Sending handshake to PORT_A.
Handshakes exchanged

From PORT_A: GetNextPixelValue
From PORT_B: RGB(1, 1) = (255, 219, 117)
From PORT_A: GetNextPixelValue
From PORT_B: RGB(2, 2) = (255, 215, 122)
From PORT_A: GetNextPixelValue
From PORT_B: RGB(3, 3) = (255, 222, 125)
From PORT_A: GetNextPixelValue
From PORT_B: RGB(4, 4) = (255, 222, 119)
From PORT_A: GetNextPixelValue
From PORT_B: RGB(5, 5) = (255, 224, 130)
. . .

This exercise demonstrates the basic idea of how a programmable USB_HUB can facilitate device inter-communication when each device has a USB serial port. Now lets go further.

Transmit Camera Images

Text messages are good for a start but text does not admit to the entire range of ascii values, only 0..127. Plus, the number of bytes (characters) required to send numeric data increases quite a bit when messages are limited to text. So, it is better if we move on to binary messages, those where each byte can take on the full ascii range of 0..255. That is what we will do as we send entire images via LoRaP2P. As before, we use a programmable USB HUB to connect a camera to a RAK transceiver. This second effort expands the camera node to binary data. The camera node sends an entire image, row by row.

Periodically, the camera sends an image to the transceiver. The image is segmented into LoRa-sized messages and sent to the hub, which sends the messages to the LoRa transceiver. The transceiver creates LoRa packets and broadcasts them. Packets are received by the Basestation's LoRa transceiver, which extracts the message contents from the packet. The hosting computational unit passes the message contents to the Basestation, which reconstructs the image. As the image components are received, the Basestation posts them to the image display grid.

Running the Modules' Software

Review Figure 11 again. We speak here about both parts of the figure, the camera and basestation nodes. The basestation and the USB HUB both use software written in Python-v3.10. Arduino and RAKwireless devices use the Arduino implementation of C++. Look under the directory TransmitCameraImages. Camera-USB-HUB is the code for the USB HUB. Camera-Arduino contains the code for the Arduino Uno R3 and the Pixy2 camera. Camera-RAK contains the code for the

RAK19001/RAK11310. Loading and running the code on the appropriate device, the USB HUB displays messages between the two devices. These are similar to:

Serial ports:
Detected 3 total ports
COM1 : COM1 : (Standard port types)
COM6 : COM6 : Microsoft
COM7 : COM7 : Arduino LLC (www.arduino.cc)

Configuring serial ports...
Connected to PORT_A.
Port Name: COM6
Port Baudrate: 9600
Connected to PORT_B
Port Name: COM7
Port Baudrate: 9600

Looking for PORT_A handshake: Success. Sending handshake to PORT_B.
Looking for PORT_B handshake: Success. Sending handshake to PORT_A.
Handshakes exchanged

1730754883.5562887 > PORT_B: 250
1730754884.7680705 > PORT_B: 250
1730754885.9777308 > PORT_B: 250
1730754887.1895087 > PORT_B: 250
1730754888.4015765 > PORT_B: 250
1730754889.6096697 > PORT_B: 250
1730754890.8221722 > PORT_B: 250
1730754892.0305297 > PORT_B: 250
1730754893.2429543 > PORT_B: 250
. . .

The first value in the last set of lines is a decimal version of time. The second value is the source port. The third value is the length of message. Notice that PORT_A gives no feedback. In this case, PORT_A hosted the RAK transceiver. PORT_B hosted the Arduino/Pixy camera. Recall that the camera sends a picture periodically. It requires no request to do so.

Having broadcast the camera's image, we want now to receive the broadcast and display the picture. That is the job of the basestation and its transceiver. Basestation-RAK is the software running on the RAK device. BaseStation-PC is the software running on the PC. Load those two programs into their appropriate device and start them. When the basestation starts you will see "Press START to begin" on the console. This is a reference to the GUI shown in Figure 12. Whenever you are ready, press "Click to Start".

