

Red Teaming with LoRA

March 5, 2026 @ SunSecCon

Venky Raju

Maker, Hacker, Engineer, Reverse Engineer

Agenda

1. Intro to LoRa & Meshtastic
- B. Hands-on with Meshtastic
- iii. Red-teaming with LoRa / Meshtastic
- Δ Building your own Meshtastic image

What is **LoRa**?

What is LoRa (Long Range)?

- A proprietary radio communications technology.
- Based on spread spectrum modulation.
- Developed by a French company, Cycleo, in 2010.
- Cycleo was acquired by Semtech in 2012.
- Operates in the unlicensed, sub-Ghz ISM bands (US 902-928 MHz)



LoRa vs WiFi / Bluetooth



WLAN
Low Gbps
100 m



WPAN
Low Mbps
10 m



LPWAN
Low kbps
2-20 km

Unlicensed Spectrum

Low Power

Applications

Long Range

Low Speed

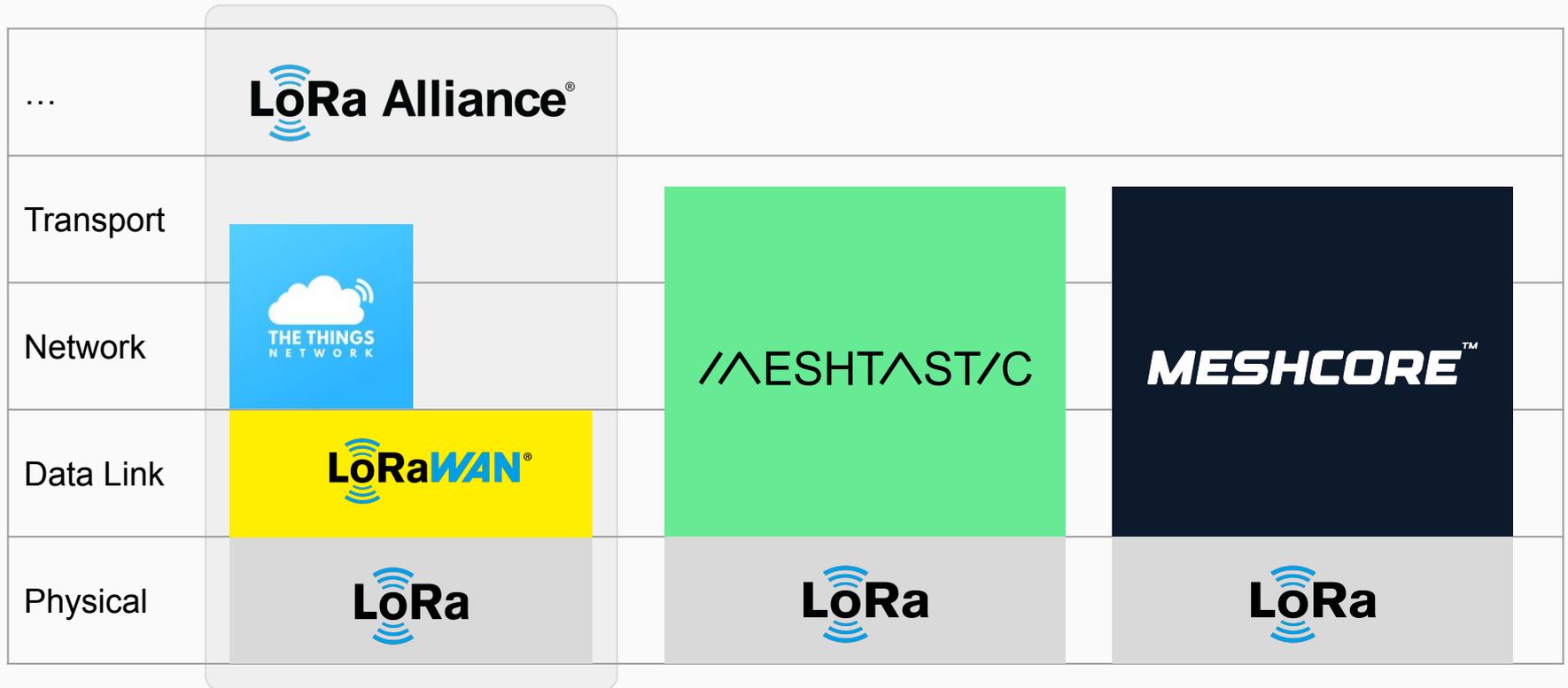
- Sensor networking
 - Industrial IOT
 - Smart Cities
 - Agriculture
- Community networking
 - Public networks
 - Ad hoc / Conferences
 - Hiking / Expeditions

How do we use LoRa?

After all, LoRa is a physical layer protocol



There are quite a few ways to use LoRa!



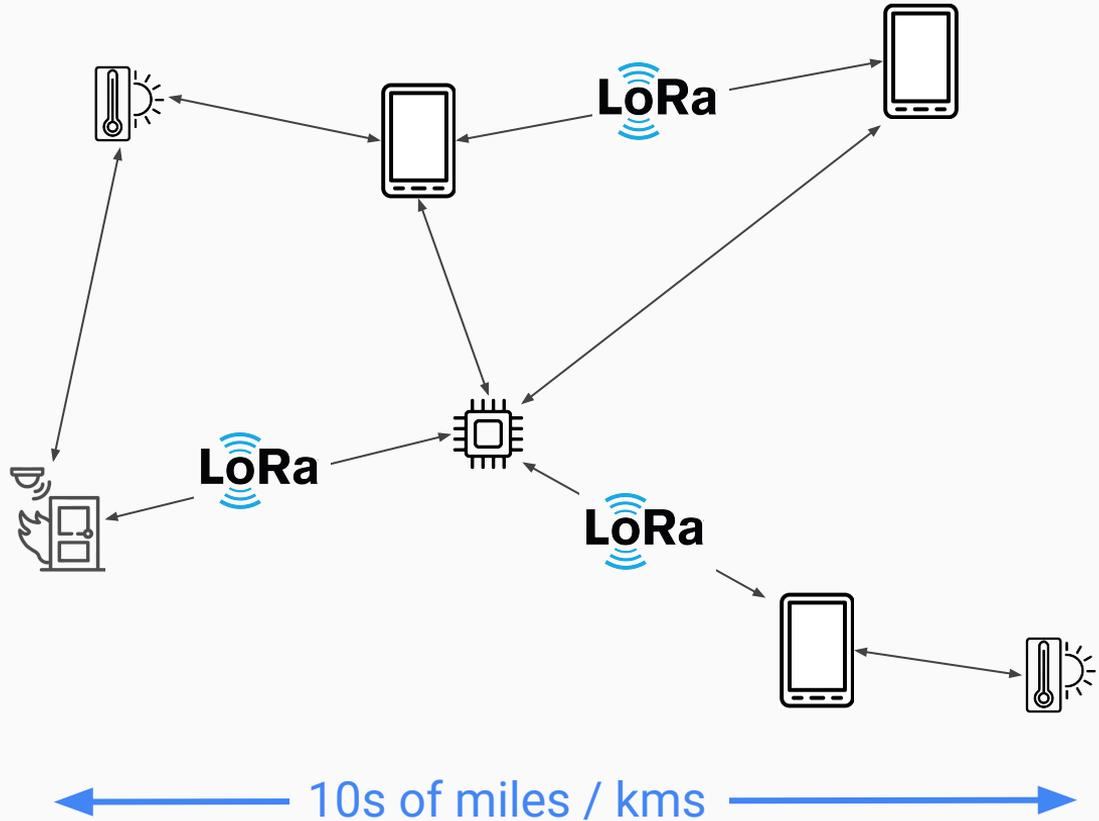
What is Meshtastic?

A mesh network of devices that use LoRa as the wireless link.

Fully decentralized, and every device can send, receive or relay messages.

100% open source and community driven!

<https://meshtastic.org>



TTN vs Meshtastic vs Meshcore?

TTN - Consortium-driven, focussed on industrial IoT use cases.

Meshtastic - User friendly, community networking and personal IoT.

Meshcore - Similar use cases as Meshtastic, arguably more capable.

Hands-on with Meshtastic

How do I get started?

STEP 1:

Build or purchase a
Meshtastic-compatible
LoRa device

Handheld

Looks like a RIM Blackberry

LCD screen with QWERTY keyboard

Ideal for messaging



LILYGO T-Deck



LILYGO T-Deck Plus

Pocket | Belt

Pair with a smartphone app
(iOS / Android) for screen
and keyboard

Messaging and some IOT



LowMesh Pocket-S
Solar / Battery



RAKwireless
WisMesh

Node | Module

Pair with a smartphone app
(iOS / Android) for screen
and keyboard

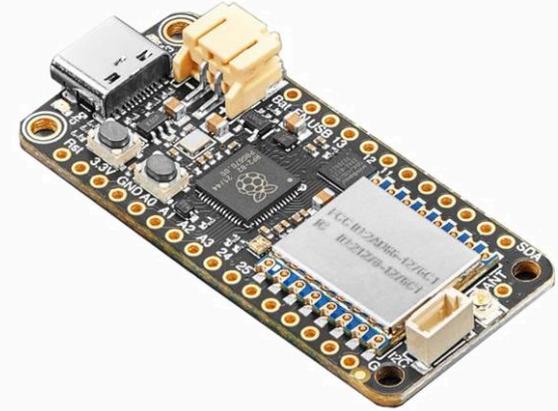
May have a small on-board
OLED display

Breakout pins available for
some functions

IOT + Hacking



Heltec ESP32 WiFi + BT
LoRa Node



Adafruit RP2040
+ RFM95 LORA

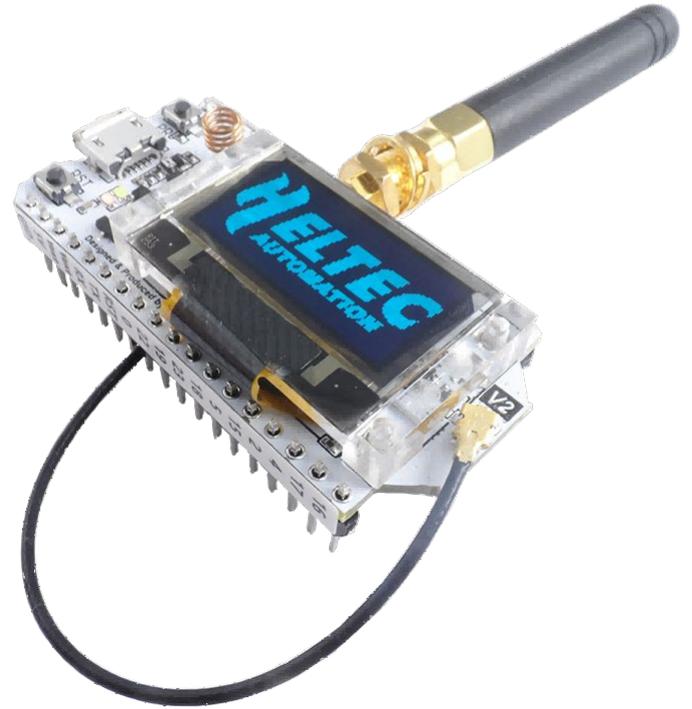
How do I get started?

STEP 2:

Flash the Meshtastic firmware onto your shiny new device!



STOP



First, attach the antenna!

Two ways to flash your device

Web Flasher:

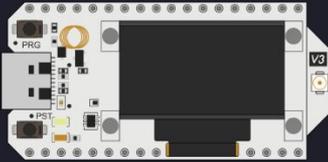
- For ESP32-based devices
- Requires USB drivers

Drag-n-Drop:

- For RP2040 and nRF52 devices
- Typically do not require a driver
- Device shows up as a drive (UF2 bootloader)

<https://meshtastic.org/docs/getting-started/serial-drivers/>

FLASHER



Heltec V3



2.6.11.60ec05e Beta



Flash

Device

Plug in your device via USB. Please ensure the cable is not a power-only one.

Firmware

Choose from the release options or upload a release zip downloaded from Github.

Flash

Flash your device. Choose whether you wish to update your device or wipe the flash and install from scratch.

