Feather + Raspberry Pi Weather Monitoring Network with LoRa or LoRaWAN

Created by Brent Rubell

https://learn.adafruit.com/multi-device-lora-temperature-network

Last updated on 2023-08-29 03:58:19 PM EDT
# Table of Contents

**Overview**
- Selecting a Transport
- LoRa to adafruit.io
- LoRaWAN to The Things Network
- Parts

**Wiring**
- Feather Wiring
- Raspberry Pi Wiring

**Feather Setup**

**LoRa Raspberry Pi Setup**
- Installing CircuitPython Libraries

**Adafruit IO Setup**

**Using with Adafruit IO**
- Feather Code Usage
- Feather Code Overview
- Pi Code Usage
- Code Overview

**Things Network Raspberry Pi Setup**
- Installing CircuitPython Libraries
- Testing the Setup
- Install the Single Channel Pi Gateway
- Gateway Usage

**Configuring The Things Network**
- Registering a Gateway with The Things Network
- Usage with Pi Zero
- Troubleshooting your Gateway

**Using with The Things Network**
- Feather Usage
- Running the Code using Mu Editor
- Pi Usage
- Checking the data on The Things Network Console
Overview

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

In this guide, we'll be connecting a sensor to two Feathers and pairing them with a gateway to bring all the data collected into the cloud. We're going to build our own network to send data from devices to a router. The router will then decode the data packets and route them to Adafruit IO or The Things Network.

Here's a network map (also known as a network topology) of the connections between the nodes/gateways/protocols of our network.

We will show how to make two different types of Feather to Raspberry Pi gateway networks - plain LoRa to adafruit.io and LoRaWAN for sending to The Things Network
Selecting a Transport

Devices need a way to communicate their data - the first step in building a network is selecting a mode of transport. As humans, we talk to each other in our native language(s). Electronic devices can also talk to each other, and there's a plethora of ways for them to communicate (for more information on transports, check out our All the Internet of Things - Part 1, which explains more about selecting a transport for your project)

LoRa to adafruit.io

In the first part of this guide, we're going to employ an Internet-of-Things transport protocol called LoRa to have our Feathers communicate with the Raspberry Pi Gateway. While we could use WiFi/BTLE/Cellular, we're not sending a large amount of data (a few bytes). We also do not need to send this data quickly, our sensors will transmit their data to the router every 15 minutes or so.

By sacrificing transmit speed and the amount of data which we're able to send, LoRa radios can transmit much farther than Bluetooth Low-Energy and WiFi with lower power utilization.

LoRaWAN to The Things Network

The second part of this guide uses LoRaWAN. 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.

Selecting a Device

To begin building a sensor network, we'll specify what we want the sensor device to do:

- Read a sensor
- Send data over LoRa (it'll need a LoRa radio)
- Connect to a LiPo battery (wireless operation!)
- Run CircuitPython ()
A board which meets all of these specifications is the Feather M0 with RFM95 LoRa Radio. This microcontroller runs CircuitPython, has a built-in LoRa radio, and a connector for LiPo batteries.

**Adafruit Feather M0 with RFM95 LoRa Radio - 900MHz**
This is the Adafruit Feather M0 RFM95 LoRa Radio (900MHz). We call these RadioFruits, our take on an microcontroller with a...
https://www.adafruit.com/product/3178

Next up, we'll need a sensor to gather and send data. The BME280 is a fantastic environmental sensor - it can read a location's temperature, barometric pressure, and humidity.

