MeshtASKtic: A conversion of Meshtastic's LoRa mesh network to support alternate transmission mediums.

Daniel Calco - dcalco@umich.edu
https://github.com/dcalkoj/MeshtASKtic
EECS 507 Fall 2024

***Abstract* - This paper presents a port of the LoRa-based software Meshtastic to instead use 433 MHz transceivers for sending messages. It begins by giving an overview of how Meshtastic functions, followed by a plan of how the software is to be modified. After modification and construction, testing shows the system to work, creating a mesh network over the 433MHz band. Some limitations exist in the form of message length restrictions, which hinder identification of individual nodes, but the system supports basic sending, receiving, and rebroadcasting.**

I. Introduction

Meshtastic is an open source, decentralized communication protocol for LoRa peer to peer networking. LoRa itself is a long range radio broadcasting technology utilizing a proprietary spread-spectrum technique, modulating in the unlicensed 915 MHz spectrum in the United States [1]. Advertising ranges of up to 20 kilometers in ideal conditions, LoRa-compatible chips are beginning to see more and more use in applications meant for low speed data transfers.

Meshtastic utilizes LoRa to form a mesh of nodes that are able to communicate text-based information, limited to 200B payloads. Nodes can freely contact any other discovered node in the mesh, or talk in any number of public channels to broadcast a message to any listening parties. Messages are sent over an epidemic (also known as flooding) routing system. Upon a transmission from one node to another, the receiver will check to see if the node is marked for retransmit, and if it is, will rebroadcast it again to any other nodes in the network. This system also works to give implicit ACKs, as the original transmitter may receive its own message back as a new transmission. Further Meshtastic API features extrapolate on the basic messaging system, adding in abstracted calls for how to share location via GPS, environment metrics, and traceroutes. Nodes can also be modified to alter retransmission behavior, being able to be flagged as sole clients, sensors, repeaters or routers depending on their intended functionality. [2]

Meshtastic's intended functionality is for decentralized communications, primarily general outdoors activities - such as hiking or skiing - cases where cell service may not be readily accessible or usable to transmit messages. However, nothing inhibits the use of Meshtastic for general purpose communications without cell provider interference. This is enhanced by LoRa, as a single transmission can bounce tens of kilometers under the right conditions with a suitable antenna and transmission power.

A very glaring security flaw exists in this type of opportunistic network that blindly sends out transmissions to any listening parties: the possibility to use transmission receipts to localize, and possibly track a sender down to their exact coordinates without an associated GPS tag via multilateration. [3]. While not yet attempted on a Mesh LoRa network, Xue et al. describes an attack of this method on the opportunistic network YikYak, an anonymous local message board [4].

Mesh networks provide an interesting field of exploration for hobbyists, developers, and penetration testers from the aforementioned information. While Meshtastic currently only supports LoRa as a transmission medium, it still seemingly poses an excellent base point for any general mesh networking system. It offers an open source codebase and is well documented on setup procedures and module functionality. This paper and project test how easy it would be to modify Meshtastic's firmware to use another transmission medium, the functionality, and further extrapolate if it would be feasible for any number of different mediums.

II. Transmission Mediums and their applicability to Meshtastic

A. WiFi 802.11 & BLE

WiFi and Bluetooth Low Energy (BLE) both support broadcast messages that could be used in the setup of a mesh network. A large problem exists to work around in both of these protocols however - both are already in use by default. WiFi would only be applicable for ESP32-boards, and any relevant

WiFi features would likely need to be disabled to begin testing broadcast messages. While BLE is available on both chipsets, again, Meshtastic already depends upon BLE and would require somewhat invasive code changes to modify.

B. ANT/ANT+

ANT is a unique protocol exclusive to the nRF chip, produced by and mainly for Garmin smart devices. ANT+ is an additive layer on top to support dedicated handling of sensor data such as heart rates or pedometers. Given the vulnerability described above of attempting to locate a party based off of transmissions, a mesh network sharing health data such as this would be a very important target. The only downside inhibiting ANT as a choice for this project is the relatively large entry barrier. A license key for ANT must be requested from Garmin to start development, and the chip's bootloader must be updated to reflect the new license permissions. While interesting, it deviates from the project goal of attempting to minimally invade Meshtastic's native framework.

