# Table of Contents

## Overview
- Parts
- All-in-One
- Breadboard Parts
- Sending Data using LoRa
- Materials

## Raspberry Pi Wiring
- Raspberry Pi Radio Bonnet Wiring
- Raspberry Pi Breadboard Wiring
- Wiring the Push-Buttons
- Wiring the OLED
- Wiring the RFM Radio Module

## LoRa Raspberry Pi Setup
- Installing CircuitPython Libraries

## Sending Data with LoRa
- RFM9x and CircuitPython
- CircuitPython Transmitter/Receiver Example

## Sending Data over LoRaWAN
- Overview

## TinyLoRa TTN Setup

## Usage
- Code
- Setting up the code for The Things Network
- Usage
- Checking Data on The Things Network Console
- Decoding the Payload
Overview

If you’re building a Raspberry Pi project and need to send data back-and-forth, the WiFi module built into newer Pi’s should be fine for most cases.

However, WiFi does not have a long range and also requires both investment (through your ISP) and setup (routers, access points, network security schemes).

If you’re deploying your project somewhere more remotely, without WiFi or a strong cellular network, like a farm, you'll need another option. Something that can be deployed quickly with no overhead setup and that will transmit/receive over a long distance.

By sacrificing transmit speed and the amount of data which you’re able to send, packet radios can transmit much farther than Bluetooth Low-Energy and WiFi.

This makes using a radio perfect for scenarios where your project would need to periodically transmit sensor and/or device data (such as the Pi’s battery life) over a long distance. The RFM69 module can transmit up to 50m and the RFM9x module can send packets up to 2km over LoRa and even further using LoRaWAN!
Parts

**Raspberry Pi 3 - Model B+ - 1.4GHz Cortex-A53 with 1GB RAM**
The Raspberry Pi 3 Model B is the most popular Raspberry Pi computer made, and the Pi Foundation knows you can always make a good thing better! And what could make the Pi 3...
[https://www.adafruit.com/product/3775](https://www.adafruit.com/product/3775)

**Raspberry Pi Zero - Version 1.3**
At first glance, the Pi Zero isn't much. It just looks like a slimmed down version of the Raspberry Pi we know and love. But when we started to think of the...

All-in-One

The Adafruit LoRa Radio Bonnet for Raspberry Pi sets you up with a RFM95W Radio Module, a 128x32 OLED display and three push-buttons for LoRa and LoRaWAN experiments.

**Adafruit LoRa Radio Bonnet with OLED - RFM95W @ 915MHz**
The latest Raspberry Pi computers come with WiFi and Bluetooth, and now you can add even more radio options with the Adafruit Radio Bonnets! Upgrade your Raspberry Pi with a LoRa /...
[https://www.adafruit.com/product/4074](https://www.adafruit.com/product/4074)
There's also a 433MHz version which uses the amateur or license-free ISM band (ITU "Europe" license-free ISM or ITU "American" amateur with limitations).

Adafruit LoRa Radio Bonnet RFM96W @ 433MHz
The latest Raspberry Pi computers come with WiFi and Bluetooth, and now you can add even more radio options with the Adafruit Radio Bonnets! Upgrade your Raspberry Pi with a LoRa /...
https://www.adafruit.com/product/4075

Breadboard Parts

You'll need a way to access the GPIO (pins) from the Raspberry Pi. The T-Cobbler plus is a plug-and-play breakout which works with almost every new Raspberry Pi variation.

Assembled Pi T-Cobbler Plus - GPIO Breakout
This is the assembled version of the Pi T-Cobbler Plus. It only works with the Raspberry Pi Model Zero, A+, B+, Pi 2, Pi 3 & Pi 4! (Any Pi with 2x20...
https://www.adafruit.com/product/2028

The Pi setup used for this guide is operated as a headless Pi. This means that no monitor is plugged into the Pi and we control it over SSH. We'll add a small display to monitor and interact with the code.
Sending Data using LoRa

The RFM9x LoRa comes in two different frequencies - 868/915MHz and 433MHz. It can broadcast over LoRa (longer range than the RFM69HCW) and LoRaWAN (even longer range to gateways).

There's also a 433MHz version which uses the amateur or license-free ISM band (ITU "Europe" license-free ISM or ITU "American" amateur with limitations).