- Need less data? Just want the humidity from a location? [We have lots of sensors to help build a sensor node](https://www.adafruit.com/)

**Adafruit BME280 I2C or SPI Temperature Humidity Pressure Sensor**
Bosch has stepped up their game with their new BME280 sensor, an environmental sensor with temperature, barometric pressure and humidity! This sensor is great for all sorts...
https://www.adafruit.com/product/2652

**Selecting a Router/Gateway**

We're going to use the LoRa Radio Bonnet and a Raspberry Pi Zero W to act as a gateway - converting incoming packets (from the feathers) to readable values. After it converts this data, it'll forward it to either Adafruit IO or The Things Network.
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

Parts

You'll want to grab the following parts for this project:

1 x Raspberry Pi Zero
Raspberry Pi Zero with Headers

2 x FeatherWing Proto
FeatherWing Proto - Prototyping Add-on For All Feather Boards

1 x 8GB SD Card with NOOBS
8 GB MicroSD Card with full PIXEL desktop NOOBS - v2.8

1 x MicroUSB Power Supply
5V 2.5A Switching Power Supply with 20AWG MicroUSB Cable

1 x MicroUSB Cable
USB cable - USB A to Micro-B - 3 foot long

Wiring

Feather Wiring

You can connect the BME280 directly to a feather, or to a breadboard. But, we’d like to be able to remove it (and possibly add it to a Featherwing Doubler, or even Tripler!)
Make the following connections between the LoRa Radio Feather and the BME280:

- Board 3V to sensor VIN
- Board GND to sensor GND
- Board SCL to sensor SCK
- Board SDA to sensor SDI

Raspberry Pi Wiring

Since we're using an Adafruit LoRa Radio Bonnet with OLED, we don't need to wire anything. Connect the 20x2 header socket on the bonnet the header on your Pi and you're ready to roll!
Feather Setup

Some of the CircuitPython compatible boards come with CircuitPython installed. Others are CircuitPython-ready, but need to have it installed.

If you're using a Feather M0 RFM9x, you'll need to install CircuitPython on your board.

• To install CircuitPython on your Feather M0 RFM9x, follow this page

We'll need to install a few CircuitPython libraries on the Feather. To download them all at once, download the CircuitPython Library bundle.

• Don't have the latest CircuitPython Bundle installed? Click here to navigate a guide which will explain the process of downloading and installing the bundle

Once the bundle is downloaded and unzipped, drag and drop the following files over to the CIRCUITPY drive lib folder:

• If you're building a LoRa Device for communicating with Adafruit IO, you'll need the following libraries:
  ◦ adafruit_rfm9x
  ◦ adafruit_bme280
  ◦ adafruit_bus_device

• If you're building a LoRaWAN Device for communicating with TheThingsNetwork, you'll need the following libraries:
  ◦ adafruit_tinylor
  ◦ adafruit_bme280
  ◦ adafruit_bus_device

After the libraries are dragged over to the lib folder, we'll move on to setting up the Raspberry Pi to act as a gateway.
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 framebuffer module in order to write to the display.

```
sudo pip3 install adafruit-circuitpython-framebuf
```

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

```
sudo 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.

```
ladyada@limorFried:~/Desktop
$ ls -l font5x8.bin
-rw-r--r-- 1 ladyada None 1282 Oct 22 18:22 font5x8.bin
ladyada@limorFried:~/Desktop
```
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`.

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

Wiring Check, Pi Radio w/RFM9x

Author: Brent Rubell for Adafruit Industries

```
# Attempt to set up the RFM9x Module

```python
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)
```

display.show()
time.sleep(0.1)

To use the code, copy and paste the following command into your terminal:

```bash
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.

**Adafruit IO Setup**

If you do not already have an Adafruit IO account set up, head over to io.adafruit.com to link your Adafruit.com account to Adafruit IO.

We'll need to create a new feed to hold data for the first feather's BME280 sensor. The BME280 can output Temperature, Humidity, and Pressure data, so we'll need to create three feeds to hold data from each node's sensor.

![Image of radio module and OLED](image)

To do this, navigate to the feeds page on Adafruit IO.

Then click Actions -> Create New Feed, and name this feed `feather-1-temp`.

If you do not already know how to create a feed, head over to Adafruit IO Basics: Feeds.

Next, we'll create two more feeds for the first feather:

- `feather-1-humid`
Now that the first feather has feeds to store its data in, we’ll create two additional feeds for the second feather:

- **feather-2-humid**
- **feather-2-pressure**
- **feather-2-temp**

We’re also going to need our Adafruit IO username and our secret API key.

Navigate to your profile and click the View AIO Key button to retrieve them. Write them down in a safe place, we’ll need them for later.

Now that the feeds are set up - if you wish to visualize incoming data, you’ll want to set up a new dashboard to display feed data from the sensors.

- If you do not know how to create a new dashboard, or use the dashboard feature of Adafruit IO - [click here to learn more about Adafruit IO Dashboards](#) (1)

I set my dashboard up with three gauge blocks connected to each of the feeds, and a text box to identify the gauge cluster for each sensor node.
With Adafruit IO set up, let's move on to using the sensor network we've created.

Using with Adafruit IO

Feather Code Usage

We'll want to start by loading the code onto the Feather sensor nodes. To do this, copy the code below and rename it as code.py.