C. 433MHz Radio

433MHz, registered under amateur radio operations in most nations, has very little entry difficulty for sending and receiving transmissions. The most common uses of it are seen in keyless entry devices such as for cars or garage door openers. Outside of this, it is also a general purpose hobbyist tool for sending wireless transmissions. 433MHz Transceiver kits are often found within starting modules for hobbyists looking to get into electrical design.

433MHz chips use amplitude shift keying (ASK) to perform short range communications, normally on the scale of tens of meters. Particularly, 433MHz chips use on-off keying (OOK), where 100% amplitude is counted as a '1', and anything lower a '0'. This transmission protocol can be manipulated to send full streams of data with the appropriate library.

In total, a pair of 433MHz chips were selected for use in this project - the SYN480R receiver and SYN115 transmitter. A quarter wavelength antenna of 17 cm is also mounted to both chips. While originally chosen due to timing constraints, 433MHz does offer a noteworthy advantage over a normal LoRa chip - the price point A complete set of transmitter + receiver costs about $2.00, while a sample LoRa chip (the SX1262, supported by Meshtastic) runs $7.62 from Mouser.

III. Meshtastic Overview

The Meshtastic firmware can run on any microprocessor with the bare minimum of LoRa and BLE support - although currently only officially supported are the ESP32 and nRF52840 chips. Boards using the nRF52 generally see much smaller power consumption, but lack Wi-Fi connectivity. [5] End users can purchase a number of prefab boards ranging in levels of complexity to set up. Devices will generally come with, at minimum, a LoRa chip, an antenna, and either one of the aforementioned processors. Higher end devices may include a screen, GNSS modules, or exposed GPIO pins. Some of the more common consumer devices include the LILYGO T-Beam and T-Echo (**Figure 1**). This project uses the WisBlock RAK4631 by RakWireless (**Figure 2**) due to the GPIO exposure and nRF52 chipset's support of ANT (discussed in Section IIIb.)

Figure 1: The LILYGO T-Beam (left, ESP32 chipset) and T-Echo (right, nRF52 chipset). Images sourced from lilygo.cc

Figure 2: The RAK4631 (green, center) mounted onto a RAK19007 baseboard. Image sourced from rakwireless.com

A. Class Hierarchy

The router of Meshtastic is a threaded task responsible for managing a single transceiver interface it is given. At the time of writing, Meshtastic only supports a single interface, although comments left in `Router.cpp` indicate this may change in future releases. Because of this, setup of the router is simple - an interface for general commands is defined on startup and never changed. This interface is of type `RadioInterface`, a pure virtual class used to communicate with an unspecified radio device. This class is neatly defined to include only the bare minimum, high level operations the router will call to do - send, receive, and sleep among other less frequent commands.

As a pure virtual class, `RadioInterface`'s abstracted functions must be implemented in a descendant. The full inheritance tree is shown in **Figure 3**. To talk to LoRa chips, Meshtastic uses RadioLib, a general purpose wireless communication library. This is done in `RadioLibInterface`. Specific functions again are overridden and handled in the next level down class of the actual LoRa chipset in use for each device, such as the `SX1262Interface`, or `LR1110Interface`.

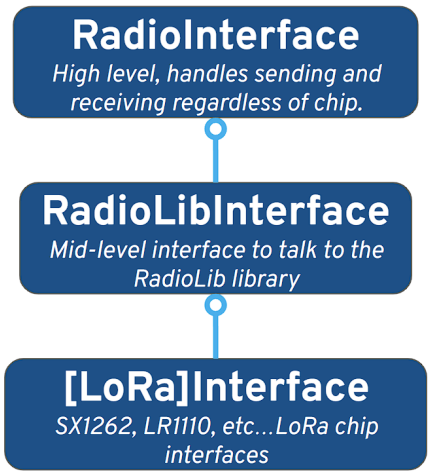


Figure 3: Inheritance structure of RadioInterface

B. RTOS Overview

Meshtastic implements a basic RTOS loosely modelled after FreeRTOS, allowing for the implementation of threaded functions. Two main cases of this functionality are utilized by Meshtastic's firmware.