1 x LoRa Breakout RFM96W
LoRa Radio Module - 433MHz Version

Materials

If you don’t have these parts already, grab them from the Adafruit store:

1 x Tactile Switch Buttons
10-pack of medium-sized momentary switches

Monochrome 128x32 I2C OLED graphic display
These displays are small, only about 1” diagonal, but very readable due to the high contrast of an OLED display. This display is made of 128x32 individual white OLED pixels, each...
https://www.adafruit.com/product/931

Adafruit RFM95W LoRa Radio Transceiver Breakout - 868 or 915 MHz
"You see, wire telegraph is a kind of a very, very long cat. You pull his tail in New York and his head is meowing in Los Angeles. Do you understand this? And...
https://www.adafruit.com/product/3072

There’s also a 433MHz version which uses the amateur or license-free ISM band (ITU "Europe" license-free ISM or ITU "American" amateur with limitations).
## Raspberry Pi Wiring

### Raspberry Pi Radio Bonnet Wiring

The Radio Bonnet for Raspberry Pi sets you up with a RFM9x (or RFM69HCW) Radio Module, a 128x32 OLED display and three push-buttons.

If you have this bonnet, everything is already "wired up" for you - move on to setting up your Raspberry Pi.

### Raspberry Pi Breadboard Wiring

#### Wiring the Push-Buttons

We'll start by wiring three buttons. These will be the physical interface between the Raspberry Pi and the Radio - we'll want to control the Radio by pressing some buttons.
Make the following connections between the Pi and the three push-buttons:

- Pi GPIO #5 to Button A (orange wire)
- Pi GPIO #6 to Button B (blue wire)
- Pi GPIO #12 to Button C (green wire)
- Pi 3.3V to Power Rail (red wire)
- Pi GND to Ground Rail (black wire)

Wiring the OLED

- Pi 3.3V to OLED VIN
- Pi GND to OLED GND
- Pi SCL to OLED SCL
- Pi SDA to OLED SDA
- Pi GPIO4 to OLED RST
Wiring the RFM Radio Module

Make the following connections between the Raspberry Pi and the RFM Radio Module:

- Vin to Raspberry Pi 3.3V
- GND to Raspberry Pi Ground
- RFM G0 to Raspberry Pi GPIO #5
- RFM RST to Raspberry Pi GPIO #25
- RFM CLK to Raspberry Pi SCK
- RFM MISO to Raspberry Pi MISO
- RFM MOSI to Raspberry Pi MOSI
- RFM CS to Raspberry Pi CE1

Next, let's move on to installing the necessary files for this project.

LoRa Raspberry Pi Setup

This guide assumes that you've gotten your Raspberry Pi up and running, and have CircuitPython installed.

- If you have not done this yet, visit the installation guide here and come back when you're set up.

Installing CircuitPython Libraries

We're running CircuitPython on the Raspberry Pi, installing the libraries for radio communication is simple.

To install the library for the display, enter the following into the terminal:
sudo pip3 install adafruit-circuitpython-ssd1306

You'll also need to install the framebuf module in order to write to the display.

dsudp pip3 install adafruit-circuitpython-framebuf

To install the library for the RFM9x Module, enter the following into the terminal:

dsudp pip3 install adafruit-circuitpython-rfm9x

You'll also want to download the font file, font5x8.bin, and copy it into the same directory as the script

wget https://github.com/adafruit/Adafruit_CircuitPython_framebuf/raw/main/examples/font5x8.bin

Click to download font5x8.bin

Make sure the font file is 1282 bytes long, if not something went wrong with the download

RFM9x Connection Test!

Do not use this if you have a RFM69 Radio

The following code is for checking if the RFM9x radio is set up for transmitting and receiving. Save the code on your Pi (save this to a location you can remember, like your Desktop or Downloads folder) as rfm9x_check.py.

# SPDX-FileCopyrightText: 2018 Brent Rubell for Adafruit Industries
#
# SPDX-License-Identifier: MIT

###
Wiring Check, Pi Radio w/RFM9x

Author: Brent Rubell for Adafruit Industries

```python
import time
import busio
from digitalio import DigitalInOut, Direction, Pull
```
import board
# Import the SSD1306 module.
import adafruit_ssd1306
# Import the RFM9x radio module.
import adafruit_rfm9x

# Button A
btnA = DigitalInOut(board.D5)
btnA.direction = Direction.INPUT
btnA.pull = Pull.UP

# Button B
btnB = DigitalInOut(board.D6)
btnB.direction = Direction.INPUT
btnB.pull = Pull.UP

# Button C
btnC = DigitalInOut(board.D12)
btnC.direction = Direction.INPUT
btnC.pull = Pull.UP

# Create the I2C interface.
i2c = busio.I2C(board.SCL, board.SDA)

# 128x32 OLED Display
reset_pin = DigitalInOut(board.D4)
display = adafruit_ssd1306.SSD1306_I2C(128, 32, i2c, reset=reset_pin)
# Clear the display.
display.fill(0)
display.show()
width = display.width
display.fill(0)
display.show()