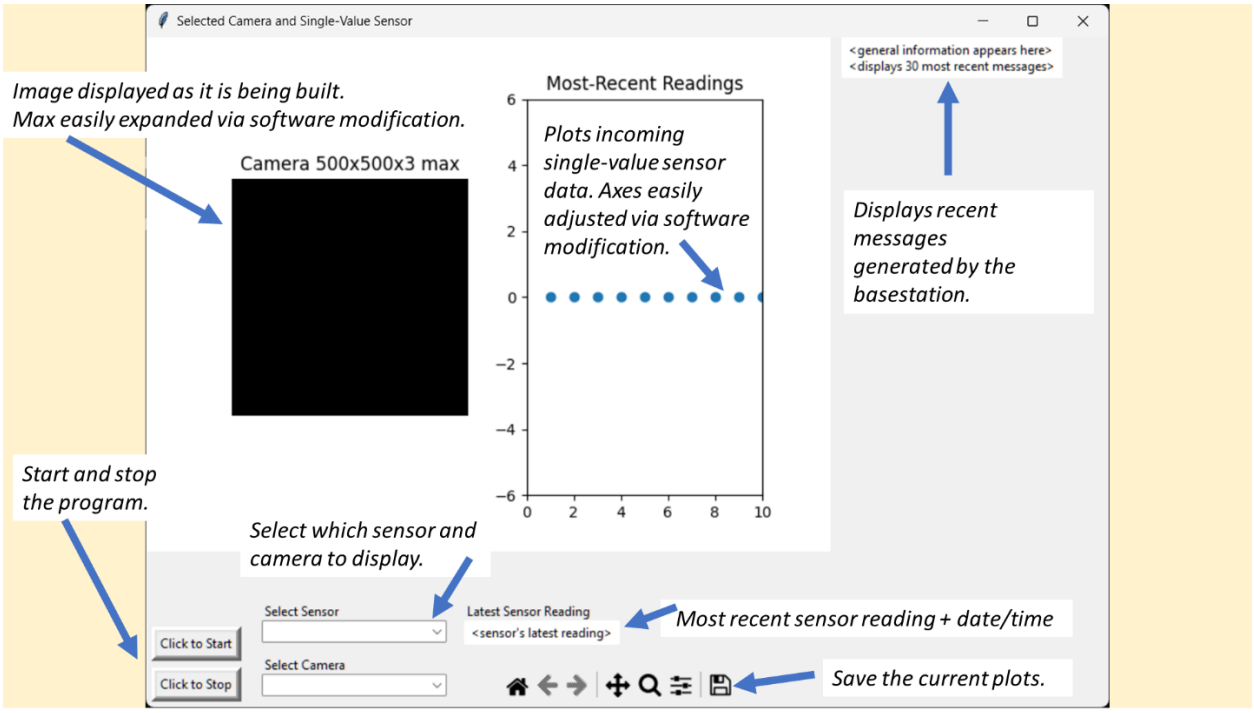


Figure 12. Annotated GUI produced by basestation.

Once started, the basestation's console will display an output similar to:

```
Press START to begin
17:39:39: USB_Serial_Connection: Starting
17:39:39: Serial Ports:
17:39:39: Detected 4 total ports
17:39:39:       COM8  :       COM8  :       Microsoft
17:39:39:       COM1  :       COM1  :       (Standard port types)
17:39:39:       COM6  :       COM6  :       Microsoft
17:39:39:       COM7  :       COM7  :       Arduino LLC (www.arduino.cc)

Configuring serial port COM8
Connected to PORT_A.
        Port Name:      COM8
        Port Baudrate:  9600

Looking for PORT_A handshake: Success. Returning handshake to PORT_A.
Handshakes exchanged

17:39:42: Awaiting Messages...

Message 2449 of type 0 ( 250 )      2 / 2
Message 2451 of type 0 ( 250 )      2 / 158
Message 2452 of type 0 ( 250 )      2 / 236
Message 2453 of type 0 ( 250 )      3 / 2
Message 2454 of type 0 ( 250 )      3 / 80
Message 2456 of type 0 ( 250 )      3 / 236
Message 2457 of type 0 ( 250 )      4 / 2
Message 2458 of type 0 ( 250 )      4 / 80
Message 2459 of type 0 ( 250 )      4 / 158
Message 2461 of type 0 ( 250 )      5 / 2
Message 2462 of type 0 ( 250 )      5 / 80
Message 2463 of type 0 ( 250 )      5 / 158
Message 2464 of type 0 ( 250 )      5 / 236
Message 2466 of type 0 ( 250 )      6 / 80
Message 2467 of type 0 ( 250 )      6 / 158
. . .
```

In the last set of lines there is the Message ID as sent by the camera, the type of message (see MessageTypes.html), the number of bytes in the message, the image row and starting column contained in the image segment contained in the message. Any diagnostic information can be contained in the console display.

As image segments are received, the basestation activates the indicated pixels on the GUI. An example is in Figure 13.