[Open Serial Monitor >_](#)

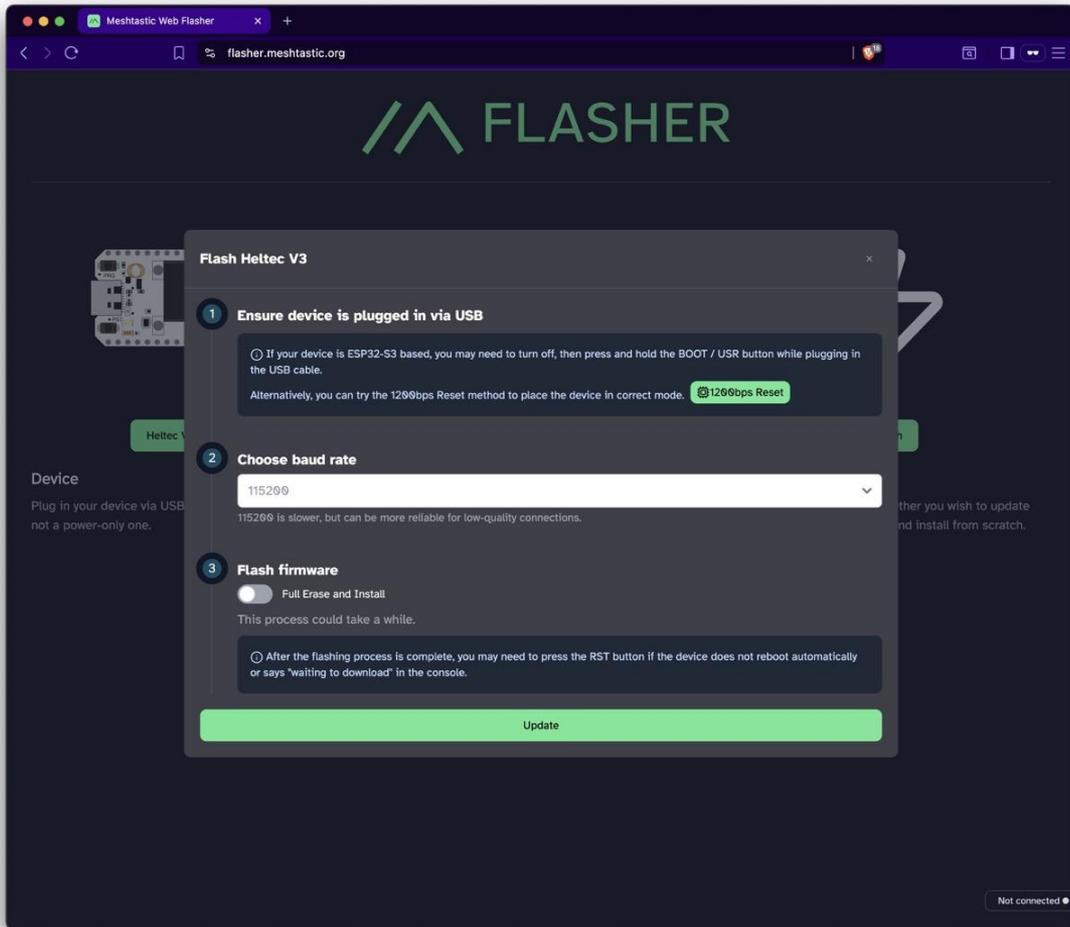
[Meshtastic Docs](#)

[Contribute on GitHub](#)

[Language](#)

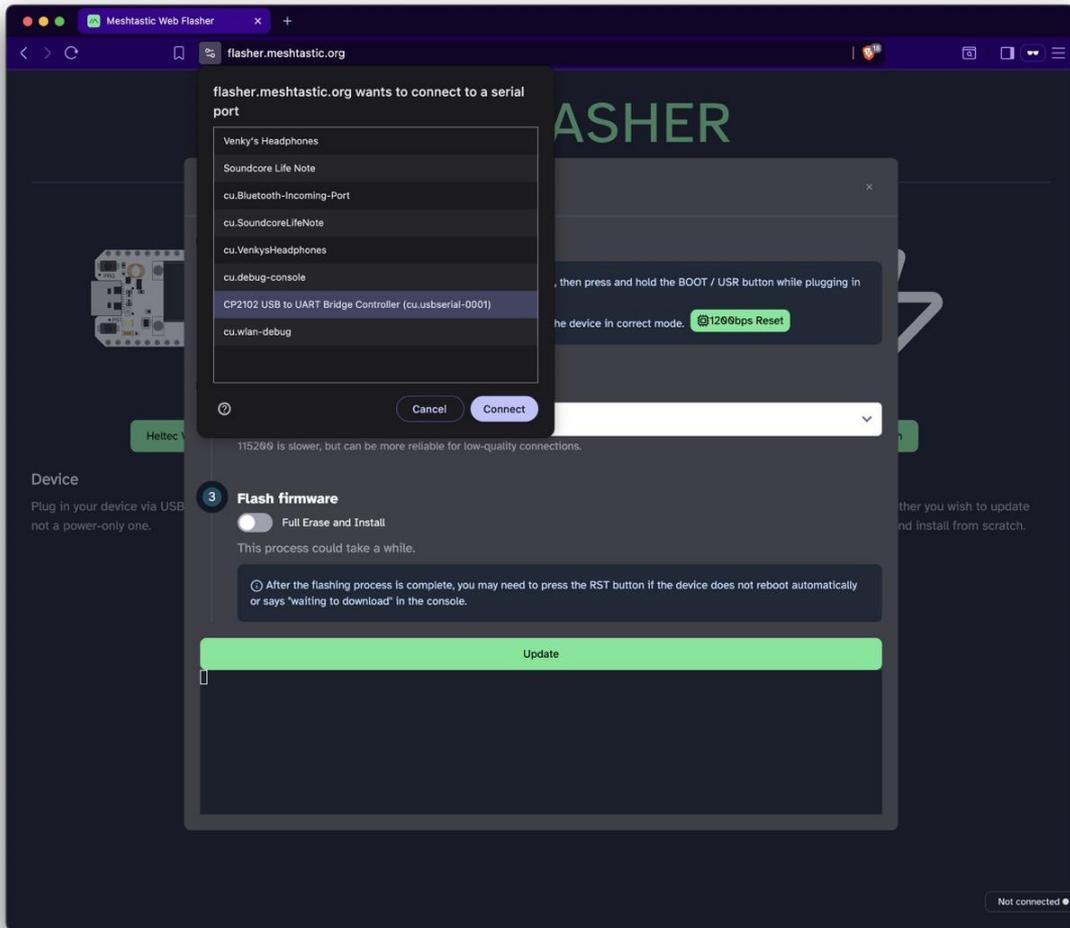
1

Connect your device and start the update wizard.



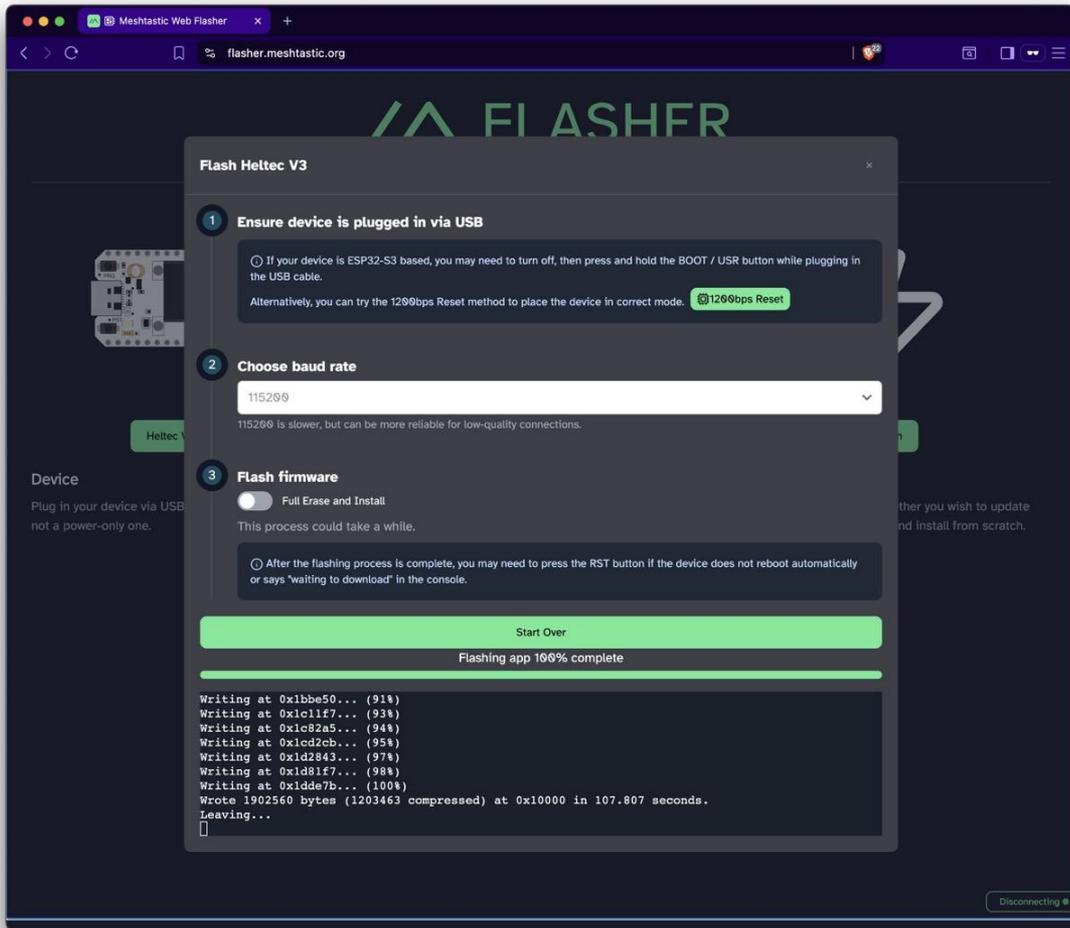
2

Select the virtual COM port and initiate the update process.



3

Wait until you see the “Leaving...” message after the update finishes.



Time for a
keyboard and
larger display!





Connected Devices

Manage your connected Meshtastic devices.

 Change Language: English



No devices connected

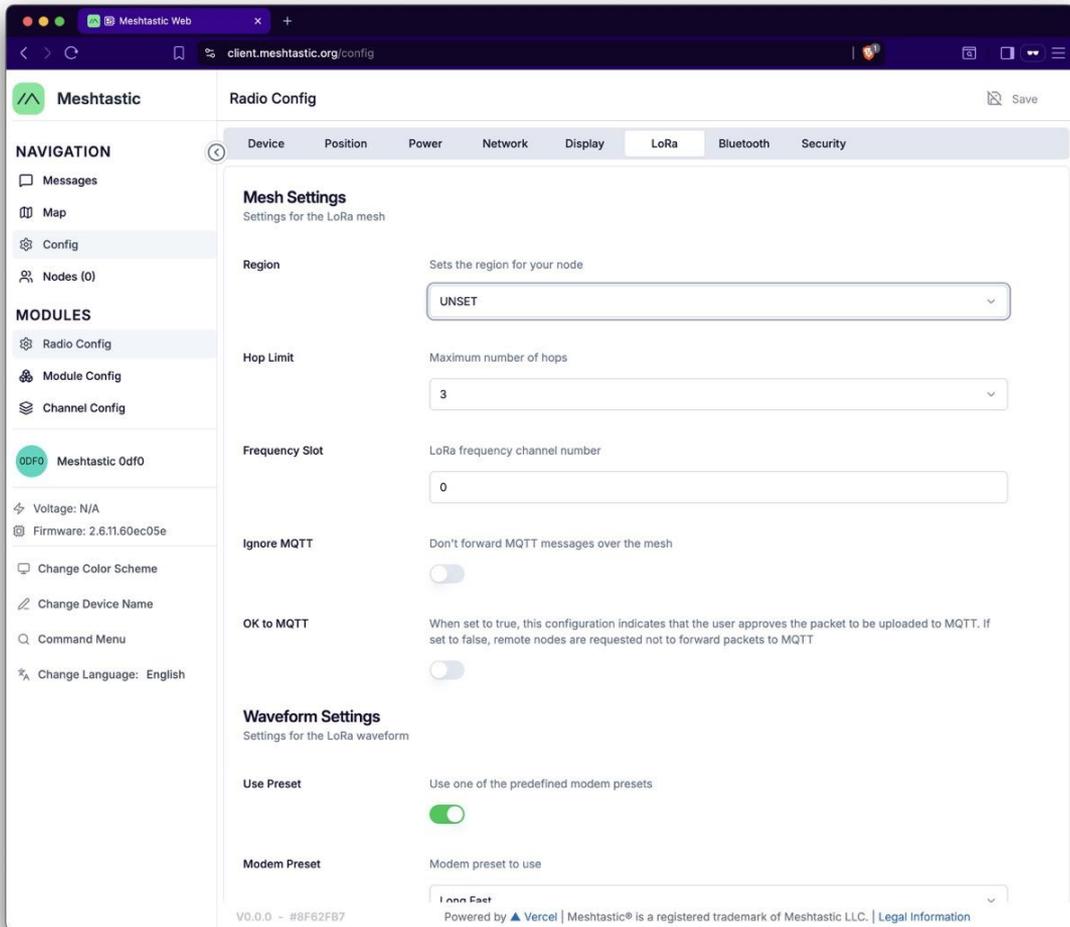
Connect a new device to get started.