```python
# SPDX-FileCopyrightText: 2019 Brent Rubell for Adafruit Industries
# SPDX-License-Identifier: MIT

import time

CircuitPython LoRa Device w/BME280

import time
```
import busio
import digitalio
import board

# Import LoRa Library
import adafruit_rfm9x

# Import BME280 Sensor Library
import adafruit_bme280

# Device ID
FEATHER_ID = 0x01

# Delay between sending radio data, in minutes.
SENSOR_SEND_DELAY = 1

# Create library object using our Bus I2C port
i2c = busio.I2C(board.SCL, board.SDA)
bme280 = adafruit_bme280.Adafruit_BME280_I2C(i2c)

# Define radio frequency, MUST match gateway frequency.
RADIO_FREQ_MHZ = 905.5

# Define pins connected to the chip, use these if wiring up the breakout according to the guide:
CS = digitalio.DigitalInOut(board.RFM9X_CS)
RESET = digitalio.DigitalInOut(board.RFM9X_RST)

# Define the onboard LED
LED = digitalio.DigitalInOut(board.D13)
LED.direction = digitalio.Direction.OUTPUT

# Initialize SPI bus.
spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)

# Initialize RFM radio
rfm9x = adafruit_rfm9x.RFM9x(spi, CS, RESET, RADIO_FREQ_MHZ)

# Set transmit power to max
rfm9x.tx_power = 23

# sensor data
bme280_data = bytearray(8)

while True:
    # Get sensor readings
    temp_val = int(bme280.temperature * 100)
    print("Temperature: %0.1f C" % bme280.temperature)
    humid_val = int(bme280.humidity * 100)
    print("Humidity: %0.1f %" % bme280.humidity)
    pres_val = int(bme280.pressure * 100)
    print("Pressure: %0.1f hPa" % bme280.pressure)

    # Build packet with float data and headers
    bme280_data[0] = FEATHER_ID
    bme280_data[1] = (temp_val >> 8) & 0xff
    bme280_data[2] = temp_val & 0xff
    bme280_data[3] = (humid_val >> 8) & 0xff
    bme280_data[4] = humid_val & 0xff
    bme280_data[5] = (pres_val >> 16) & 0xff
bme280_data[6] = (pres_val >> 8) & 0xff
bme280_data[7] = pres_val & 0xff

# Convert bytearray to bytes
bme280_data_bytes = bytes(bme280_data)

# Send the packet data
print('Sending data...')
LED.value = True
rfm9x.send(bme280_data_bytes)
print('Sent data!')
LED.value = False

# Wait to send the packet again
time.sleep(SENSOR_SEND_DELAY * 60)

We'll also want to set the frequency which we are transmitting on. This value must be the same between the Feathers (nodes) and the Raspberry Pi (gateway), otherwise the gateway will not receive the packet, or visa-versa.

To do this, set `RADIO_FREQ_MHZ` to a frequency value in MHz. For example, I'll set it to 905.5MHz.

```
# Define radio frequency, must match device.
RADIO_FREQ_MHZ = 905.5
```

We also need a way for the gateway to identify each Feather. To do this, we'll set the `FEATHER_ID`. If you're preparing the first sensor node, leave this variable alone as its already set to 1.