# Configure RFM9x LoRa Radio
CS = DigitalInOut(board.CE1)
RESET = DigitalInOut(board.D25)
spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)

while True:
    # Clear the image
display.fill(0)

    # Attempt to set up the RFM9x Module
    try:
        rfm9x = adafruit_rfm9x.RFM9x(spi, CS, RESET, 915.0)
display.text('RFM9x: Detected', 0, 0, 1)
    except RuntimeError as error:
        # Thrown on version mismatch
        display.text('RFM9x: ERROR', 0, 0, 1)
        print('RFM9x Error: ', error)

    # Check buttons
    if not btnA.value:
        # Button A Pressed
        display.text('Ada', width-85, height-7, 1)
display.show()
time.sleep(0.1)

    if not btnB.value:
        # Button B Pressed
        display.text('Fruit', width-75, height-7, 1)
display.show()
time.sleep(0.1)

    if not btnC.value:
        # Button C Pressed
        display.text('Radio', width-65, height-7, 1)
display.show()
time.sleep(0.1)
To use the code, copy and paste the following command into your terminal:

```
python3 rfm9x_check.py
```

Now to check the setup:

If the RFM9x/RFM69 is detected, the OLED will display Detected. You can test the buttons by pressing them.

If the wiring of the radio module is incorrect - the display will show ERROR. Check over your wiring on the Wiring Page and re-run the test. You may also need to ensure the correct CircuitPython library is installed for the module.

If the OLED does not turn on - first check that it is wired correctly. Then, make sure you enabled I2C from `raspi-config` and installed the required libraries (`adafruit-circuitpython-framebuf` and `adafruit-circuitpython-ssd1306`).

With everything working, let's move on to using the radio.

**Sending Data with LoRa**

Do not use this if you have a RFM69 Packet Radio, its for RFM9X radio's only!
The RFM9x is a more expensive module than the RFM69, but it has a trick - LoRa. LoRa is a spread spectrum modulation technique patented by Semtech. It allows your packets to be sent over farther distances (a few km in a city like New York and around 40km in a rural area).

It's also a low power protocol with batteries lasting in the year range, instead of a few days with WiFi. This is made possible by only powering up the LoRa radio when packets are being sent, instead of keeping the radio always-on (like the WiFi radio on your cell phone).

When building a project which uses LoRa, keep in mind that only a few hundred bytes per-transmission can be sent, and that each transmission will cause the battery life to decrease.

If a LoRa project involves a lot of sensors, expect to cram a lot of data into a small packet which is transmitted infrequently.

**RFM9x and CircuitPython**

It's easy to use the RFM9x LoRa radio with CircuitPython and the Adafruit CircuitPython RFM9x module. This module allows you to easily write Python code that sends and receives packets of data with the radio.

Be careful to note this library is for the RFM9x (RFM95/96/97/98) Radio only and will not work with the RFM69.

To demonstrate how the RFM9x LoRa module sends packets, we'll build an example where we send and receive data between two radios.
CircuitPython Transmitter/Receiver Example

Below is an example of using the RFM9x to transmit, or receive from, another RFM9x radio. Save this as radio_rfm9x.py on your Raspberry Pi:

```python
# SPDX-FileCopyrightText: 2018 Brent Rubell for Adafruit Industries
# SPDX-License-Identifier: MIT

Example for using the RFM9x Radio with Raspberry Pi.

Author: Brent Rubell for Adafruit Industries

# Import Python System Libraries
import time

# Import Blinka Libraries
import busio
from digitalio import DigitalInOut, Direction, Pull
import board

# Import the SSD1306 module.
import adafruit_ssd1306

# Import RFM9x
import adafruit_rfm9x

# Button A
btnA = DigitalInOut(board.D5)
btnA.direction = Direction.INPUT
btnA.pull = Pull.UP

# Button B
btnB = DigitalInOut(board.D6)
btnB.direction = Direction.INPUT
btnB.pull = Pull.UP

# Button C
btnC = DigitalInOut(board.D12)
btnC.direction = Direction.INPUT
btnC.pull = Pull.UP

# Create the I2C interface.
i2c = busio.I2C(board.SCL, board.SDA)

# 128x32 OLED Display
reset_pin = DigitalInOut(board.D4)
display = adafruit_ssd1306.SSD1306_I2C(128, 32, i2c, reset=reset_pin)

display.fill(0)
display.show()

width = display.width
height = display.height

# Configure LoRa Radio
CS = DigitalInOut(board.CE1)
RESET = DigitalInOut(board.D25)
spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
rfm9x = adafruit_rfm9x.RFM9x(spi, CS, RESET, 915.0)
rfm9x.tx_power = 23
prev_packet = None

while True:
    packet = None
    # draw a box to clear the image
display.fill(0)
```