[+ New Connection](#)

1

Select the “LoRa” tab under
Modules > Radio Config

Change the region to US

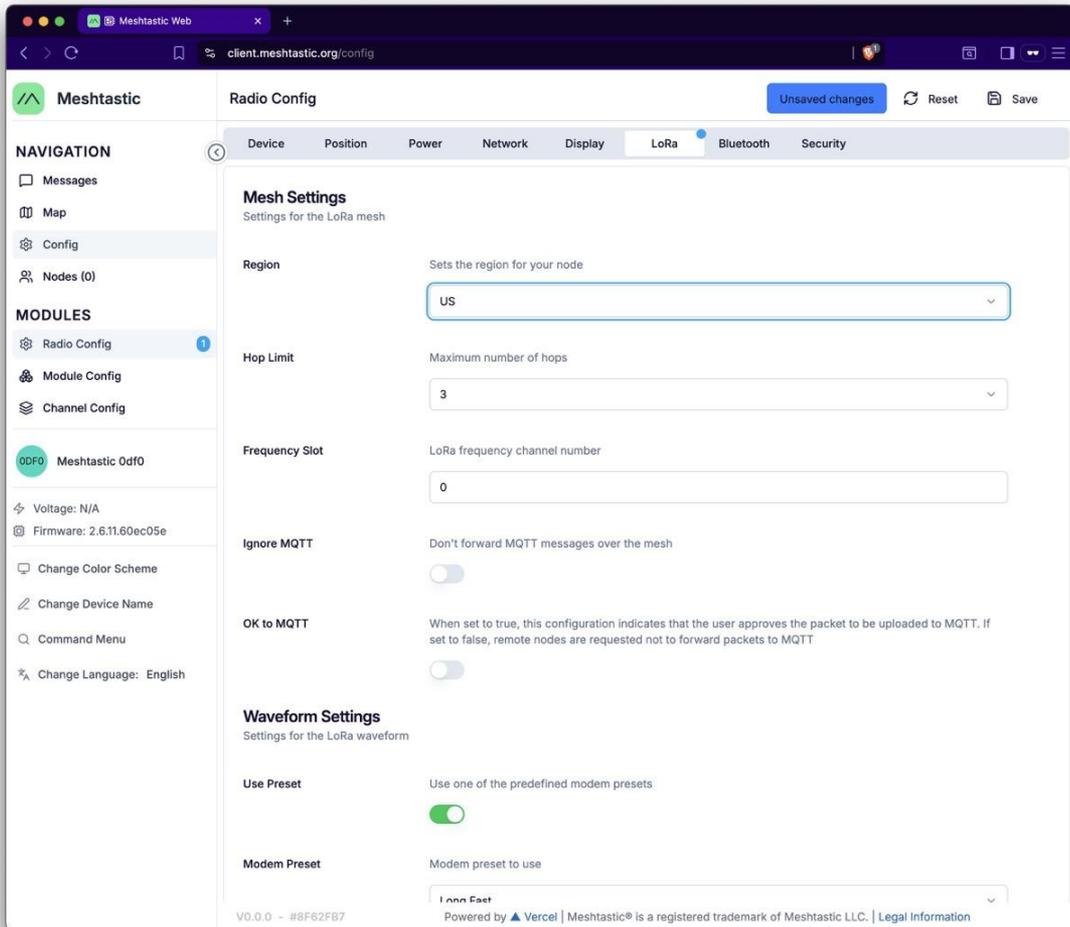


2

Click on “Save”

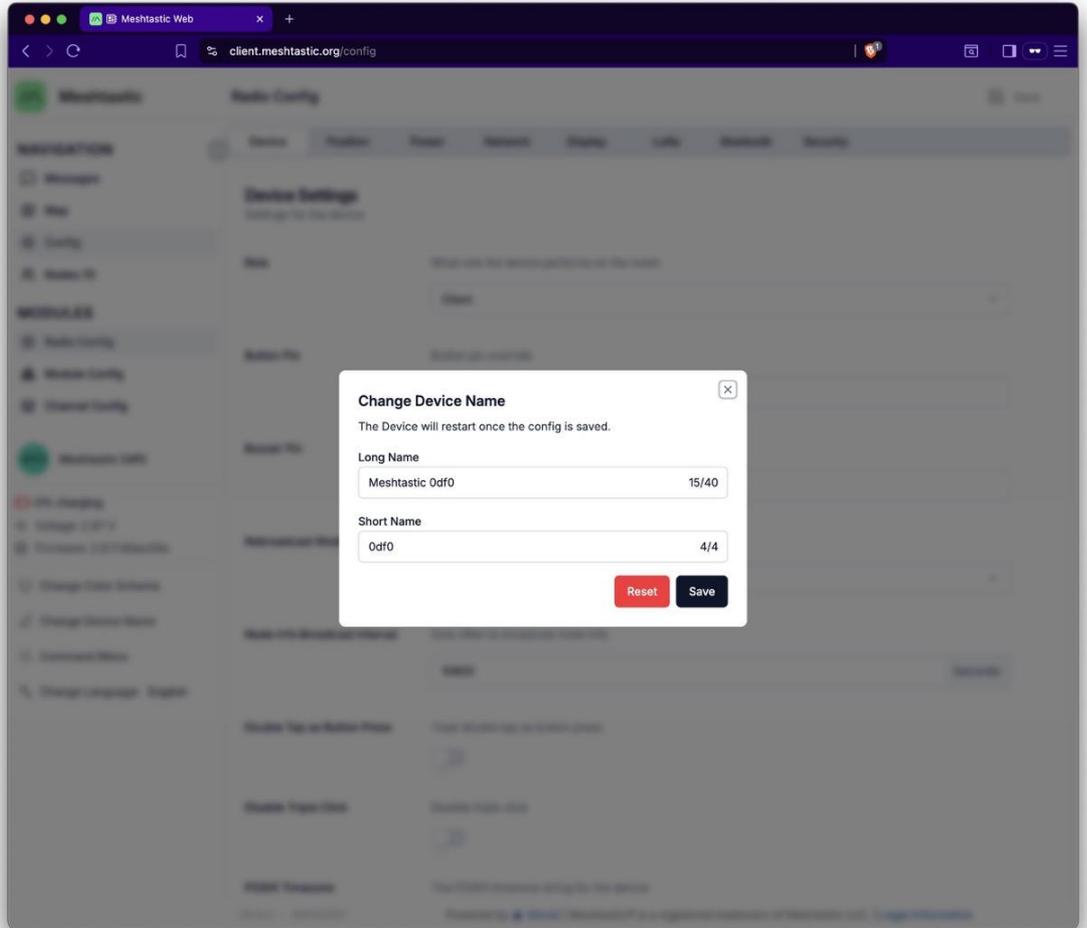
The device should reboot.

Reconnect when the device is ready.



3

Change the device name and save.



The Meshastic Stack

/MESH/STACK



Broadcast using
Managed Flooding

Reliable Zero-hop
Messaging

Best-effort Zero-hop
Messaging

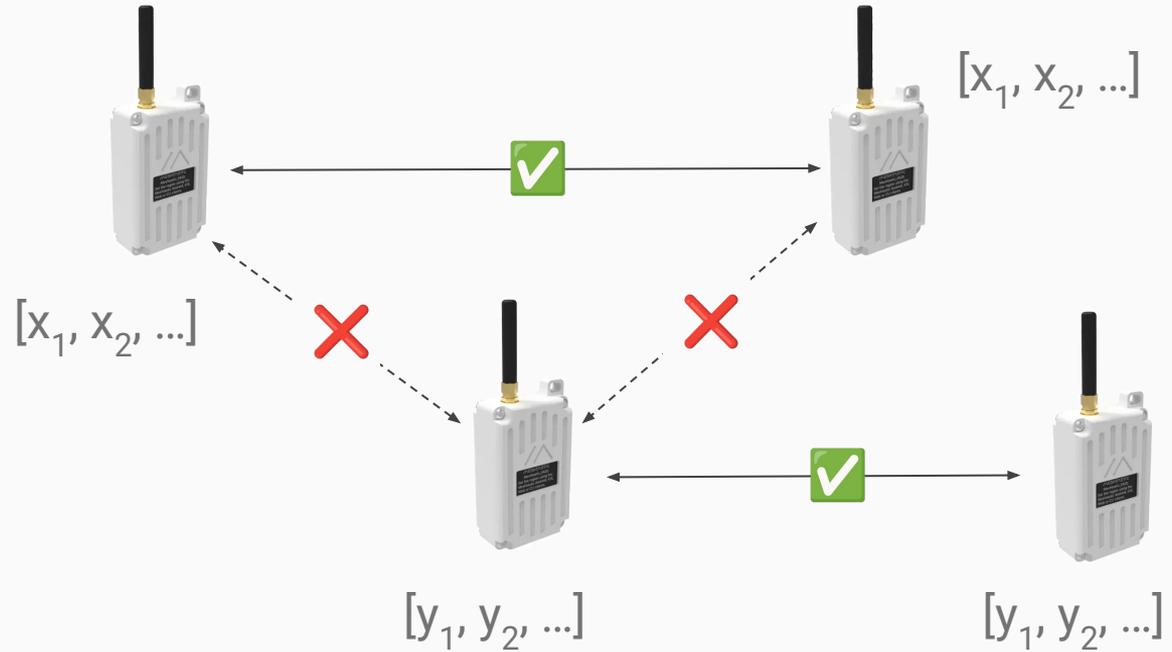
Protocol Buffers

Framing

Radio parameters

Radio Parameters

For two LoRa nodes to communicate, their radio parameters must match.



LoRA Parameters

- Spreading factor
(bits per chirp)
- Bandwidth
(slice of spectrum)
- Coding Rate
(amount of redundancy)

Example

SF = 11 bits/chirp

BW = 250 KHz

Coding Rate = 4/5

Chirp time $T_c = (2^{SF}/BW) = 2048/250k = 8.2ms$

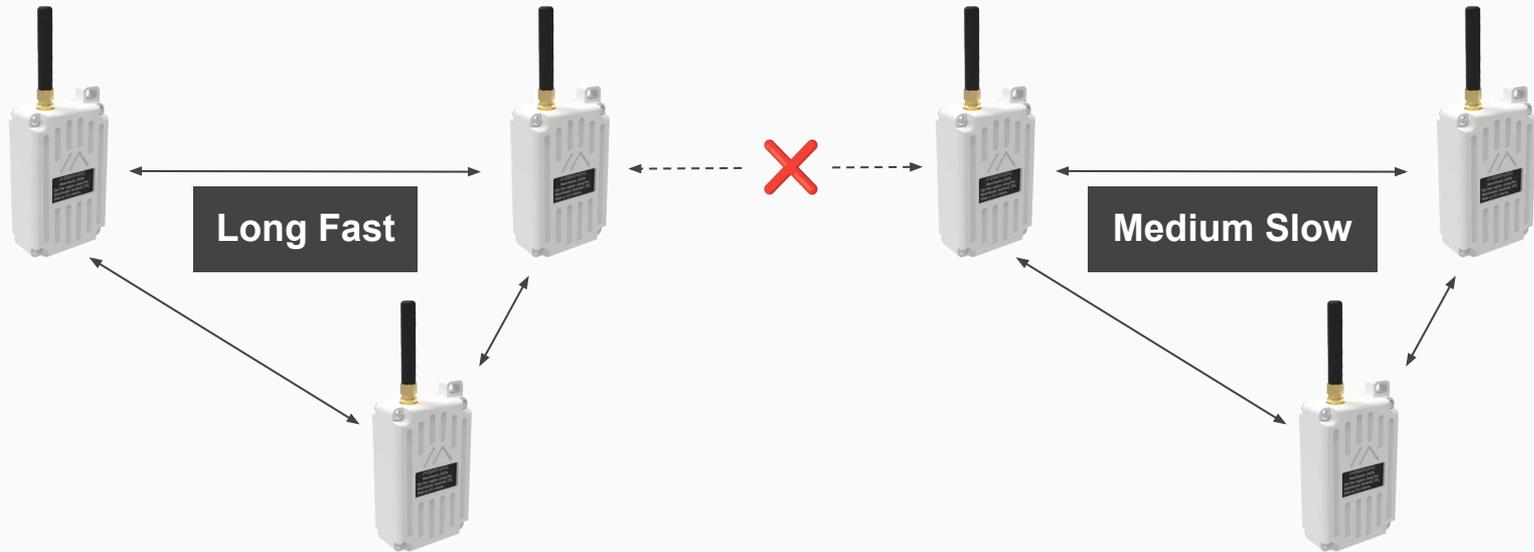
Bit rate = $SF/T_c = 11/8.2ms = 1.34 \text{ kbps}$