```
# Device ID
FEATHER_ID = 1
```

Save the file (CTRL/CMD+S). This will reload and run the code from code.py.

Using the Mu editor, there is a built in REPL/Serial monitor. This is the recommended way to view output from your Python code. Alternatively you can use a terminal program to connect to the Feather's USB serial port.

You should see the following output from the REPL:

```
code.py output:
Temperature: 23.8 C
Humidity: 21.5 %
Pressure: 1005.2 hPa
Sending data..
Data sent!
```

Whenever the Feather node(s) send data using the radio, they'll blink an onboard LED (located on Pin #13) to show that the packet has been sent.
Feather Code Overview

One of the advantages of using LoRa over another type of transport (like WiFi) is that the radio is only turned on during transmission. Doing this reduces the amount of power consumption by the node.

We're going to build a packet to transmit the data to the gateway. This packet comprises of 7 bytes (the RFM9x library currently supports sending packets <= 252 bytes in length) which will hold all of our required sensor data, along with a unique identifier for our node, set in the code as **FEATHER_ID**.

The temperature values read by the BME280 are floating point (decimal) values (this is a value like 3.14 instead of 3). To send these values, we'll assemble the packet by sending the values as integers (which will later be re-assembled by the gateway)

Once assembled, the LoRa packet is sent using the radio.

```python
rfm9x.send(bme280_data)
```

To modify how frequently the packet is sent, change the variable **SENSOR_SEND_DELAY**, keeping in mind that the delay is in minutes.

That's it! Then, load the code onto the other feather(s) sensor nodes, changing the **FEATHER_ID** as you go.

Pi Code Usage

Next, we'll want to set up the Raspberry Pi with the code to use the RFM9x as a LoRa gateway. Save the code below on your Pi as lora_gateway.py.

```python
# SPDX-FileCopyrightText: 2019 Brent Rubell for Adafruit Industries
#
# SPDX-License-Identifier: MIT

---

Adafruit IO LoRa Gateway


by Brent Rubell for Adafruit Industries

---

# Import Python System Libraries
import time

# Import Adafruit IO REST client.
from Adafruit_IO import Client

# Import Blinka Libraries
import busio
```

©Adafruit Industries
import board
from digitalio import DigitalInOut, Direction, Pull

# Import SSD1306 module.
import adafruit_ssd1306

# Import RFM9x module
import adafruit_rfm9x

# Define radio frequency, must match device.
RADIO_FREQ_MHZ = 905.5

# 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
display = adafruit_ssd1306.SSD1306_I2C(128, 32, i2c, addr=0x3c)

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, RADIO_FREQ_MHZ)
prev_packet = None

# Set to your Adafruit IO username.
# (go to https://accounts.adafruit.com to find your username)
ADAFRUIT_IO_USERNAME = 'USER'

# Set to your Adafruit IO key.
ADAFRUIT_IO_KEY = 'KEY'

# Create an instance of the REST client.
aio = Client(ADAFRUIT_IO_USERNAME, ADAFRUIT_IO_KEY)

# Set up Adafruit IO feeds

classtemperature_feed_1 = aio.feeds('feather-1-temp')
humidity_feed_1 = aio.feeds('feather-1-humid')
pressure_feed_1 = aio.feeds('feather-1-pressure')

classtemperature_feed_2 = aio.feeds('feather-2-temp')
humidity_feed_2 = aio.feeds('feather-2-humid')
pressure_feed_2 = aio.feeds('feather-2-pressure')

def pkt_int_to_float(pkt_val_1, pkt_val_2, pkt_val_3=None):
    """Convert packet data to float."
    ""
    if pkt_val_3 is None:
        float_val = pkt_val_1 << 8 | pkt_val_2
    else:
        float_val = pkt_val_1 << 8 | pkt_val_2
else:
    float_val = pkt_val_1 << 16 | pkt_val_2 << 8 | pkt_val_3
    return float_val/100

while True:
    packet = None
    # draw a box to clear the image
    display.fill(0)
    display.text('Adafruit.IO LoRa GTWY', 0, 0, 1)

    # check for packet rx
    packet = rfm9x.receive()
    if packet is None or prev_packet:
        display.show()
        display.text('- Waiting for PKT -', 10, 20, 1)
    else:
        prev_packet = packet
        print('> New Packet!')
        # Decode packet
        temp_val = pkt_int_to_float(packet[1], packet[2])
        humid_val = pkt_int_to_float(packet[3], packet[4])
        pres_val = pkt_int_to_float(packet[5], packet[6], packet[7])

        # Display packet information
        print('Device ID: LoRa Feather #', packet[0])
        print("Temp: %0.2f C" % temp_val)
        print("Humid: %0.2f %" % humid_val)
        print("Pressure: %0.2f hPa" % pres_val)

        # Send to Feather 1 feeds
        if packet[0] == 0x01:
            display.fill(0)
            display.text('Feather #1 Data RX''d!', 15, 0, 1)
            display.text('Sending to IO...', 0, 20, 1)
            display.show()
            aio.send(temperature_feed_1.key, temp_val)
            aio.send(humidity_feed_1.key, humid_val)
            aio.send(pressure_feed_1.key, pres_val)
            display.text('Sent!', 100, 20, 1)
            display.show()

        # Send to Feather 2 feeds
        if packet[0] == 0x02:
            display.fill(0)
            display.text('Feather #2 Data RX''d!', 15, 0, 1)
            display.text('Sending to IO...', 0, 20, 1)
            display.show()
            aio.send(temperature_feed_2.key, temp_val)
            aio.send(humidity_feed_2.key, humid_val)
            aio.send(pressure_feed_2.key, pres_val)
            display.text('Sent!', 100, 20, 1)
            display.show()

        time.sleep(1)

    display.show()

We'll need to install the Adafruit IO Python library() to forward the messages by the LoRa radio to Adafruit IO. To do this, enter the following in your terminal:

```bash
sudo pip3 install adafruit-io
```

Once the Adafruit IO library is installed, we'll need to set the username and the key in the code:
Set the **ADAFRUIT_IO_USERNAME** to your Adafruit IO Username.

Then, set the **ADAFRUIT_IO_KEY** to your Adafruit IO Key.

And save the file.

Now that the Adafruit IO username and password are configured, run the code by typing the following into your terminal:

```
python3 lora_gateway.py
```

The display should show Adafruit IO GTWY and - Waiting for PKT -. This is the idle state, the gateway is listening for a new packet from the Feather node.

When a new packet is received, the display will change to show that a new packet from the Feather was received. It'll also forward the packet to Adafruit IO.

The terminal will change when a new packet is received:

```
- New Packet! -
Device ID: LoRa Feather # 1
Temp: 22.83 C
Humid: 17.90 %
Pressure: 370.11 hPa
- New Packet! -
Device ID: LoRa Feather # 2
Temp: 22.89 C
Humid: 10.48 %
Pressure: 370.11 hPa
```
To verify Adafruit IO has received the data, navigate to the monitor page on Adafruit IO (). This page shows a live-view of all incoming data (and error messages).

Your Adafruit IO Dashboard should look like the following as it receives data from the LoRa devices:
If you want this code to run when the Pi boots up, you'll need to make an entry in `rc.local`.

To do this, type the following into the terminal

```bash
sudo nano /etc/rc.local
```

Then, add the following line of code the file

```python
(sleep 30; python3 home/pi/rfm9x_gateway.py) &
```

and save the file.

Reboot the Pi by entering the following into the terminal

```bash
sudo shutdown -r now
```

When the Pi restarts, it'll run the gateway code.

Congrats! You've built your first LoRa network, and forwarded data to the internet.

**Code Overview**

Since the network topology can seem complicated at a glance, we'll take it piece by piece and explain how the code relates.

**Devices**

- **LoRa Packet**
  - ID: Feather #1
  - Temp: 24 C
  - Humid: 82.9% RH
  - Pressure: 357.45 hPA

- **LoRa Packet**
  - ID: Feather #2
  - Temp: 22.2 C
  - Humid: 45.9% RH
  - Pressure: 257.45 hPA

**Gateway/Router**

First, the gateway receives a new packet.