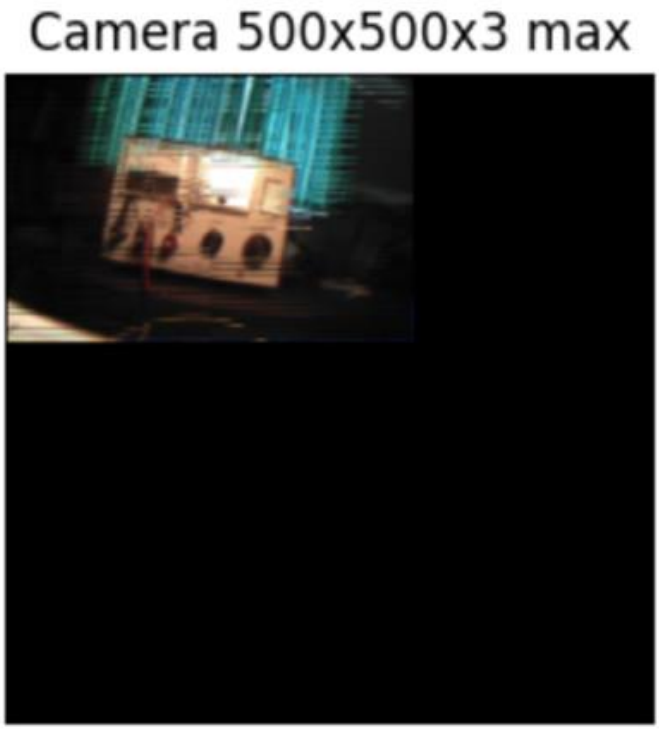


Figure 13. Example camera image capture.

Pixy2 camera resolution is only 316x208. That is why a portion of the display is black. The picture is fuzzy because the camera was moved while the snapshot was being taken. It took about 20 minutes for the picture to be taken, transmitted, and displayed. Certainly not real-time but suitable for single-shot observations.

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. 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)	**Part Number**	**Price (USD)**	**Total Number Required**	**USA Source**	**Global Source**
Basic Essentials - Grassroots Network					
Baseboard + USB C Cable (full featured)	RAK19001	13.50	1	Rokland	RAKwireless
Baseboard + USB C Cable (few features)	RAK19007	9.99	2	Rokland	RAKwireless
(2) Computational Core + Small Antenna	RAK11310	9.95	3	Rokland	RAKwireless
Temp & Humidity Sensor	RAK1901	4.60	1	Rokland	RAKwireless
(1) Total Cost		67.93			

Optional Accessories					
(4) Manual Precision Screwdriver	SKU:920430	10.00	1	Rokland	RAKwireless
(3) Stand-alone magnifying light	many types	40.00	1	Amazon	Amazon
(5) Solar Unify Enclosure IP67	RAKBox-B2	29.00	1	Rokland	RAKwireless
(6) Soil-Moisture Sensor	VH400-2M	45.00	1	Vegetronix	Vegitronix
0-5V Interface Module	RAK5811	6.00	1	Rokland	RAKwireless
Total if all purchased:		130.00			
For Exploring Image Transmission					
Arduino Microcontroller	Uno R3	28.00	1	Amazon	Amazon
USB 2.0 Cable	Type A/B	7.00	1	Amazon	Amazon
RaspberryPi single-board computer	RPi-v4	129.00	1	Amazon	Amazon
Ethernet Cable	select length	7.00	1	Amazon	Amazon
Micro SD Card	16gb	8.00	1	Amazon	Amazon
USB Micro SD Card Adaptor	UNI Reader	9.00	1	Amazon	Amazon
PixyCam + Connector Cable	Pixy2.1	70.00	1	Amazon	Amazon
Total if all purchased:		258.00			

(1) Prices are from RAKwireless at time of writing. They do not include shipping or taxes.
Rokland prices are generally higher but they pay for
shipping.
"Cost" refers to the sum of individual prices times the number
required.

(2) Be sure to specify the LoRa frequency appropriate to your country.
Rokland only sells for North America. RAKwireless' product page has
an
interactive map that shows which frequency to select. Notice that
some
nations do not allow unlicensed radio broadcasts.

(3) When purchasing a lamp, be sure it works well with your nation's
electrical outlets. Not all lamps are dual voltage/frequency.
Nor do all have the correct plug for a nation's electrical outlets.
A nation-specific adaptor added to the plug may be sufficient to fit the plug to the outlet.
See these examples of plug adaptors.

(4) The screwdriver that I received came with a sheath. Retain the sheath to protect the blade itself and the user when the tool is not in use.

(5) Depending on your needs, you may also want the pole-mouinting and antenna accessories. These add about $20 to the cost.

(6) 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. These are not direct substitutes. One has to study and understand the new sensor, and correctly integrate it with the host device and the soil in question.