Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Medición del nivel de ruido urbano con una red IoT sobre Lora

Autor: Juan Carlos Álvarez Marín
Tutores: Rafael María Estepa Alonso
José Félix Ontañón Carmona

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Medición del nivel de ruido urbano con una red IoT sobre LoRa

Autor:
Juan Carlos Álvarez Marín

Tutores:
Rafael María Estepa Alonso
José Félix Ontañón Carmona

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Medición del nivel de ruido urbano con una red IoT sobre LoRa

Autor: Juan Carlos Álvarez Marín
Tutores: Rafael María Estepa Alonso
José Félix Ontañón Carmona

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

A mi familia

A mis amigos

Agradecimientos

Agradecer a mi familia la educación que me ha ofrecido en todos los aspectos ya que no se viene al mundo con alguna enseñanza, casi todo depende de ellos desde el principio. Y me siento afortunado de que mis padres y mis tíos tomasen hace años la decisión de estudiar carreras, tanto científicas como artísticas, ya que en esta vida hay que tener riqueza cultural de todo tipo y mentalidad abierta para afrontar los diversos problemas que van sucediendo. Sin duda venir a vivir a Sevilla y estudiar teleco ha sido un problema que he sabido afrontar, incluso disfrutar, gracias al apoyo de los increíbles colegas que he conocido, a los profesores que con su estricta pero necesaria docencia me han enseñado la cultura del esfuerzo y a mis padres que me apoyaron económicamente poniendo toda su fe en mí.

Juan Carlos Álvarez Marín

Sevilla, 2020

Resumen

Este TFG trata sobre el internet de las cosas (IoT) aplicado al desarrollo de las ciudades inteligentes (*smart cities*). En concreto se estudia la red LoRaWAN como medio para transferir datos entre los sensores utilizados para medir el ruido en entornos urbanos y el servidor donde se alojarán los datos de esta aplicación.

Para el desarrollo de esta aplicación se utiliza el Arduino IDE para implementar el código del firmware que se instala en el dispositivo con capacidad de transmitir señales de radio LoRa, y esta será la parte técnica de un proyecto completo desarrollado en la empresa Pod Group, que tiene la intención de llevar este producto al mercado.

A lo largo de este TFG se realizan estudios de diferentes ámbitos de la tecnología de telecomunicación, tanto de telemática, señales, electrónica y sonido. Todo esto para desarrollar un producto lo más económico y eficiente posible que permita en el futuro un alto margen de ganancias y un alto rendimiento de la aplicación.

Abstract

This final project is about the Internet of Things (IoT) applied to the development of smart cities. Specifically, the LoRaWAN network is studied as a way to control data between the sensors used to measure noise in urban environments and the server where the data of this application are hosted.

For the development of this application, the Arduino IDE is used to implement the firmware code that is installed in the device with the capacity to transmit LoRa radio signals, and this will be the technical part of the complete project developed in the company Pod Group, which has the intention of bringing this product to the market.

Throughout this final project, studies are made of different movements in telecommunications technology, both telematics, signals, electronics and sound. All this to develop a product that is as economical and efficient as possible that will allow a high profit margin and a high performance in the future.

Índice

ÍNDICE DE FIGURAS

Notación

Inglés	Palabra en inglés

1 INTRODUCCIÓN

La ciencia más útil es aquella cuyo fruto es el más comunicable.

- Leonardo Da Vinci -

El término *smart city,* cuya traducción es ciudad inteligente, surgió para nombrar espacios urbanos modernos con un auge de nuevas soluciones tecnológicas que aportan calidad de vida a la ciudadanía, ya sea creada por empresas privadas, por empresas gubernamentales o por estas dos últimas en cooperación. Este concepto también implica que el desarrollo tecnológico sea eficiente, es decir, que haya un desarrollo económico y social, a la vez que el impacto contra el medio ambiente sea sostenible. En este contexto tienen una gran importancia las redes de telecomunicación, ya que son el medio a partir del cual el concepto de *smart city* se asienta, dando conectividad a las aplicaciones emergentes y optimizando la forma en la que ciertos tipos de dato se mandan entre dos sistemas finales de comunicación.

A partir de las *smart cities* surge el concepto de IoT, *Internet of Things* o internet de las cosas, que significa que los objetos que nos son útiles en nuestra vida cotidiana se interconecten mediante internet, con objetivo de ofrecer una solución más eficaz y moderna a las soluciones de una *smart city*. Este concepto surgió a partir del trabajo realizado por Kevin Ashton. Él fué asistente de gerente en la multinacional estadounidense dedicada a bienes de consumo *The Procter & Gamble Company (P&G),* y trabajó con el investigador David Brock y los profesores Sanjay Sarmay y Sunny Siu en el *Massachusetts Institute of Technology (MIT),* donde desarrolló una idea que en 1999 presentó en la *(P&G)* como *"Internet of Things"*, donde exponía una mejora en la cadena de suministro usando identificación por radiofrecuencia (RFID) y el uso de internet. Y con internet en pleno auge (para poner en contexto, esto solo ocurre un año después del nacimiento de Google), la idea llamó el interés de los ejecutivos.