```python
packet = rfm9x.receive()
```
Then, it checks if the packet's value is empty or a retransmission. If it's a new packet value, it'll decode the packet.

Since we converted the float sensor value into an integer for transmission, and know the order of the packet, we can decode the temperature, humidity and air pressure. We also know the first byte of the packet (the header) is the Feather's unique identifier.

```python
# Decode packet
feather_id = packet[0]
temp_val = pkt_int_to_float(packet[1], packet[2])
humid_val = pkt_int_to_float(packet[3], packet[4])
pres_val = pkt_int_to_float(packet[5], packet[6])
```

Now that the data has been decoded, we're going to tackle the second role of the Radio Bonnet - sending the data as a transmission to Adafruit IO.

If the decoded `feather_id` is 1, we'll send the values to Adafruit IO feeds `temperature_feed_1`, `humidity_feed_1`, and `pressure_feed_1`.

```python
if feather_id is 1:
    display.fill(0)
    display.text('Feather #1 Data RX''d!', 15, 0, 1)
    display.text('Sending to IO...', 0, 20, 1)
    display.show()
    aio.send(temperature_feed_1.key, temp_val)
    aio.send(humidity_feed_1.key, humid_val)
    aio.send(pressure_feed_1.key, pres_val)
    display.text('Sent!', 100, 20, 1)
    display.show()
```

We have code for detecting a second feather, but if you have a third (or fourth!), you'll want to add another `elif` statement identical to the second one.
Things Network 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:

```
pip3 install adafruit-circuitpython-ssd1306
```

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

```
sudo pip3 install adafruit-circuitpython-framebuf
```

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

```
sudo pip3 install adafruit-circuitpython-rfm9x
```

### Testing the Setup

There have been a lot of steps so far - let's verify that everything is wired up and all the libraries are installed properly before moving on to sending and receiving data.

The following code is for checking if the RFM9x radio is set up for transmitting and receiving. Save the code on your Pi as `rfm9x_check.py`.
import time
import board
import adafruit_ssd1306
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 RFM9x LoRa Radio
CS = DigitalInOut(board.CE1)
RESET = DigitalInOut(board.D25)
spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)

while True:
    # 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:
        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)
To use the code, enter the following in your terminal:

```python
python3 rfm9x_check.py
```

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

![Font Image](image-url)

Then, let's check the setup:

If the RFM9x is detected, the OLED will display Detected!

You can test the buttons by pressing them and watching the display for changes.
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).

Install the Single Channel Pi Gateway

WiringPi () is PRE-INSTALLED with standard Raspbian systems. To check if it is pre-installed on your system, run:

```
gpio -v
```

If you get something, then you have it already installed. Otherwise, install WiringPi by running:

```
sudo apt-get install wiringpi
```

Then, clone the packet forwarder repository

```
git clone https://github.com/adafruit/single_chan_pkt_fwd.git
```

Navigate into the packet forwarder directory

```
cd single_chan_pkt_fwd/
```

and compile the code for the gateway

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

```plaintext
font5x8.bin
```

If you are using a Raspberry Pi ZERO, you will need to change a value within the configuration file. Using a text editor, open the `global_conf.json` file. Replace "is_pi_zero": false with "is_pi_zero": true. Your configuration file should look like the following.

```json
{
    "SX127x_conf": {
        "freq": 905100000,
        "spread_factor": 7,
        "pin_nss": 11,
        "pin_dio0": 3,
        "pin_rst": 25
    },
    "gateway_conf": {
        "ref_latitude": 0.0,
        "ref_longitude": 0.0,
        "ref_altitude": 10,
        "name": "Radio Pi Bonnet",
        "email": "contact@email.com",
        "desc": "Adafruit Radio Pi LoRa 1-Ch Gateway",
        "is_pi_zero": true,
        "servers": [
            {
                "address": "router.us.thethings.network",
                "port": 1700,
                "enabled": true
            },
            {
                "address": "router.eu.thethings.network",
                "port": 1700,
                "enabled": false
            }
        ]
    }
}
```

**Gateway Usage**

From within the `single_chan_pkt_fwd/` folder, run the Python program by entering the following into the terminal:

```bash
python3 lorawan_gateway.py
```
The display should show the Gateway EUI. Keep this screen open, we’ll need it for the next step.

---

Configuring The Things Network

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

Registering a Gateway with The Things Network

Before a Gateway can be used with The Things Network (TTN), it needs to be registered.

First, 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 Gateways
Click register gateway

Tick the I'm using the legacy packet forwarder checkbox.

Then, enter the Gateway EUI displayed on the gateway.

Fill out a description of what the gateway is.

Set the frequency plan to your region's frequency, we set ours to United States.

Set the Router to the location closest to you.

Adafruit is located in New York City, so I set the router to ttn-router-us-west.
Click Register Gateway. TTN will register the gateway and redirect you to the Gateway Overview page.

The Things Network does not list Single Channel gateways on their map - these gateways are not great at handling multiple devices or accurately receiving a packets every time they're sent by a device. However, they're great for hacking on, experimenting, and using in a staging/lab setup.

We'll want to set the Gateway Privacy to Private and set the location to Unlisted.

From the Gateway Overview page, click Edit Info.

On the Privacy Page, untick all of the options and click Save Settings.
Before starting up the gateway, we'll optionally set the gateway information.

From the Gateway Overview page, click Edit Info

For brand, select Single-Channel DIY Gateway

For model, type Raspberry Pi

It's time to test! Run the Python program by entering the following into the terminal:

```
python3 lorawan_gateway.py
```

and press the middle button to launch the gateway.

You should see the Status of the gateway on your console switch from from Not Connected to Connected. If you've had previously connected the gateway: the Last Seen label should update to Now.

Our gateway is now listening for new packets, it'll refresh whenever it's received a new packet.

Next, we're going to set up a device to send data to the gateway.
Usage with Pi Zero

If you are using a Pi Zero, you will need to modify the global_conf.json file prior to running the command above.

Change the following line in the JSON configuration from:

```
"is_pi_zero": false,
```

to

```
"is_pi_zero": true,
```

And save the file.

```json
{
    "SX127x_conf": {
        "freq": 905100000,
        "spread_factor": 7,
        "pin_nss": 11,
        "pin_dio0": 3,
        "pin_rst": 25
    },
    "gateway_conf": {
        "ref_latitude": 0.0,
        "ref_longitude": 0.0,
        "ref_altitude": 10,
        "name": "Radio Pi Bonnet",
        "email": "contact@email.com",
        "desc": "Adafruit Radio Pi LoRa 1-Ch Gateway",
        "is_pi_zero": false,
        "servers": [
            {
                "address": "router.us.thethings.network",
                "port": 1700,
                "enabled": true
            },
            {
                "address": "router.eu.thethings.network",
                "port": 1700,
                "enabled": false
            }
        ]
    }
}
```

Then, run the Python program by entering the following into the terminal:

```
python3 lorawan_gateway.py
```

Troubleshooting your Gateway
My gateway doesn't appear as "connected" in the console.

Double check the configuration steps on this page. Make sure the Pi is connected to the internet and that the Gateway EUI matches the Gateway EUI displayed on the Pi's display.

---

**Using with The Things Network**

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

We've sent data from our sensor devices to Adafruit IO over LoRa, but what about sending data over a different protocol - LoRaWAN?

The Adafruit_TinyLoRa() library allows the RFM9x to be used as a LoRaWAN device capable of communicating its data over longer distance with less power utilization.

**Feather Usage**

The code below will use the adafruit_tinyloras() library which provides high-level access to your radio transceiver features.

The code for sending data using a BME280 sensor, along with a unique feather identifier, to a Things Network Gateway is shown below:

```python
# SPDX-FileCopyrightText: 2019 Brent Rubell for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time
import board
import busio
from digitalio import DigitalInOut
import adafruit_bme280
from adafruit_tinylora.adafruit_tinylora import TTN, TinyLoRa