Data rate = Bit rate * Coding rate = **1.07 kbps**

Meshtastic Presets

Preset Name	Data-Rate	SF / Symbols	Coding Rate	Bandwidth
Short Turbo	21.88 kbps	7 / 128	4/5	500 kHz ¹
Short Fast	10.94 kbps	7 / 128	4/5	250 kHz
Short Slow	6.25 kbps	8 / 256	4/5	250 kHz
Medium Fast	3.52 kbps	9 / 512	4/5	250 kHz
Medium Slow	1.95 kbps	10 / 1024	4/5	250 kHz
Long Fast (Default)	1.07 kbps	11 / 2048	4/5	250 kHz
Long Moderate	0.34 kbps	11 / 2048	4/8	125 kHz
Long Slow	0.18 kbps	12 / 4096	4/8	125 kHz

If you don't see your friend, they may be on a different network!



What about interference from other LoRa networks?

Meshcore, TTN and other LoRaWAN networks share the same LoRa physical layer.

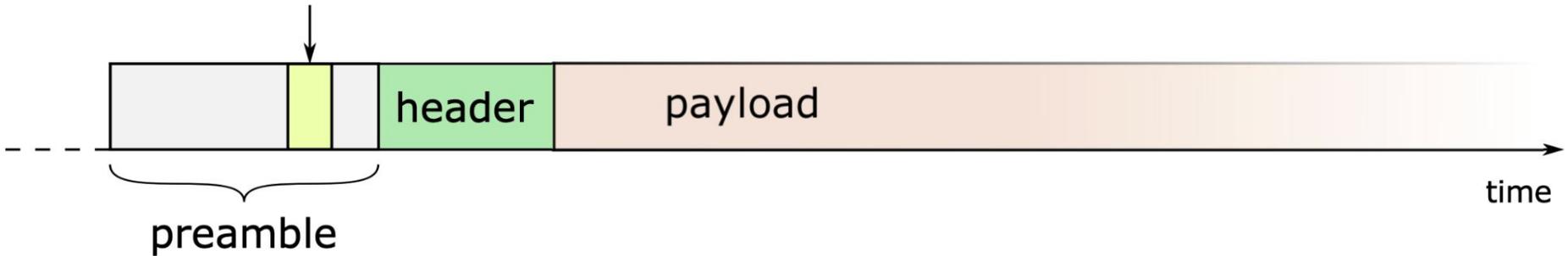
How can my Meshtastic node avoid processing packets of those networks?

LoRa Framing and the “Sync word”

0x2B for Meshtastic

sync word

sync word: 2 symbols
header: 8 symbols
payload: N_{pl} symbols



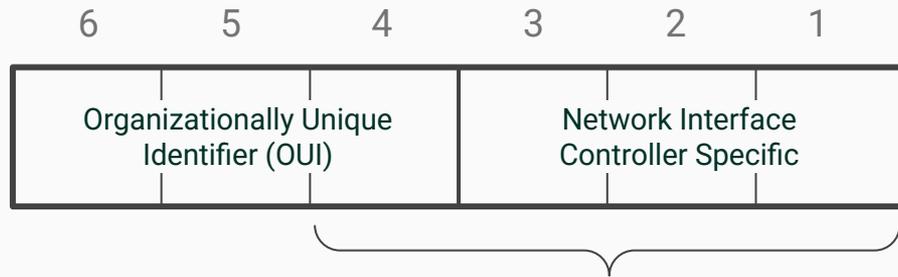
Source: **Vincent Savaux, Christophe Delacourt, Patrick Savelli.**
Considering Sync Word and Header Error Rates for Performance
Assessment in LoRa System. Wireless telecommunications
Symposium, Apr 2021, San Francisco (virtual), United States.



Node Addressing

Lower 4 octets of
Bluetooth EUI
(aka MAC address)

Broadcast is *0xFFFFFFFF*



Meshtastic Node ID

Zero-Hop Direct Message

Channel Activity Detection
Transmit message



Source ID = A
Destination ID = B
Want ACK = 1
Data

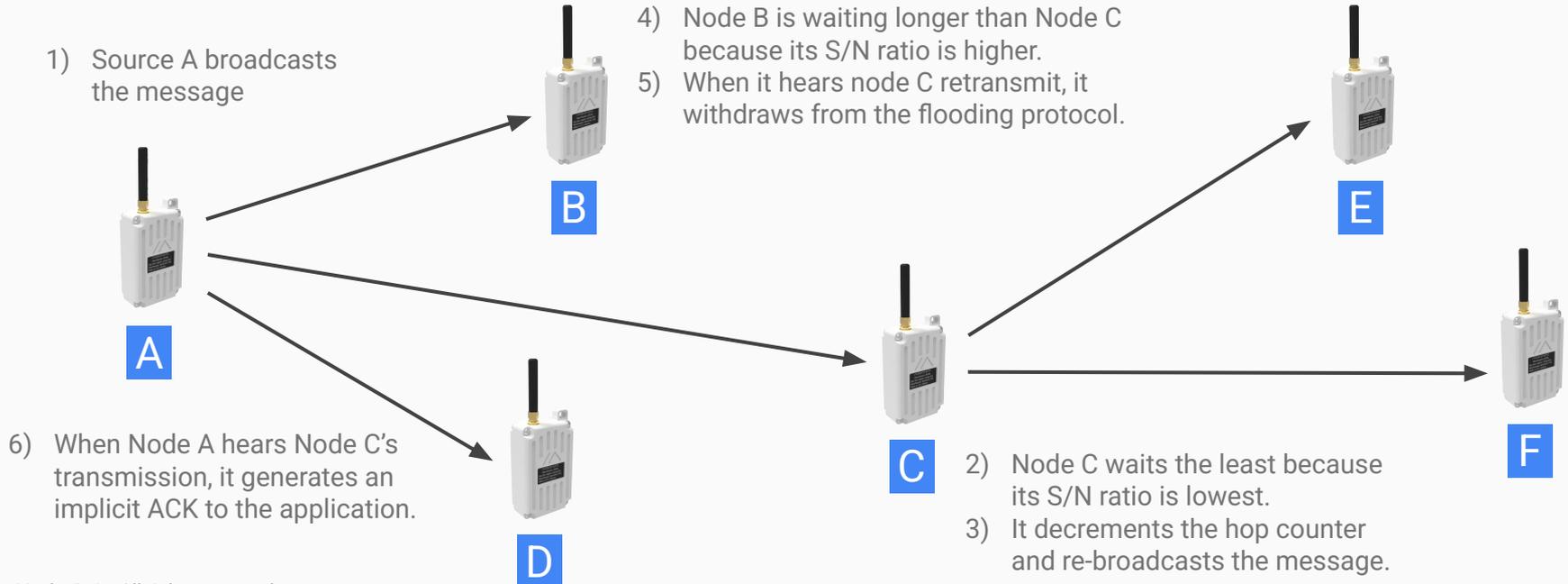


Channel Activity Detection
Transmit message

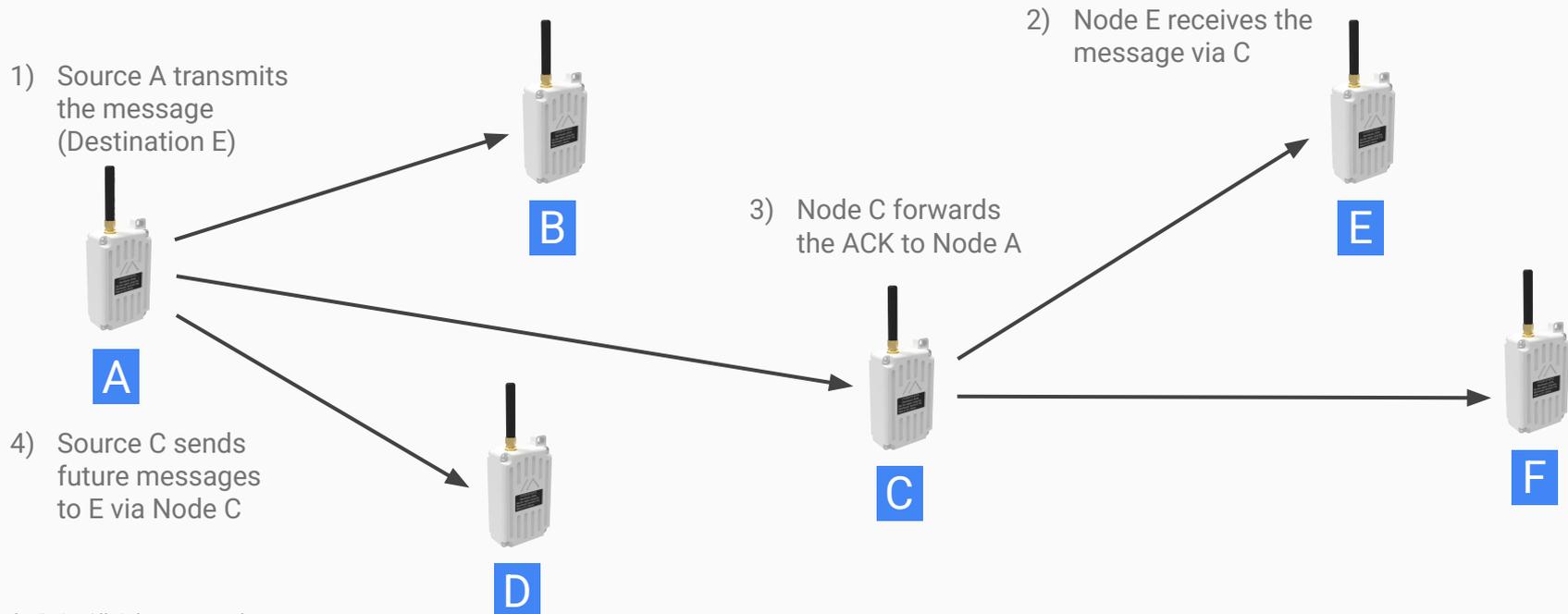


Source ID = B
Destination ID = A
ACK

Broadcast Message [Managed flooding]

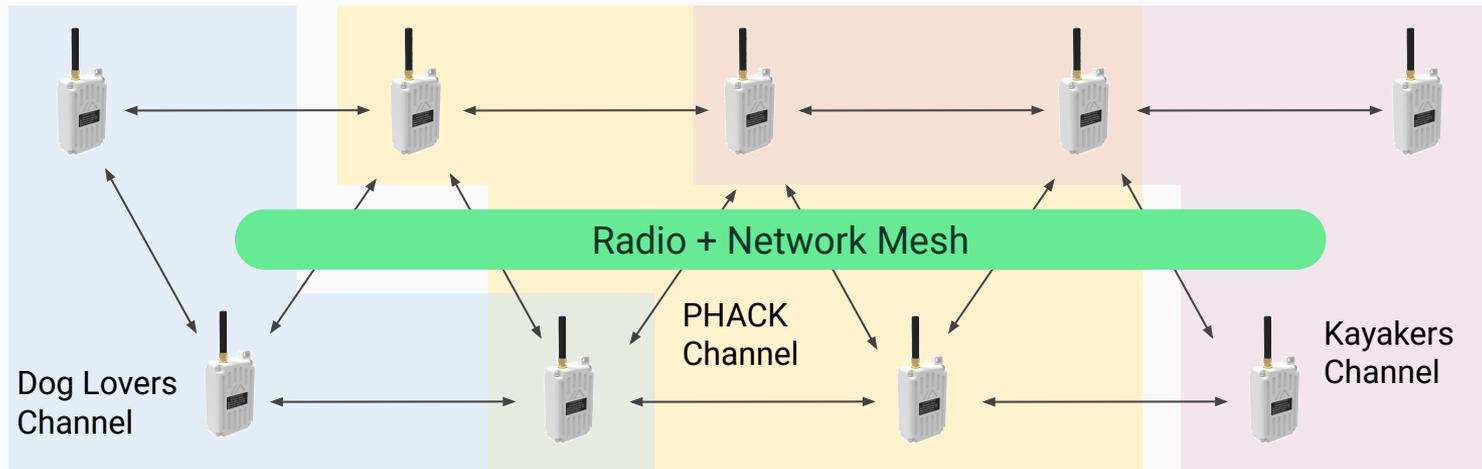


Multi-hop Direct Message [Managed flooding + memory]



Channels

[A sub-mesh, or community within the larger mesh]



About Channels

A channel is identified by a name and an encryption key

A node may belong to a maximum of 8 channels

All nodes are members of the unnamed Channel 0, with encryption key AQ==

Red Teaming



Notice

- This demonstration is strictly for informational purposes only.
- Do not use any of the techniques for malicious or illegal activities.
- The presenter and host organization disclaim all liability.
- Attendees are solely responsible for their own actions.