RTOS Use 1 - `OSThread` **&** `RunOnce()`

The OSThread class can be inherited in any new class meant to be a standalone module, as long as `RunOnce` is implemented. `RunOnce` will be run each time the thread gets called, and returns an int32 telling the scheduler the next period to run the thread at. Use 1 is used for higher level modules which run periodically, such as the NodeInfoModule, which will send out information about the node every 15 minutes.

RTOS Use 2 - `NotifiedWorkerThread`**,** `notifyLater()` **&** `onNotify()`

`NotifiedWorkerThread` derives from `OSThread`, and provides support for an interrupt service routine (ISR) within the RTOS. Instead of specifying a period for the next run, `notifyLater()` can be used to set a time to schedule a task to be resumed at a later point, setting a timer. `onNotify()` will then be run once the timer is fired. **Use 2** is used for the existing `RadioLibInterface` code for collision avoidance. A `notifyLater()` is generated to wait a moment before transmitting whatever may be in the Tx queue to ensure that there is nothing currently being received. If there is, another `notifyLater()` is generated, continuing until there is availability.

C. Proposed Implementation

RadioLib does not support ASK, necessitating the use of a different library: RadioHead. RadioHead provides code for efficient encoding of ASK packets to be used for text transmissions. This proposed addition is shown in **Figure 4.**

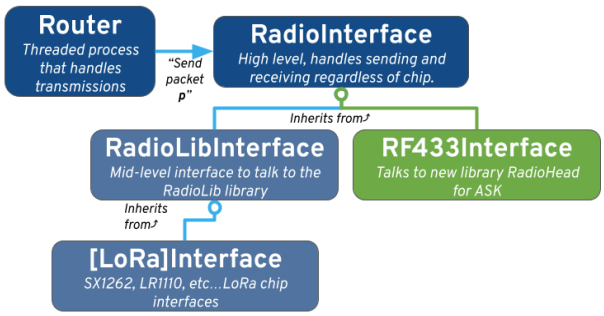


Figure 4: Proposed modification of Meshtastic's class hierarchy, featuring new class `RF433Interface` to be provided to `Router`.

IV. Development

A. Debugging

RAKWireless does provide a dedicated debugger for use on the WisBlock platform [6], but due to shipping times, it was not used for this project. Instead, a generic J-Link debugger was with similar success. Setup procedure is described in Appendix A.

B. RadioHead Library

RadioHead claims to support the NRF52 chip, but does not actually provide code for ASK transmissions using it, missing preprocessor directives to set up the relevant timers. This was resolved somewhat easily, but still ended up requiring a port of RadioHead's code. This is endorsed under the GPL license that RadioHead is developed with.

C. Transmitting

Modifying transmissions to work with 433MHz was the most intensive part of the workload. In vanilla Meshtastic, sending is done via two "send" commands - the first enqueuing a transmission into a vector. The actual transmit comes later once the receive line has been verified to be inactive. For simplicity purposes, the verification of an inactive receive line can be foregone - which is how this project managed the first implementation of send. Unfortunately, RadioHead's supported transmission size is nearly a quarter of what Meshtastic traditionally limits messages to. This reduction of 200B transmissions down to 67B does not fully inhibit the system from working as expected, but does raise a bug further discussed in Section V & VII. On a second revision, the collision avoidance code was re-added to the `RF433Interface` transmission code, still working as Meshtastic originally intended.