# Unique feather identifier
FEATHER_ID = 1

i2c = busio.I2C(board.SCL, board.SDA)
bme280 = adafruit_bme280.Adafruit_BME280_I2C(i2c)

# TinyLoRa/RFM9x Setup
spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
```
# pylint: disable=c-extension-no-member
cs = DigitalInOut(board.RFM9X_CS)
irq = DigitalInOut(board.RFM9X_D0)
rst = DigitalInOut(board.RFM9X_RST)

# TTN Device Address, 4 Bytes, MSB
devaddr = bytearray([0x00, 0x00, 0x00, 0x00])

# TTN Network Key, 16 Bytes, MSB
nwkey = bytearray([0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00])

# TTN Application Key, 16 Bytes, MSB
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, channel = 6)

# bme data packet
bme_d = bytearray(7)

while True:
    # Grab sensor data
    temp_val = int(bme280.temperature * 100)
humid_val = int(bme280.humidity * 100)

    bme_d[0] = FEATHER_ID
    # Temperature data
    bme_d[1] = (temp_val >> 8) & 0xff
    bme_d[2] = temp_val & 0xff
    # Humidity data
    bme_d[3] = (humid_val >> 8) & 0xff
    bme_d[4] = humid_val & 0xff

    print('Sending packet...')
lora.send_data(bme_d, len(bme_d), lora.frame_counter)
    print('Packet sent!')
lora.frame_counter += 1
time.sleep(1 * 60)

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 feather(s) with The Things Network, follow these steps and come back when you’re done.

Running the Code using Mu Editor

Upon saving (ctrl/cmd +s) your code, the board will refresh.

But where is the output? The serial monitor is hidden by default in Mu Editor.

In the Mu Editor, click the Serial button on the top icon-bar to bring up the Serial REPL.
The REPL should output the following:

```
 code.py output:
 Sending packet...
 Packet sent!
```

We still need to check that the Raspberry Pi Gateway receives the packet from the Feather. Let's set that up next.

**Pi Usage**

Run the Python program by entering the following into the terminal:

```
python3 lorawan_gateway.py
```

Press the first button to display statistics about the Pi, such as its IP, CPU load, and available memory.

Press the third button to display the name of the gateway along with the frequency, spreading factor, and The Things Network router.
Press the second button to launch the gateway. It'll display the current status (if a packet is received or not) and update the timestamp every minute.

When a LoRa Packet is received by the gateway, the terminal will display that a packet has been received, and it'll print out useful data coming from the packet:

```json
incoming pkt...
{"rxpk": [{"tmst":4067850409,"freq":905.1,"chan":0,"rfch":0,"stat":1,"modu":"LORA","datr":"SF7BW125","codr":"4/5","rssi":-35,"lsnr":10.0,"size":24,"data":"QLERAiYAKAABE0NCX+3lkSnK09wVlk8N"}]
```

The display will also refresh to display that a packet has been received.

We can't read the incoming data since it's encrypted. After all, device data shouldn't be readable by a gateway operator. We'll need to check our device on The Things Network Console to read the decrypted data.

**Checking the 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.

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 - BME280 and Feather ID Decoder
function Decoder(bytes, port) {
  var decoded = {};

  // Decode bytes to int
  var celciusInt = (bytes[1] << 8) | bytes[2];
  var humidInt = (bytes[3] << 8) | bytes[4];

  // Decode Feather ID
  decoded.featherID = bytes[0]

  // Decode int to float
  decoded.celcius = celciusInt / 100;
  decoded.humid = humidInt / 100;

  return decoded;
}
```

Then, click the data tab. Next to the raw payload data, you should see the decoded data for humidity and temperature, along with the unique ID of the feather.
<table>
<thead>
<tr>
<th>Fields</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;celcius&quot;: 22.32,</td>
</tr>
<tr>
<td>&quot;featherID&quot;: 1,</td>
</tr>
<tr>
<td>&quot;humid&quot;: 8.08</td>
</tr>
</tbody>
</table>