How can adversaries exploit LoRa?

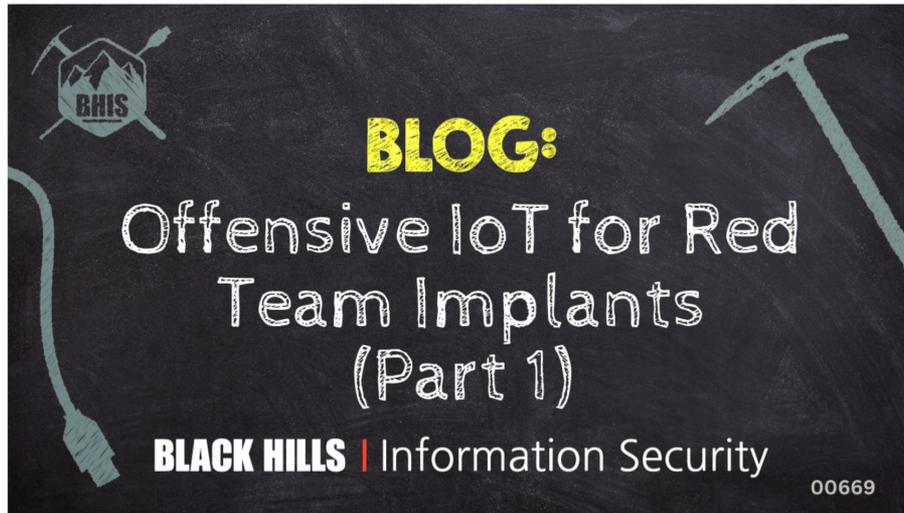
Enablers:

1. Long range
2. Low power
3. Public mesh

Offensive IoT for Red Team Implants – Part 1



| [Tim Fowler](#)



Meet LoKey

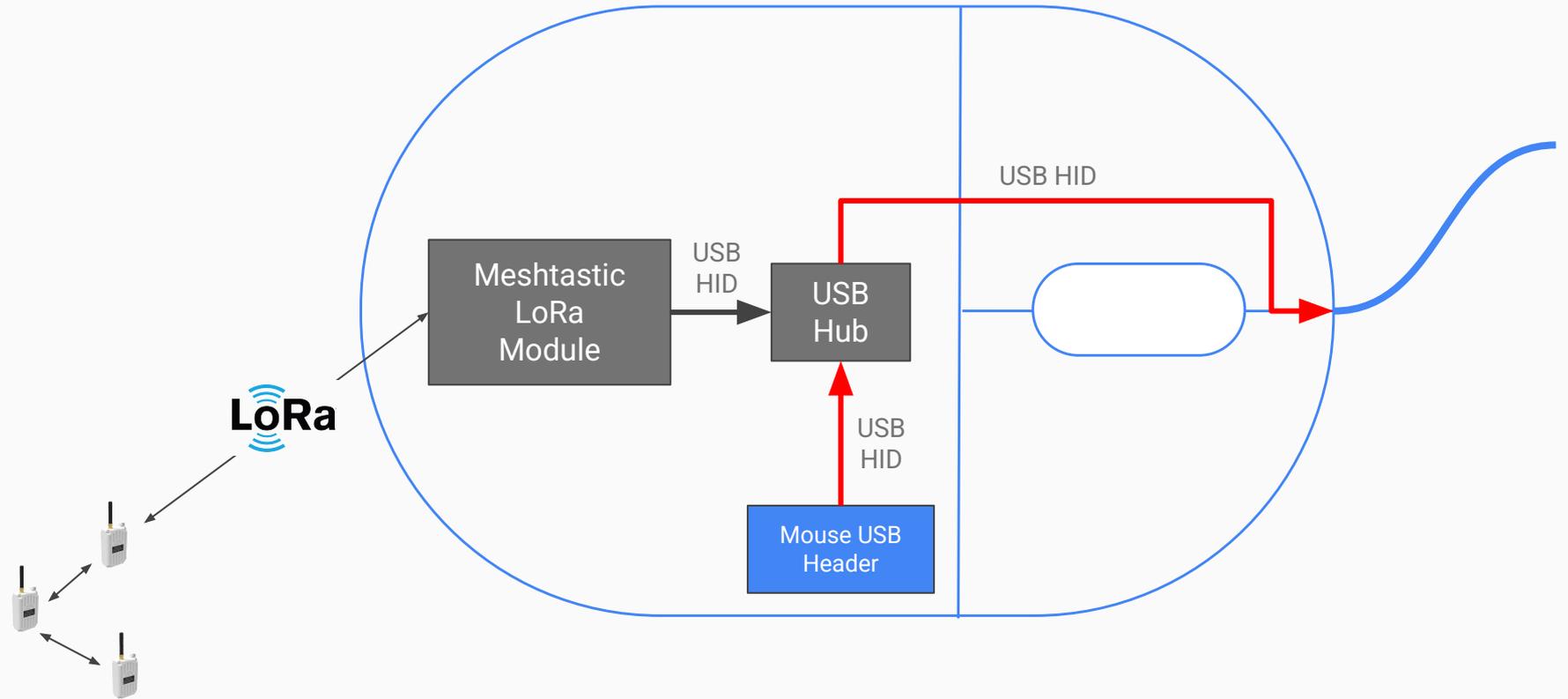
A LoRa-based Implant in an off-the-shelf wired USB mouse.

Emulates a keyboard with Ducky Script™ compatible commands.

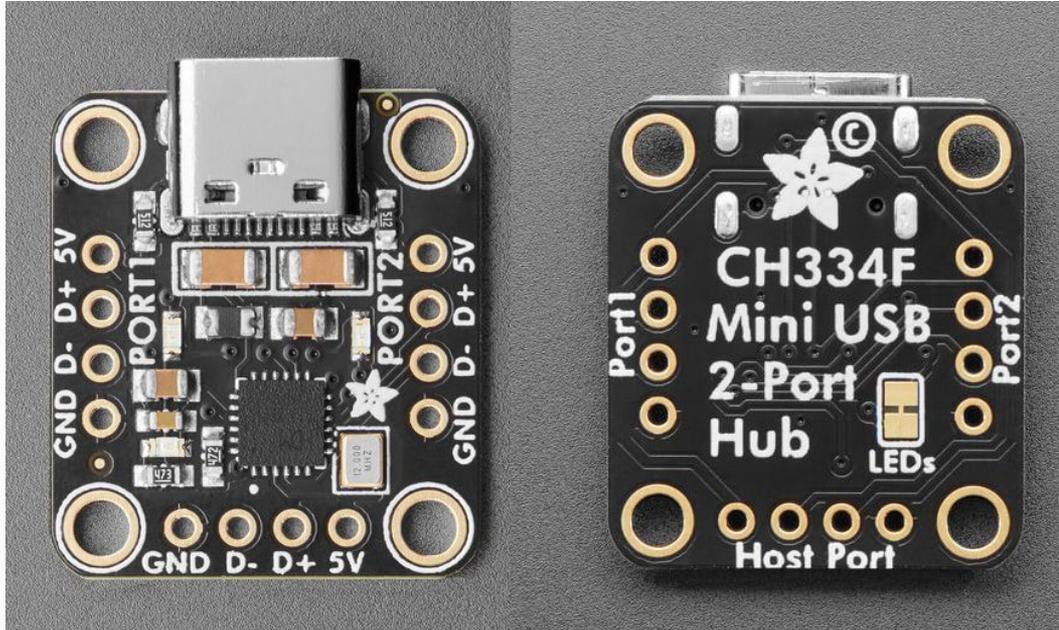
Retains full mouse functionality!