©Adafruit Industries
display.text('RasPi LoRa', 35, 0, 1)

# check for packet rx
packet = rfm9x.receive()
if packet is None:
    display.show()
    display.text('- Waiting for PKT -', 15, 20, 1)
else:
    # Display the packet text and rssi
    display.fill(0)
    prev_packet = packet
    packet_text = str(prev_packet, "utf-8")
    display.text('RX: ', 0, 0, 1)
    display.text(packet_text, 25, 0, 1)
    time.sleep(1)

    if not btnA.value:
        # Send Button A
        display.fill(0)
        button_a_data = bytes("Button A!\r\n","utf-8")
        rfm9x.send(button_a_data)
        display.text('Sent Button A!', 25, 15, 1)
    elif not btnB.value:
        # Send Button B
        display.fill(0)
        button_b_data = bytes("Button B!\r\n","utf-8")
        rfm9x.send(button_b_data)
        display.text('Sent Button B!', 25, 15, 1)
    elif not btnC.value:
        # Send Button C
        display.fill(0)
        button_c_data = bytes("Button C!\r\n","utf-8")
        rfm9x.send(button_c_data)
        display.text('Sent Button C!', 25, 15, 1)

display.show()
time.sleep(0.1)

You'll also want to download the font file, `font5x8.bin`, and copy it into the same directory as the script:

`font5x8.bin`

To run the example, enter the following into the terminal:

`python3 radio_rfm9x.py`

Both of the radios will listen for a new incoming packet. When they receive a new packet, they'll print the text from the packet to the display (and to the terminal).

Press any of the three buttons to send data between radios. Pressing button a will send a packet with data 'Button A!' to the other radio, and so on!
Sending Data over LoRaWAN

The LoRaWAN part of this Guide is deprecated and is no longer possible! Single-channel packet forwarders no longer work after the Things Network migration to The Things Stack v3. For more information about this decision, visit: https://www.thethingsnetwork.org/forum/t/single-channel-packet-forwarders-scpf-are-deprecated-and-not-supported/31117

Overview

LoRaWAN is a Low Power Wide Area Network (LPWAN) specification which allows connection of LoRa devices to the internet. LoRaWAN differs from LoRa by the inclusion of a physical layer.

This may seem confusing, but it's really no different from other types of networks (like cellular), except you can use a gateway (like a cell tower) that you own, or someone else owns and operates, to connect your device to the Internet.

One gateway can cover a large area, depending on the obstructions (a gateway operated in NYC would have a range of a few hundred kilometers vs thousands of kilometers in the countryside)
We're going to be connecting our Raspberry Pi to The Things Network, a LoRaWAN network made up of volunteers operating their own gateways, using CircuitPython.

Want to learn more about The Things Network and CircuitPython? There's a full guide written on the Adafruit Learning System which dives into deeper detail.

TinyLoRa TTN Setup

This guide is deprecated and is no longer possible! Single-channel packet forwarders no longer work after the Things Network migration to The Things Stack v3. For more information about this decision, visit: https://www.thethingsnetwork.org/forum/t/single-channel-packet-forwarders-scpf-are-deprecated-and-not-supported/31117

Feathers running TinyLoRa require a different setup process. Be sure to follow these steps closely if you haven't set up an ABP device before.

Before your Feather can communicate with The Things Network, you'll need to create an application.

First, we're going to register an account with TTN. Navigate to their account registration page to set up an account.

Once logged in, navigate to the The Things Network Console. This page is where you can register applications and add new devices or gateways. Click Applications
Click Add Application.

Fill out an Application ID to identify the application, and a description of what the application is. We set our Handler Registration to match our region, us-west. If you're not located in the U.S., TTN provides multiple regions for handler registration.