La característica esencial de una red IoT, tal y como dice la Oficina de Seguridad del Internauta (OSI), es que los dispositivos y sensores electrónicos interconectados entre sí se encarguen de medir, recopilar y enviar datos a un servidor centralizado o a la nube, donde estos pueden ser procesados y reenviados [1]. Estos dispositivos serán en general pequeños y tendrán actuadores accionados por órdenes de los servidores y sensores que recogerán información del medio y la comuniquen. Para esta comunicación se puede utilizar cualquier tecnología, tanto tecnologías que ya llevan un tiempo desarrollándose, como WIFI o Bluetooth, tanto tecnologías pioneras como LoRaWAN, ya que ofrecen características que dependiendo del medio en el que se utilice pueden ser ventajosas, ya sea por eficacia, coste o seguridad.

1.1. Motivación

La motivación de este TFG parte de la falta de una infraestructura que sirva para medir el ruido urbano y dejar a disposición pública los datos medidos. Esta infraestructura consiste en una red IoT de sensores acústicos fijos que mandarán constantemente los datos medidos a un servidor, que será accesible a cualquier entidad interesada en conocer el nivel de ruido en los diferentes puntos de su ciudad.

LoRaWAN será la arquitectura web desarrollada [2] sobre la cual se enviarán los datos, ya que se ajusta a los requisitos de IoT tales como servicios de comunicación bidireccional, seguridad extremo a extremo, movilidad y localización. Y el servidor donde se alojarán estos datos se encontrará en *TheThingsNetwork (TTN)* [3].

1.2. Objetivos

El objetivo del proyecto global, hasta la comercialización de los sensores al público, es tener una base de datos donde cualquiera pueda consultar el dato de ruido en los distintos puntos de la ciudad, ya sea por interés personal o por interés del propio ayuntamiento. Además, los mismos usuarios tendrán la opción voluntaria de comprar un kit para el montaje del sensor, y ponerlo en funcionamiento no tendrá ningún coste adicional, ya que la propia red LoRaWAN está en la banda libre de radiofrecuencia y no es necesaria ninguna licencia.

En este TFG, el objetivo está en la parte técnica del proyecto, que abarcará desde la selección y configuración de los dispositivos IoT hasta la configuración de la aplicación en la web de TheThingsNetwork con sus correspondientes integraciones para verificar las pruebas realizadas.

2 CONOCIMIENTOS PREVIOS Y ESTADO DEL ARTE

2.1. Previsiones de mercado

Según el estudio realizado por Telefónica llamado Things Matter[4], hay un incremento importante del uso de tecnologías IoT y 1 de cada 2 ciudadanos ya conoce el significado de este concepto. Las empresas entrevistadas ven el IoT como una tendencia imparable, ya que desde el estudio en 2017, ha habido un aumento de un 67% en ciudadanos que tienen objetos conectados. Por otra parte, las empresas se aprovechan de esta situación, ya que un 87% de usuarios que usan estas tecnologías no renunciarían a ellas.

En concreto, este TFG se realiza en una empresa privada de la ciudad de Sevilla (Pod Group) con el objeto de que, en el futuro, el propio Ayuntamiento herede e invierta en la tecnología desarrollada y la incorpore a la ciudad. Y Francisco Lineros[4], director general de modernización digital del Ayuntamiento de Sevilla, confía plenamente en este sector tal y como expresa en el estudio Things Matter, y asegura que ahora mismo se tienen todas las herramientas tecnológicas para su desarrollo. Algo que hace 20 años era conceptualmente posible pero técnicamente imposible. Y refiriéndose a las redes de sensores urbanas, Francisco Lineros afirma que: "El concepto de IoT es aplicable a todo. El IoT es a la smart city lo que los sentidos al cuerpo humano."

2.2. Solución IoT para redes urbanas de sensores

Actualmente hay un auge en tecnologías de comunidad ciudadana que se basan en el IoT y el modelo de *smartcity,* y muchas empresas del ámbito tecnológico se están sumando a esta idea gracias a los incentivos del Estado. Por otra parte, el coste de las redes, la escalabilidad y la flexibilidad de estas son otro incentivo que están tomando en cuenta estas empresas, hasta el punto de que, aparte de las empresas, la comunidad ciudadana aprovecha el bajo coste para configurar sus redes IoT y darle alguna aplicación práctica sin ánimo de lucro.