Block Diagram of the Implant

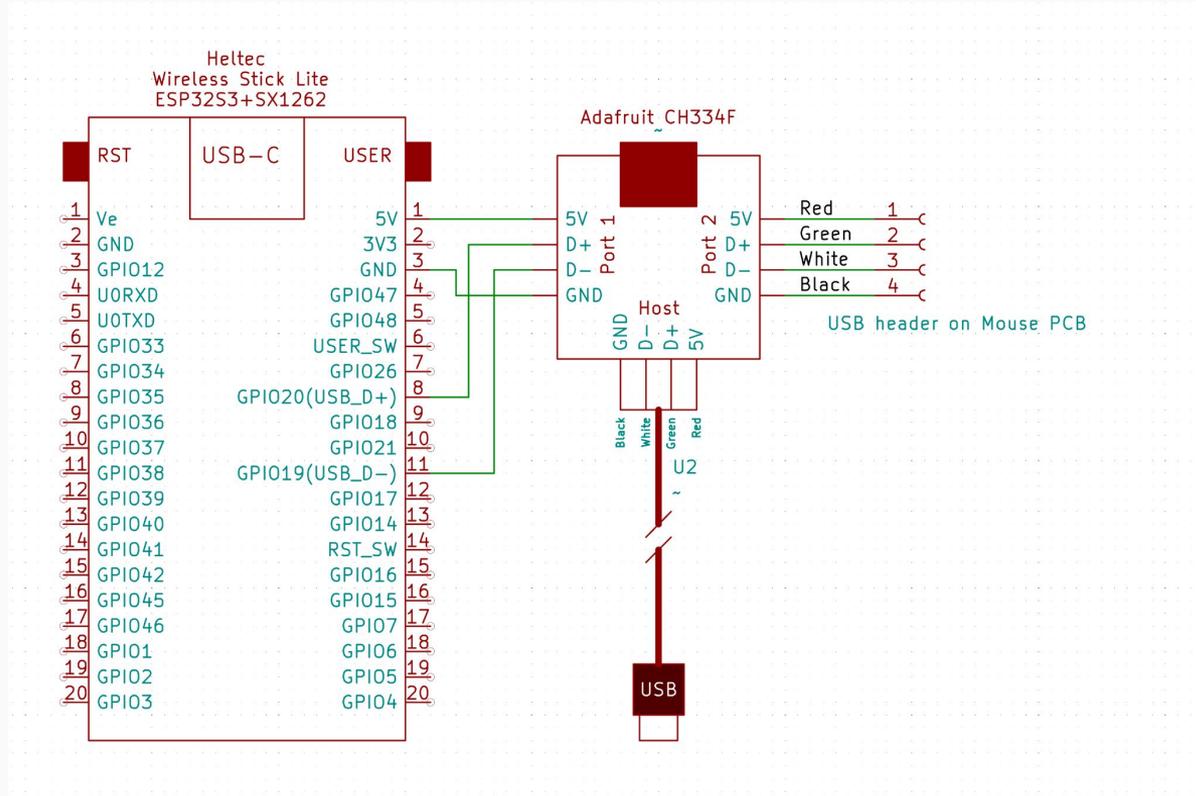


USB Hub Module - Adafruit CH334F Mini USB Hub Breakout

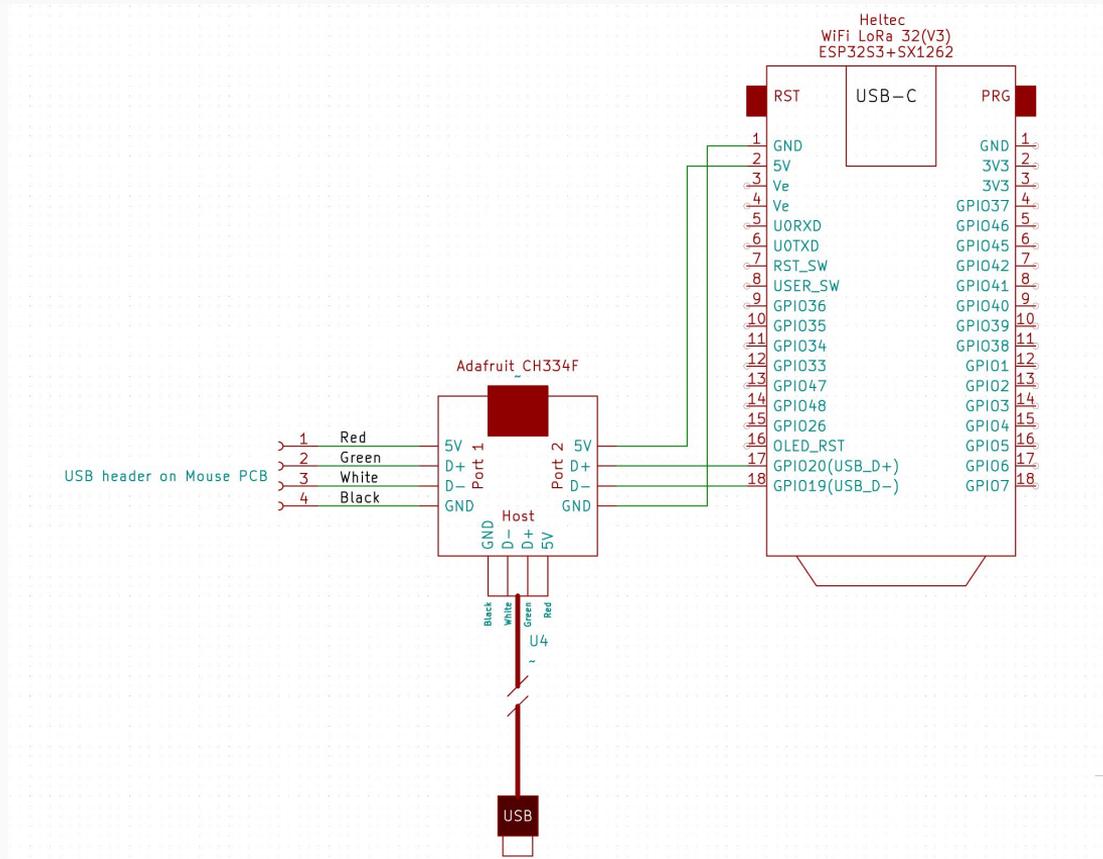


- Two downstream ports
- USB-C or pins for the host port

Schematic for the Heltec Wireless Stick Lite (No OLED)



Schematic for the Heltec WiFi LoRa 32(V3) module with OLED

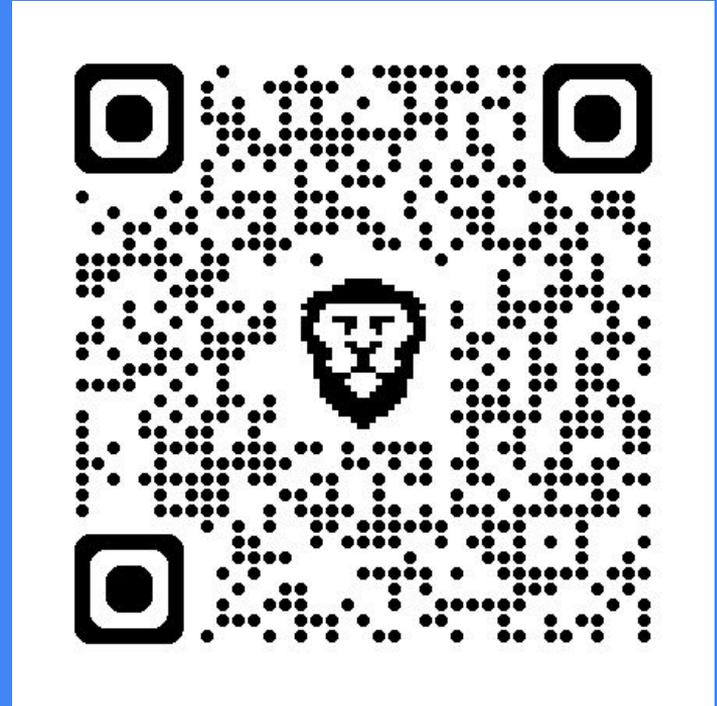


Software

Fork of official firmware

2 files added

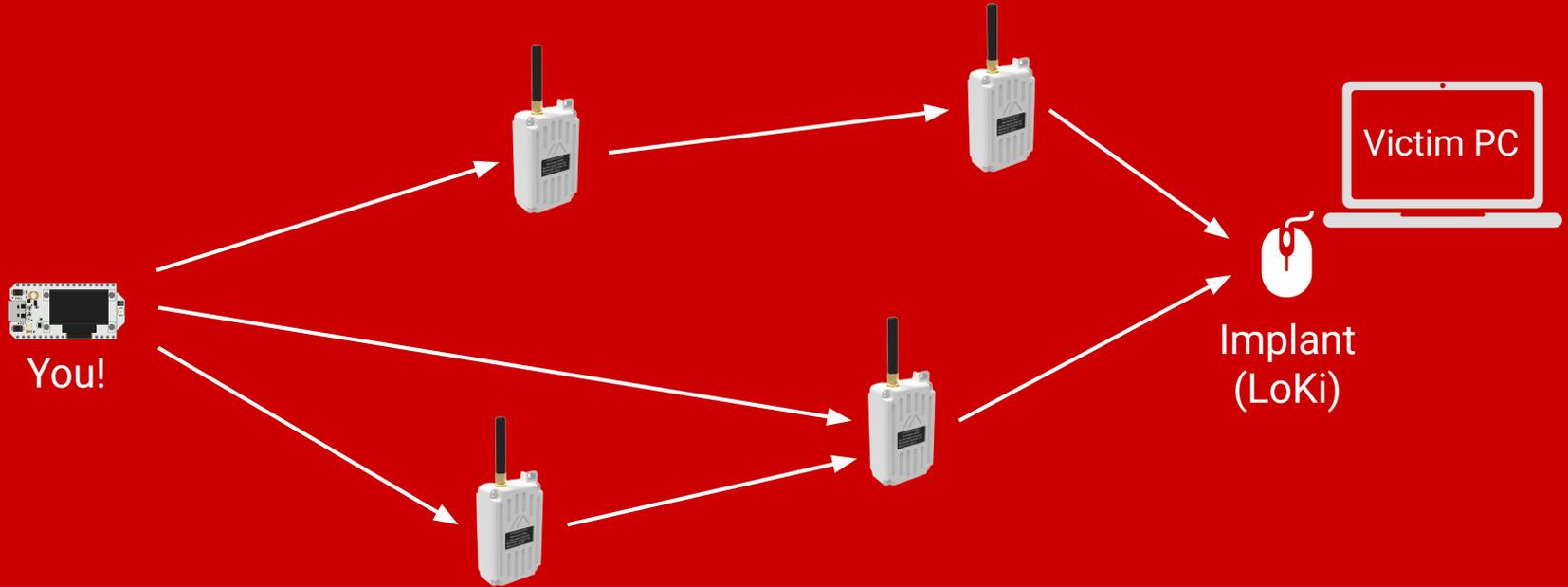
2 files modified



- Meshtastic module that converts LoRa text messages into USB HID keyboard input
- Implements a USB Rubber Ducky–style scripting language with commands: STRING, STRINGLN, ENTER, TAB, BACKSPACE, SPACE, GUI, ALT, CTRL, DELAY, and arrow keys
- Only processes direct messages (ignores broadcasts) and executes explicit LoKey commands, preventing accidental execution of plain text
- Initializes automatically at device boot, ready to receive commands without manual setup. Includes PING/PONG mechanism to signal readiness of USB host.
- Supports single modifiers (GUI/ALT/CTRL + key), dual modifiers (CTRL+SHIFT+key), function keys (F1–F12), and special keys with proper timing delays
- Multi-Line Script Execution – Parses and executes multi-line LoKey scripts line-by-line with automatic trimming and command separation

- Defines HELTEC_V3_HID
-
- Integrate HID Keyboard into the Meshtastic module system by instantiating it in `setupModules()` alongside other core modules
 - Module initializes only when HELTEC_V3_HID is defined, keeping it disabled for other boards and hardware variants

Today's Demo Scenario



Building your own Meshtastic firmware image

Pre-requisites

- Install git
- Install VSCode / Cursor / Windsurf / ...
- Clone the meshtastic firmware repo

1

Install the PlatformIO IDE extension

The screenshot displays the VS Code Extensions Marketplace interface. On the left, a list of extensions is shown, with 'PlatformIO IDE' selected. The main panel shows the details for the 'PlatformIO IDE' extension, including its logo, name, version (3.3.4), and a 5-star rating (3933 reviews). The extension is described as 'Your Gateway to Embedded Software Development Excellence: CMSIS, ESP-IDF, FreeRTOS, libOpenCM3, mbed OS, ...'. The 'Features' section lists several capabilities: cross-platform code builder, debugging, unit testing, static code analysis, remote development, C/C++ intelligent code completion, C/C++ smart code linter, and a library manager. The 'Marketplace' section shows the identifier 'platformio.platformio-ide' and the version '3.3.4'. The 'Categories' section lists 'Programming Languages', 'Linters', 'Debuggers', and 'Testing'. The 'Resources' section includes links to the repository, issues, license, PlatformIO website, and marketplace.

PlatformIO IDE
PlatformIO platformio.org | 6,041,027 | ★★★★★ (3933)
Your Gateway to Embedded Software Development Excellence: CMSIS, ESP-IDF, FreeRTOS, libOpenCM3, mbed OS, ...

PlatformIO IDE for VSCode
PlatformIO: Your Gateway to Embedded Software Development Excellence.

Unlock the true potential of embedded software development with PlatformIO's collaborative ecosystem, embracing declarative principles, test-driven methodologies, and modern toolchains for unrivaled success.

- Open source, maximum permissive Apache 2.0 license
- Cross-platform IDE and Unified Debugger
- Static Code Analyzer and Remote Unit Testing
- Multi-platform and Multi-architecture Build System
- Firmware File Explorer and Memory Inspection.

Platforms: Atmel AVR, Atmel SAM, Espressif 32, Espressif 8266, Freescale Kinetis, Infineon XMC, Intel ARC32, Intel MCS-51 (8051), Kendryte K210, Lattice iCE40, Maxim 32, Microchip PIC32, Nordic nRF51, Nordic nRF52, NXP LPC, RISC-V, Silicon Labs EFM32, ST STM32, ST STM8, Teensy, TI MSP430, TI Tiva, WIZNet W7500

Frameworks: Arduino, CMSIS, ESP-IDF, ESP8266 RTOS SDK, Freedom E SDK, Kendryte Standalone SDK, Kendryte FreeRTOS SDK, libOpenCM3, mbed, PULP OS, SPL, STM32Cube, WiringPi, Zephyr RTOS

Features

- Cross-platform code builder without external dependencies to a system software:
 - 1000+ embedded boards
 - 40+ development platforms
 - 20+ frameworks
- Debugging
- Unit Testing
- Static Code Analysis
- Remote Development
- C/C++ Intelligent Code Completion
- C/C++ Smart Code Linter for rapid professional development
- Library Manager for the thousands of popular libraries

Marketplace

Identifier: platformio.platformio-ide
Version: 3.3.4
Published: 8 years ago
Last Released: 10 months ago

Categories

Programming Languages
Linters | Debuggers | Testing

Resources

Repository
Issues
License
PlatformIO
Marketplace

1...

This may take a few minutes to complete.