Once created, you'll be directed to the Application Overview. From here, you can add devices, view data coming into (and out of) the application, add integrations for external services, and more. We'll come back to this section later in the guide.

Click Register Device
On the Register Device Page, The Device ID should be a unique string to identify the device.

The Device EUI is the unique identifier which came in the same bag as your Radiofruit Feather. We'll pad the middle of the string with four zeroes. Lost your key? No worries - you can also click the "mix" icon to switch it to auto-generate.

The App Key will be randomly generated for you by TTN. Select the App EUI (used to identify the application) from the list.

Next, we're going to switch the device settings from Over-the-Air-Activation to Activation-by-Personalization. From the Device Overview, click Settings
On the settings screen, change the Activation Method from OTAA to ABP.

Then, switch the Frame Counter Width from 32b to 16b and disable frame counter checks. TTN will display a warning, ignore it, and click Save.

---

Now that the application is set up, and the device is registered to the application, let's move on to setting up Arduino with TinyLoRa.

Why are we disabling Frame Counter Checks if The Things Network Console doesn't recommend it?

Disabling frame counter checks allows you to transmit data to The Things Network without requiring a match between your device's frame counter and the console's frame counter.

If you're making a project and doing a lot of prototyping/iteration to the code, disabling these checks is okay as it'll let you reset the device and not re-register it to the application (it'll also prevent counter overflows).
If you're deploying a project, you'll want to re-activate the frame counter for security purposes. With the frame counter disabled, one could re-transmit the messages sent to TTN using a replay attack.

The Things Network's documentation page has a full explanation about the role of the Frame Counter.

Usage

This guide is deprecated and is no longer possible! Single-channel packet forwarders no longer work after the Things Network migration to The Things Stack v3. For more information about this decision, visit: https://www.thethingsnetwork.org/forum/t/single-channel-packet-forwarders-scpf-are-deprecated-and-not-supported/31117

Do not use this code if you have a RFM69HCW Radio

Now that we've set up our Things Network application and device, we're going to move on to installing TinyLoRa onto our Raspberry Pi. To do this, enter the following into your terminal to install the library system-wide:

```
sudo pip3 install adafruit-circuitpython-tinylora
```

Code

Unlike sending data to another device, we're going to be sending data from our device (the Pi) to a gateway (check the gateway map to find one near you).

While we don't have any sensors hooked up to our radio, we'll send the Raspberry Pi's CPU utilization to The Things Network.

Below is an example of using TinyLoRa to send data to The Things Network. Save this as radio_lorawan.py on your Raspberry Pi.

```python
# SPDX-FileCopyrightText: 2018 Brent Rubell for Adafruit Industries
# SPDX-License-Identifier: MIT

Example for using the RFM9x Radio with Raspberry Pi and LoRaWAN

Author: Brent Rubell for Adafruit Industries
```
```python
import threading
import time
import subprocess
import busio
from digitalio import DigitalInOut, Direction, Pull
import board

# Import the SSD1306 module.
import adafruit_ssd1306

# Import Adafruit TinyLoRa
from adafruit_tinylora.adafruit_tinylora import TTN, TinyLoRa

# Button A
btnA = DigitalInOut(board.D5)
btnA.direction = Direction.INPUT
btnA.pull = Pull.UP

# Button B
btnB = DigitalInOut(board.D6)
btnB.direction = Direction.INPUT
btnB.pull = Pull.UP

# Button C
btnC = DigitalInOut(board.D12)
btnC.direction = Direction.INPUT
btnC.pull = Pull.UP

# Create the I2C interface.
reset_pin = DigitalInOut(board.D4)
display = adafruit_ssd1306.SSD1306_I2C(128, 32, i2c, reset=reset_pin)

# 128x32 OLED Display

# Clear the display.
display.fill(0)
display.show()

width = display.width
height = display.height

# TinyLoRa Configuration
spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
cs = DigitalInOut(board.CE1)
irq = DigitalInOut(board.D22)
rst = DigitalInOut(board.D25)

devaddr = bytearray([0x00, 0x00, 0x00, 0x00])
nwkey = bytearray([0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00])
app = bytearray([0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00])

ttn_config = TTN(devaddr, nwkey, app, country='US')
lora = TinyLoRa(spi, cs, irq, rst, ttn_config)

# 2b array to store sensor data
data_pkt = bytearray(2)

# time to delay periodic packet sends (in seconds)
data_pkt_delay = 5.0

def send_pi_data_periodic():
    threading.Timer(data_pkt_delay, send_pi_data_periodic).start()
    print("Sending periodic data...")
    send_pi_data(CPU)
    print('CPU:', CPU)
```
def send_pi_data(data):
    # Encode float as int
    data = int(data * 100)
    # Encode payload as bytes
    data_pkt[0] = (data >> 8) & 0xff
    data_pkt[1] = data & 0xff
    # Send data packet
    lora.send_data(data_pkt, len(data_pkt), lora.frame_counter)
    lora.frame_counter += 1
    display.fill(0)
    display.text('Sent Data to TTN!', 15, 15, 1)
    print('Data sent!')
    display.show()
    time.sleep(0.5)