D. Receive

Receiving is unfortunately not able to be done as neatly as the original Meshtastic protocol. LoRa chips supported by Meshtastic feature interrupt pins to pull upon receiving a message, which the 433MHz chip lacks. Receiving was instead implemented via polling with the `runOnce` function, on a frequency of 5ms. Most of the time-critical functions originally stem from `RadioLibInterface`, which no longer run due to the swap of interfaces for the router. As such, a 5ms deadline does not pose any significant challenges for the system to schedule.

V. Evaluation

Three RAK19007 boards were flashed with the same modified firmware to test the connectivity of the systems. The system is shown in **Figure 5**, with the SYN480R receiver being connected to GPIO pin 17, and the SYN115 transmitter connected to GPIO pin 34.

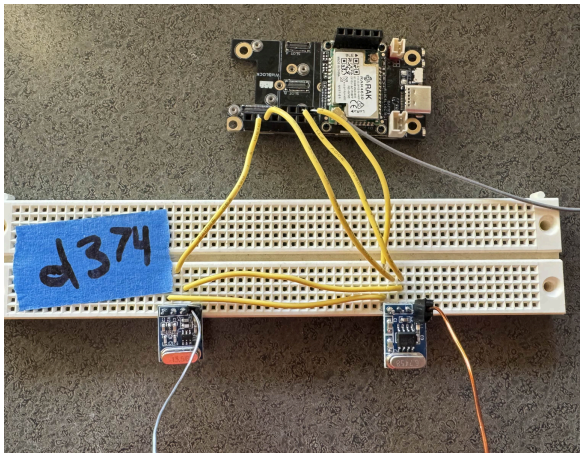


Figure 5. RAK19007 modified for custom 433MHz firmware. Not pictured: BLE antenna trailing off to the right (grey wire).

Simple tests of functionality conducted include the modified system's ability to send, receive and rebroadcast a message using the 433MHz mesh network. Maximum distances of 20m were achieved with line of sight using the aforementioned setup and antenna. Without line of sight, distances dropped to around 15-18m depending on how many walls were being penetrated through. The SYN115 chip is only rated for a maximum of 3.6V (running off the WisBlock's 3.3V output), but the SYN480R receiver can accept up to 5.5V, meaning there could be room for further evaluation. Messages were also able to be successfully "hopped" from one device to another; a node 30m away from the source was still able to receive the transmitted message when a third node was placed between the two of them.

Due to the change in transmission length from LoRa to 433MHz, one function was disabled - the `NodeInfoModule`. Some transmissions are flagged with a certain module destination to be routed to upon receiving, with the `NodeInfoModule` broadcasting information about the sending node's location and name. These transmissions tended to be around 137B long, which was unable to be sent in a single RadioHead transmission. While messages can still be sent into the general broadcast chat without this information, direct messages cannot be made

unless the nodes have already exchanged information (done prior via LoRa). An example of this is shown in **Figure 6.**

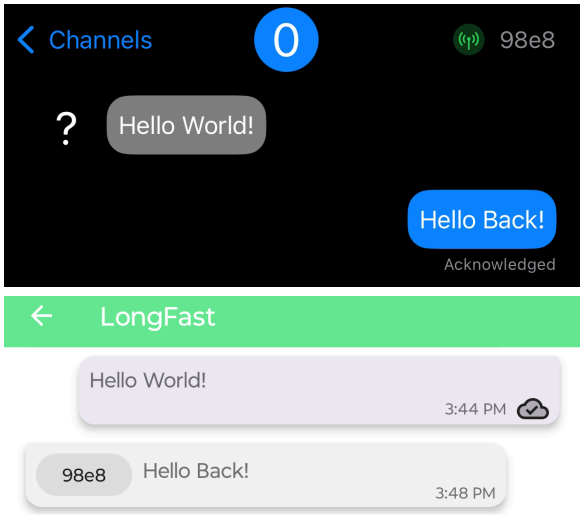


Figure 6. "Hello World!" has been received from an unknown sender - a node which never ran the `NodeInfoModule` (top). Node 98e9 had previously run `NodeInfoModule` while still on the LoRa firmware, so the sender (bottom) can still identify 98e8's messages.