The screenshot shows the VS Code interface with the Explorer view on the left displaying a project structure under 'FIRMWARE'. The main editor shows the 'platformio.ini' file with the following content:

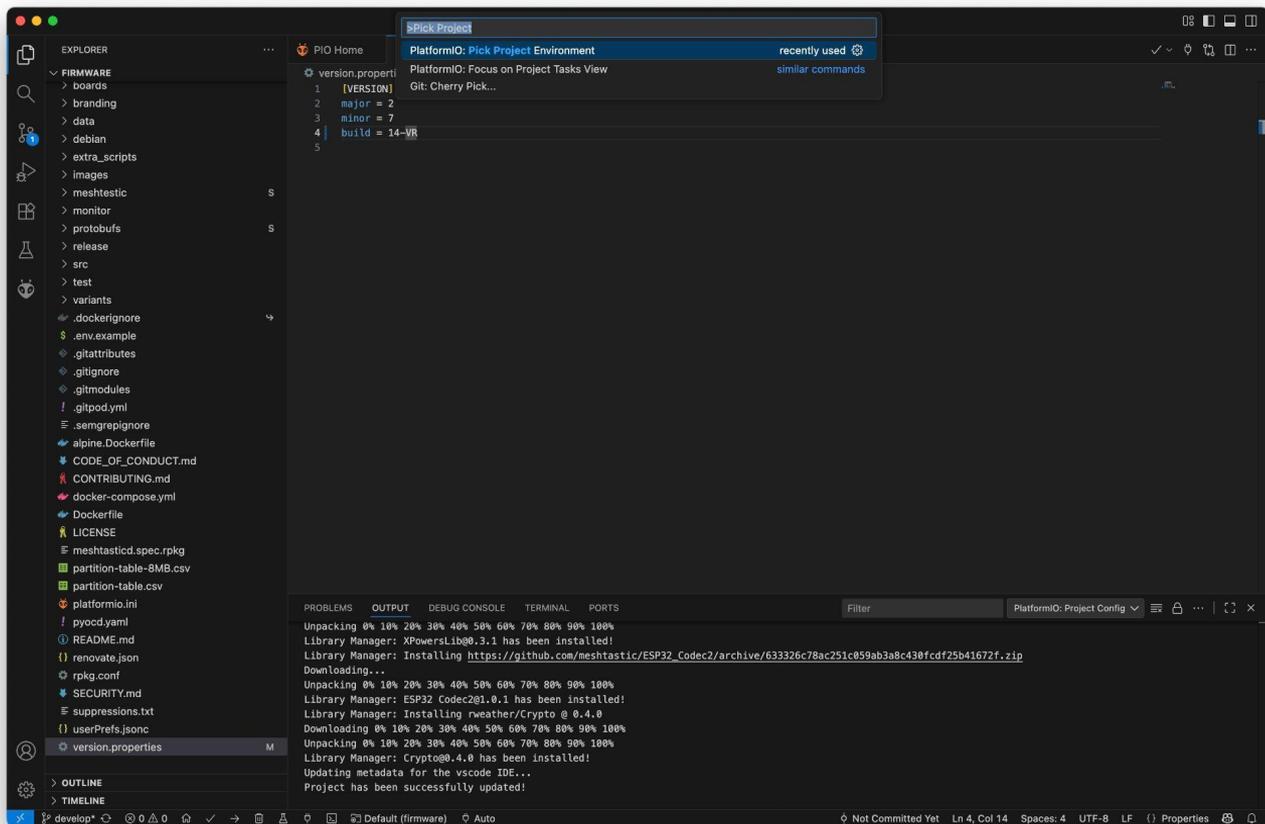
```
1 PlatformIO Project Configuration File
2 ; https://docs.platformio.org/page/projectconf.html
3
4 [platformio]
5 default_envs = tbeam
6
7 extra_configs =
8   arch/*.ini
9   variants/*/platformio.ini
10  variants/*/diy/*/platformio.ini
11  src/graphics/niche/InkHUD/PlatformioConfig.ini
12
13 description = Meshtastic
14
15 [env]
16 test_build_src = true
17 extra_scripts = bin/platformio-custom.py
18 ; note: we add src to our include search path so that lmic_project_config can override
19 ; note: TINYGPS_OPTION_NO_CUSTOM_FIELDS is VERY important. We don't use custom fields and somewhere in that pile
20 ; of code is a heap corruption bug!
21 ; FIXME: fix lib/BluetoothOTA dependency back on src/ so we can remove -Isrc
22 ; The Radiolib stuff will speed up building considerably. Exclud all the stuff we dont need.
23 build_flags = -Wno-missing-field-initializers
24
25 -Wno-format
26 -Isrc -Isrc/mesh -Isrc/mesh/generated -Isrc/gps -Isrc/buzz -Wl,-Map,"${platformio.build_dir}"/output.map
27 -DUSE_THREAD_NAMES
28 -DTINYGPS_OPTION_NO_CUSTOM_FIELDS
29 -DPB_ENABLE_MALLOC=1
30 -DRADIOLIB_EXCLUDE_CC1101=1
31 -DRADIOLIB_EXCLUDE_NRF24=1
32 -DRADIOLIB_EXCLUDE_RF69=1
33 -DRADIOLIB_EXCLUDE_SX1231=1
34 -DRADIOLIB_EXCLUDE_SX1233=1
35 -DRADIOLIB_EXCLUDE_SI4433=1
36 -DRADIOLIB_EXCLUDE_RFM2X=1
37 -DRADIOLIB_EXCLUDE_AFSK=1
38 -DRADIOLIB_EXCLUDE_BELL=1
39 -DRADIOLIB_EXCLUDE_HELLSCHREIBER=1
40 -DRADIOLIB_EXCLUDE_MORSE=1
41 -DRADIOLIB_EXCLUDE_RTTY=1
42 -DRADIOLIB_EXCLUDE_SSTV=1
43 -DRADIOLIB_EXCLUDE_AX25=1
44 -DRADIOLIB_EXCLUDE_DIRECT_RECEIVE=1
45 -DRADIOLIB_EXCLUDE_BELL=1
46 -DRADIOLIB_EXCLUDE_PAGER=1
47 -DRADIOLIB_EXCLUDE_FSK4=1
48 -DRADIOLIB_EXCLUDE_APRS=1
49 -DRADIOLIB_EXCLUDE_LORAWAN=1
50 -DMESHTASTIC_EXCLUDE_DROPZONE=1
```

A PlatformIO IDE configuration dialog is visible in the bottom right corner, showing 'PlatformIO: Configuring project: 90%' and 'Source: PlatformIO IDE' with a 'Cancel' button.

2

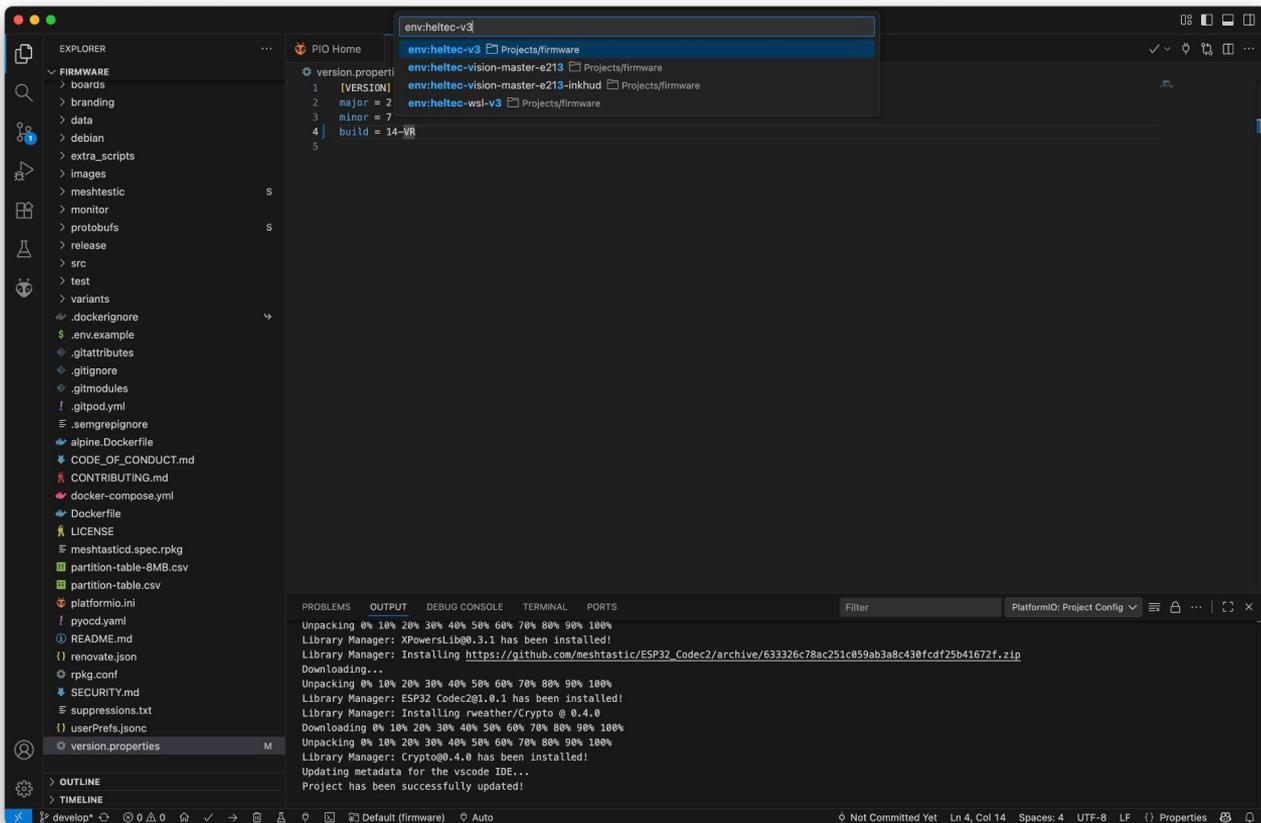
Make a small change in the version.properties file.

From the command palette, select PlatformIO : Pick Project Environment



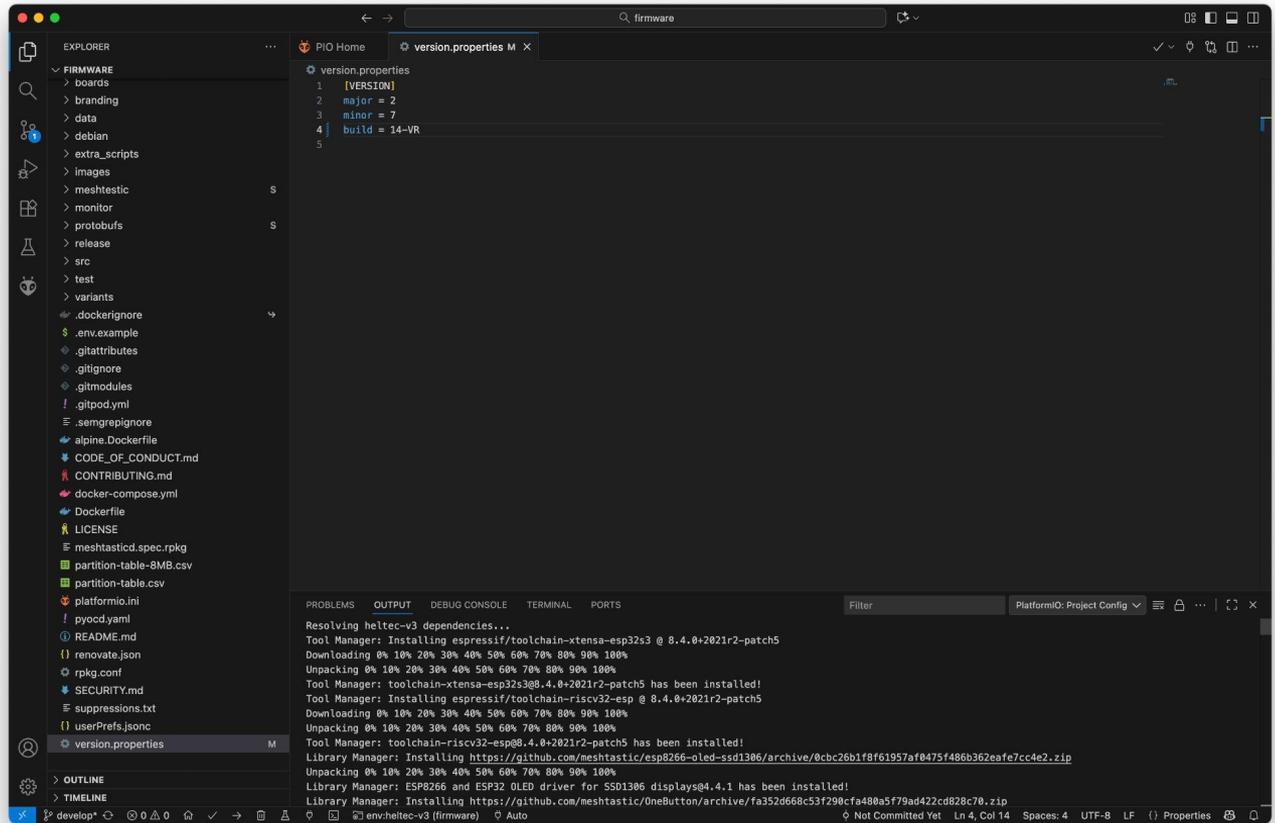
2...