Las aplicaciones de sensores en redes urbanas IoT tienen como características fundamentales la comunicación a baja tasa de envío, con rango suficiente para permitir la comunicación de extremo a extremo de la ciudad, y capaz de evitar interferencias provocadas en el medio urbano para lograr una tasa de error aceptable.

2.2.1 LoRaWAN

LoRa Alliance® es una asociación abierta sin fines de lucro que surgió en 2015 y lanzó la tecnología LoRaWAN. Nótese la diferencia entre LoRaWAN, que le da nombre al protocolo de comunicación a nivel OSI 2 y 3 y a la arquitectura de red, y LoRa, que es la capa física en el radioenlace.

LoRaWAN es una buena solución urbana para redes de sensores, ya que sus principales características son transmisión a baja potencia, alta sensibilidad en dispositivos y *Gateways* y alto rango de comunicación en la banda libre. Más adelante, en el apartado 4.3., se hará un estudio más profundo de la arquitectura.

En este punto, se realizará un breve estudio de las características de la red LoRaWAN, para luego poder profundizar más sobre el firmware del device y verificar correctamente las pruebas.

Para empezar, se presenta la siguiente arquitectura en la red LoRaWAN.

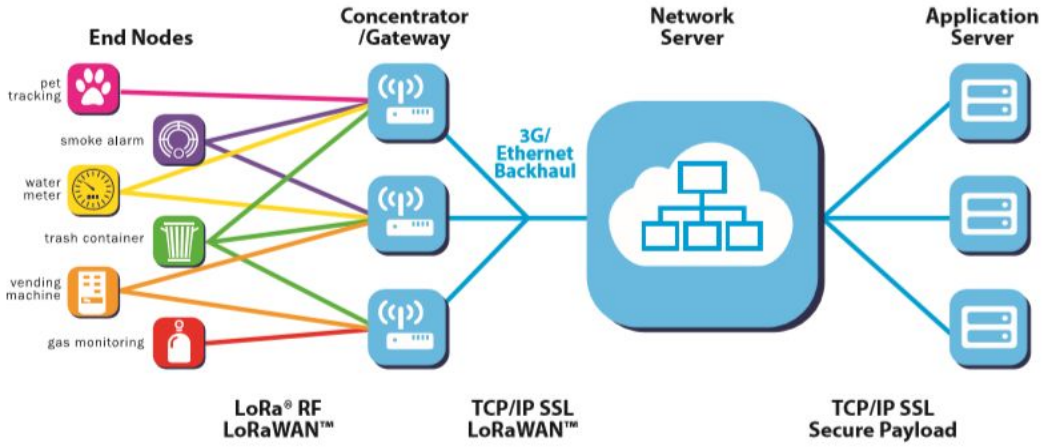

Figura 1. Arquitectura de red LoRaWAN

En la especificación *LoRaWAN Backend Interfaces* [2], vienen definidas las siguientes interfaces que componen la red LoRaWAN mostrada en la figura anterior:

- *End-device:* Dispositivo final, que es un sensor o un actuador y está conectado de forma inalámbrica a una red LoRaWAN a través de *gateways*. La capa de aplicación del dispositivo final está conectada a un servidor de aplicación específico en la nube.
- *Radio gateway:* La puerta de enlace de radio o *radio gateway* reenvía todos los paquetes de radio LoRaWAN recibidos del dispositivo final al servidor de red que está conectado a través de una red troncal IP (back-bone). El *gateway* funciona únicamente en la capa física y su función es decodificar los paquetes de radio del enlace ascendente y reenviarlos sin procesar al servidor de red. Para los enlaces descendentes, el *gateway* simplemente ejecuta las solicitudes de transmisión procedentes del servidor de red sin ninguna interpretación de la carga útil (*payload*).
- *Network server:* El *network server* (NS) o servidor de red termina la capa MAC de LoRaWAN para los dispositivos finales. Las características genéricas del NS son:

  • Comprobación de la dirección del dispositivo final.

  • Autenticación de trama y comprobaciones del contador de tramas.

  • Reconocimientos.

  • Adaptación de la tasa transmisión de datos.

  • Responder a todas las solicitudes de capa MAC procedentes del dispositivo final.

  • Reenviar la carga útil (*payload)* de aplicaciones de enlace ascendente a los servidores de aplicación apropiados.

  • Poner en cola la carga útil de los enlaces descendentes procedentes de cualquier servidor de aplicación a cualquier dispositivo conectado a la red.

  • Reenviar mensajes de solicitud y aceptación de unión (*join-request, join-accept*) entre los dispositivos finales y los servidores de unión (*join servers*).