This work was done without large modifications to Meshtastic's overall framework. Only one file was modified (`Router.h`) to change the incoming interface, all other files were able to be inserted into the existing firmware without conflict.

VI. Conclusions

The total time for completion of the bare minimum framework of swapping the interface to use a non-LoRa chip took around 25 hours starting from scratch with relatively little information about Meshtastic's workings. Further time was spent refining code and adding back in other Meshtastic features such as collision avoidance, as well as changing and testing polling speeds for the receiver.

One of the goals of this project was to demonstrate the ease and feasibility of modifying Meshtastic to run under a different transmission protocol, which was completed. This code could very easily be flashed to any nRF52-based chip able to already run Meshtastic, so long as two GPIO ports are available. Outside of this, the system worked as a general purpose mesh communications network - barring the NodeInfoModule problems - nodes could send, receive, and rebroadcast messages.

All development history and work can be found on Github, forked off of Meshtastic's main branch.

VII. Continuations/Proposed Future Work

The most glaring issue presented is the 67B limit on messages. The easiest solution would likely be to split oversized packets over multiple transmissions, but this would require further tweaking to Meshtastic's code, particularly to `Router.h`. There would also have to be additional logic in relation to checking these packets to rearrange them in correct order, and to also keep the system aware that it is waiting on additional packets.

If continued with another protocol to add onto Meshtastic, it may be worthwhile looking into ANT/ANT+. A theoretical mesh network based on sharing the things that ANT specializes in - health data - could very well be a large security risk that is worth investigating. Porting this project to support ANT would give a means of testing.

VIII. References

[1] Mekki, E. Bajic, F. Chaxel, and F. Meyer, "A comparative study of LPWAN technologies for large-scale IoT deployment," Elsevier ICT Express, vol. 5, no. 1, pp. 1–7, Mar. 2019.

[2] https://meshtastic.org/docs/configuration/radio/device/

[3] Sohel, "Localization, Tracking, and Data Transmission Using LoRa". May 2024.

[4] M. Xue, C. Ballard, K. Liu, et al., "You can yak but you can't hide: Localizing anonymous social network users," in Proceedings of the 2016 Internet Measurement Conference, ser. IMC '16, Santa Monica, California, USA: Association for Computing Machinery, 2016, pp. 25–31, isbn: 9781450345262. doi: 10 . 1145 / 2987443 . 2987449.

[5] https://meshtastic.org/docs/hardware/devices/

[6] https://meshtastic.org/docs/development/firmware/nrf52-guide/

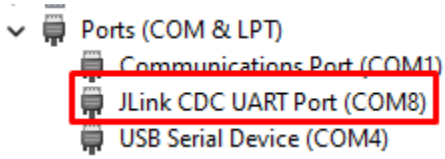
Appendix A. Debugging Instructions with J-Link

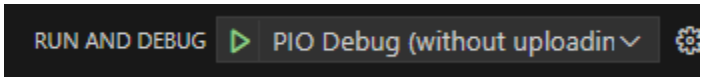
1. Read https://docs.platformio.org/en/latest/plus/debug-tools/jlink.html
2. Flash firmware normally over serial/USB connection.
3. In platformio.ini, add:

```
debug_tool = jlink

; SWD interface
upload_protocol = jlink
```

4. Attempt to upload normally with → macro in vscode
5. Verify platformio has detected the upload protocol as `jlink`.
6. **Platformio will attempt to download drivers, but you will still need to download and install the official J-Link SEGGER drivers.**
    a. Restart computer
7. Verify JLink is identified in device manager. Debugging will not work if it is identified as a generic COM.



8. Make JLink connections using SWD.
    a. Both JLink and the device must be powered.
9. Again upload normally with → macro in vscode. Device should flash with JLink instead of Slink.
10. 

    a. Debug must be run without uploading, hooking up the RESET pin.

- In case of "JLink not found", try restarting the computer.
    - If debugging was working and suddenly stops, general solution is to restart the computer.
- If the nRF device gets stuck in boot loop, unplug and replug it in.