while True:
    packet = None
    # draw a box to clear the image
    display.fill(0)
    display.text('RasPi LoRaWAN', 35, 0, 1)
    # read the raspberry pi cpu load
    cmd = "top -bn1 | grep load | awk '{printf \"%.1f\", $(NF-2)}'"
    CPU = subprocess.check_output(cmd, shell = True)
    CPU = float(CPU)
    if not btnA.value:
        # Send Packet
        send_pi_data(CPU)
    if not btnB.value:
        # Display CPU Load
        display.fill(0)
        display.text('CPU Load %', 45, 0, 1)
        display.text(str(CPU), 60, 15, 1)
        display.show()
        time.sleep(0.1)
    if not btnC.value:
        display.fill(0)
        display.text('* Periodic Mode *', 15, 0, 1)
        display.show()
        time.sleep(0.5)
        send_pi_data_periodic()
    display.show()
    time.sleep(.1)

You'll also want to download the font file, `font5x8.bin`, and copy it into the same directory as the script:

![font5x8.bin]

**Setting up the code for The Things Network**

While we can send data to The Things Network, and our gateway might notice it, it isn't registered to a device yet. To register your device with The Things Network using ABP, you'll need to set three unique identifiers in radio_lorawan.py: the **Network Session Key**, the **Device Address**, and the **Application Session Key**.
Navigate to the Device Overview page for your Raspberry Pi device.

Make sure the Activation Method is set to ABP.

Before adding the unique identifiers to our code, we'll need to first expand them by clicking the <> icon.

These are your keys. We're going to enter them into our code, but we need to be careful - the keys on the Things Network console use parentheses or curly braces { } instead of brackets [ ].

First, copy the Device Address from the TTN console to the `devaddr` variable in the code.

Then, remove the braces { } from the device address.

A device address copied from The Things Network console would look like: `{ 0x26, 0x02, 0x1F, 0x07 }`. In the code, it'd look like: `devaddr = bytearray([0x26, 0x02, 0x1F, 0x07])`.

Then, copy the Network Session Key from the TTN console to the `nwkey` variable in the code. Make sure to modify the code to remove the parentheses/curly braces { }.
Finally, copy the Application Session Key from the TTN console to the `app` variable in the code. Make sure to modify the code to remove the parentheses/curly braces `{ }`.

That's all for now - we're ready to run our code!

Usage

Enter the following into the terminal and press enter:

```
python3 radio_lorawan.py
```

The OLED should display the operating mode (LoRaWAN).

Press Button A to send this data to the The Things Network application you built earlier.
Press Button B to view the Raspberry Pi's CPU utilization percentage.

Pressing Button C will periodically send a packet to the things network. You can modify this value (in seconds) by changing the `data_pkt_delay` variable in the code.

Checking Data on The Things Network Console

We want to make sure the data has been received on the other end. To do this, we'll navigate to the The Things Network Console and select the application. From the menu on the right hand side of the page, click Data.

If everything worked correctly, you'll see the payload from your device streaming into the page, in real time.

Neat, right? But while we received a payload, we still don't understand what it means. It's been sent to The Things Network and decoded on the client (Gateway) side, so it's not an AES-encrypted payload anymore. It's just not readable by humans.

Decoding the Payload

If you're sending packets in strange formats or encodings (like we are!), The Things Network Console has a programmable data decoder to decode the packets, and assign useful labels to the data.
Copy and paste the decoder script below into the decoder's integrated text editor and click save.

```javascript
// TinyLoRa - Raspberry Pi CPU Load Decoder
function Decoder(bytes, port) {
    var decoded = {};

    // Decode bytes to int
    var CPU_Load = (bytes[0] << 8) | bytes[1];

    // Decode int to float
    decoded.CPU_Load = CPU_Load / 100;

    return decoded;
}
```

Then, click the data tab. Next to the raw payload data, you should see the decoded data for the CPU load.