Select Heltec-V3 or your device



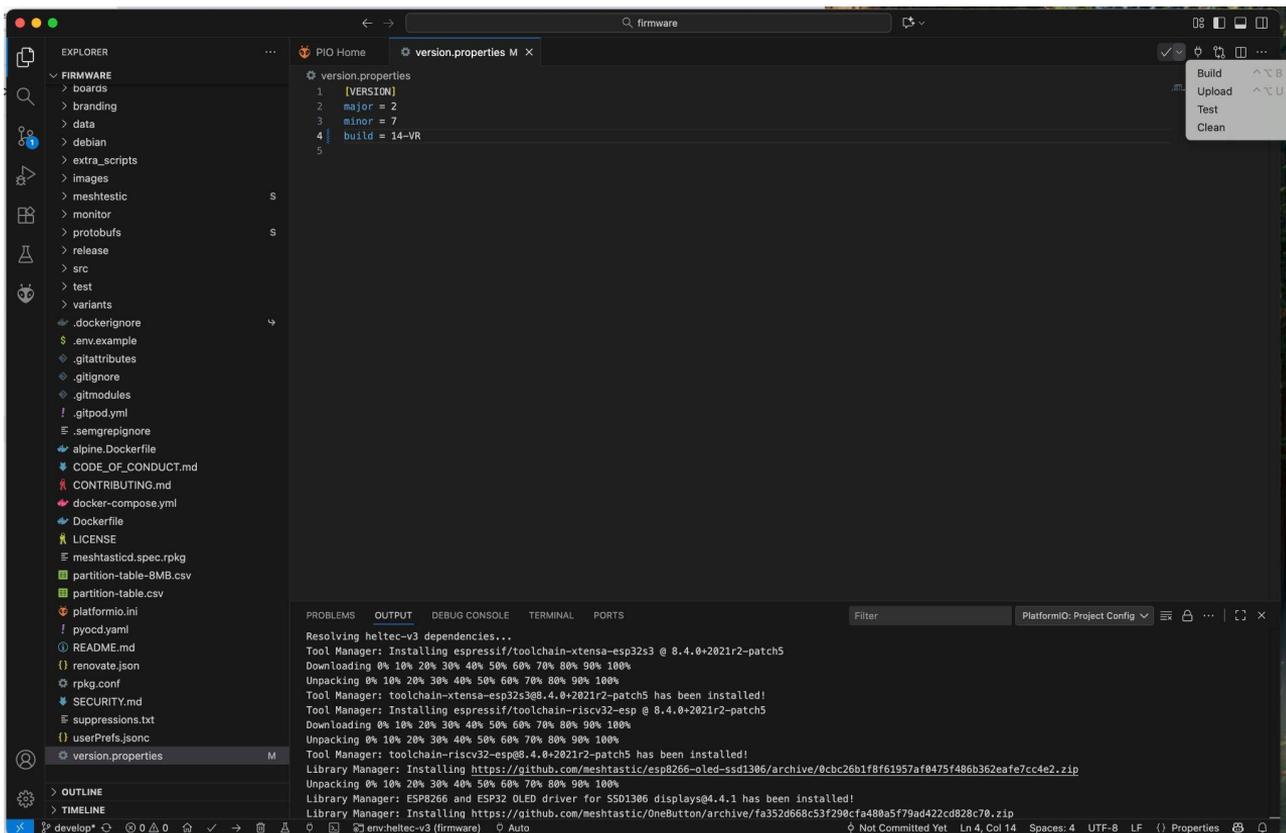
2...

This may take a couple of minutes.



3

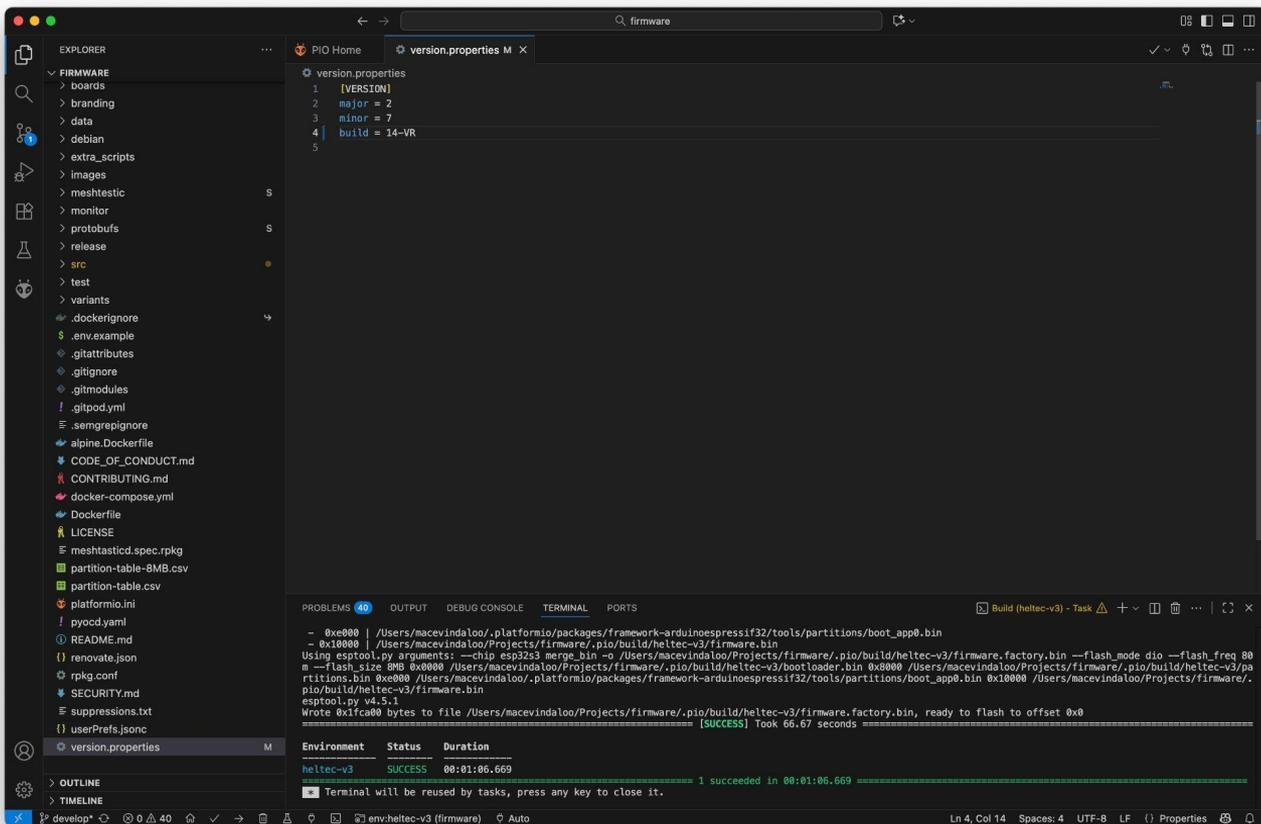
Start the build.



3...

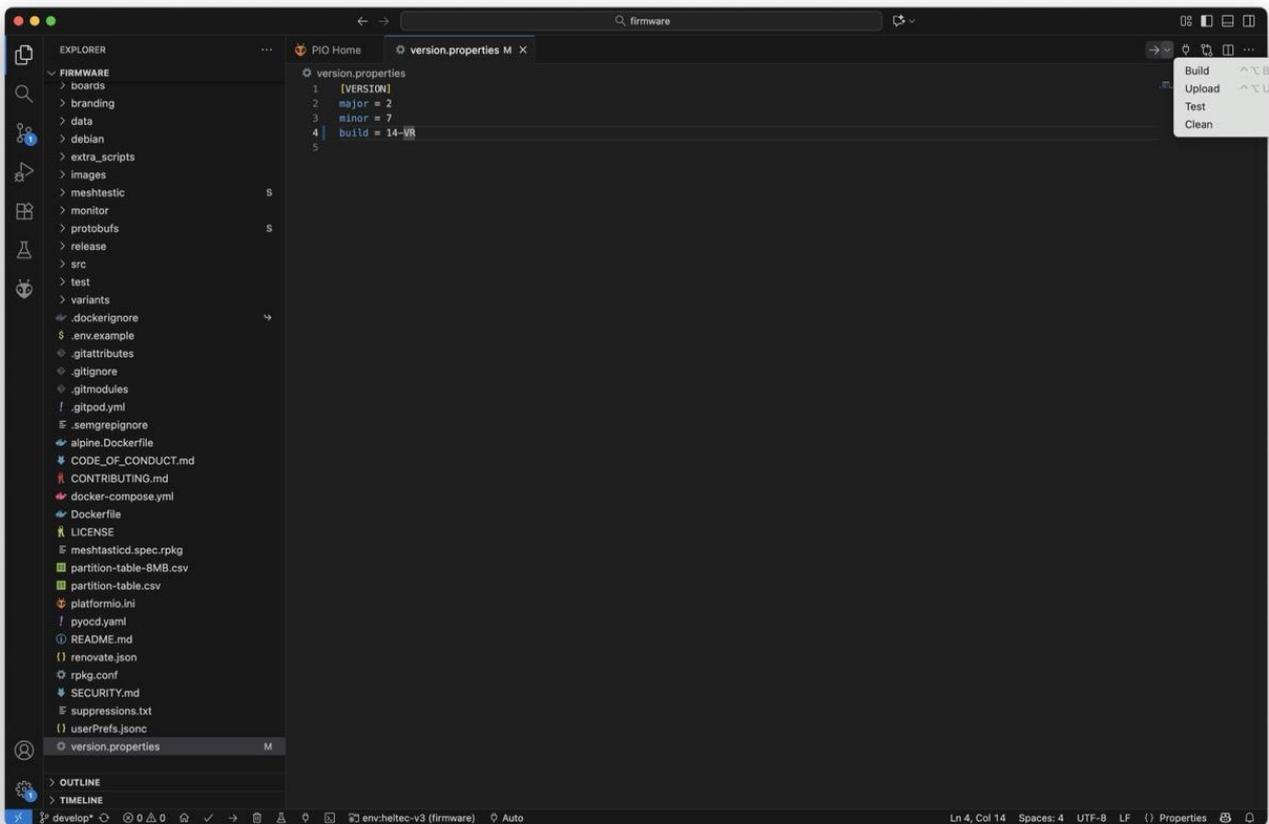
This may take a minute or so.

Congratulations, you have now built your own firmware image!



4

Upload the firmware to your device.



4...

This may take a couple of minutes.

Verify that the display shows your custom version string.

A screenshot of the Visual Studio Code editor. The Explorer sidebar on the left shows a file tree for a project named 'firmware'. The main editor area shows a file named 'version.properties' with the following content:

```
1 | VERSION|
2 | major = 2|
3 | minor = 7|
4 | build = 14-VR|
5 |
```

The bottom panel shows the Terminal output of a firmware update process. It lists various memory addresses and their corresponding progress percentages, such as 'Writing at 0x0015c4f6... (69 %)' and 'Writing at 0x001fa24a... (100 %)'. The process concludes with 'Leaving...' and 'Hard resetting via RTS pin...'. A summary table at the bottom of the terminal shows the update was successful:

Environment	Status	Duration
heltec-v3	SUCCESS	00:00:46.149

Thank you!

venky.raju@gmail.com