  En una arquitectura de itinerancia de datos, un NS puede desempeñar tres roles diferentes dependiendo de si el dispositivo final está en itinerancia o no, y el tipo de itinerancia que está implicado:

  • El servicio NS (sNS) controla la capa MAC del dispositivo final.

  • El *home* NS (hNS) es donde se almacena el perfil del dispositivo, el perfil del servicio, el perfil de enrutamiento y el devEUI (identificador del *device*). El hNS está conectado al servidor de aplicación, y cuando hNS y sNS están separados, hay un acuerdo de itinerancia entre ellos y los paquetes de enlace ascendente y de enlace descendente se reenvían entre ellos.

• El *forwarding* NS (fNS) es el NS que administra los *gateway*. Cuando el sNS y el fNS están separados, hay un acuerdo de itinerancia entre ellos y los paquetes de enlace ascendente y descendente se reenvían entre ellos. Puede haber uno o más fNS sirviendo al dispositivo final.

- *Join server*: el servidor de unión (JS) administra el proceso de activación del dispositivo final por aire (*Over-the-Air activation*, OTA). Puede haber varios JS conectados a un NS, y un JS puede conectarse a varios NS. El dispositivo final indica que el JS debe ser cuestionado a través del campo JoinEUI del mensaje Join-request (cada JS se identifica mediante un valor JoinEUI único). El JS conoce el identificador del servidor de la red doméstica del dispositivo final, y proporciona esa información a los otros NS cuando lo requieren los procedimientos de itinerancia. El JS contiene la información necesaria para procesar tramas de canal ascendente de solicitud de unión y generar las tramas de unión-aceptación del canal descendente. También realiza la transferencia de clave de sesión de red y aplicación (NwkKey, AppKey), comunicando la clave de sesión de red del dispositivo final al NS y la clave de sesión de aplicación al servidor de aplicaciones correspondiente. El JS contendrá la siguiente información para cada dispositivo final bajo su control:

  • DevEUI.

  • AppKey.

  • NwkKey.

  • Identificador del servidor de red doméstica.

  • Identificador del servidor de aplicaciones.

  • Jerarquía para seleccionar la red preferida en caso de que varias redes puedan servir al dispositivo.

  • Versión LoRaWAN del dispositivo final.

  Las claves NwkKey y AppKey solo están disponibles en JS y dispositivos finales, y nunca se envían al NS ni al *application server* (AS). El JS y el NS podrán establecer una comunicación segura que proporcione autenticación de punto final, integridad y confidencialidad. El JS también podrá enviar de forma segura la AppKey al servidor de aplicación. Por otra parte, el JS puede estar conectado a varios servidores de aplicación (AS), y un AS puede estar conectado a varios JS. El JS y el AS podrán establecer una comunicación segura, con las mismas características que entre el JS y el NS.

- *Server application* (AP): el servidor de aplicación (AS) controla todas los *payload* de capa de aplicación de los dispositivos finales asociados y proporciona el servicio de nivel de aplicación al usuario final. También genera todos los *payload* de enlace descendente de la capa de aplicación hacia los dispositivos finales conectados. Puede haber varios AS conectados a un NS, y un AS puede estar conectado a varios NS. Un AS también puede estar conectado a varios JS, y el home NS enruta en el canal de subida hacia el AS apropiado, basándose en el DevEUI. Además de los elementos de red antes mencionados, la arquitectura LoRaWAN define las siguientes interfaces de red entre estas entidades:

  • hNS-JS: Esta interfaz se utiliza para admitir el procedimiento de unión (activación) entre el JS y el NS.

  • vNS-JS: Esta interfaz se utiliza para el procedimiento de activación de itinerancia. Se utiliza para recuperar el NetID del hNS asociado con el dispositivo final.

  • ED-NS: Esta interfaz se utiliza para soportar la señalización de la capa MAC de LoRaWAN y la entrega del *payload* entre el dispositivo final y el NS.

  • AS-hNS: Esta interfaz se utiliza para soportar la entrega del *payload* de la aplicación y también los metadatos asociados entre el AS y el NS.

  • hNS-sNS: Esta interfaz se utiliza para soportar la señalización de itinerancia y la entrega del *payload* entre el hNS y el sNS.

  • sNS-fNS: Esta interfaz se utiliza para soportar la señalización de itinerancia y la entrega del

payload entre el sNS y el fNS.

• AS-JS: Esta interfaz se utiliza para entregar la clave de sesión de la aplicación (App Session Key) del JS al AS.

Las interacciones descritas anteriormente se muestran en los siguientes diagramas, el primero en el caso de tener el dispositivo final en la red de casa y el segundo en el caso de tenerlo conectado mediante itinerancia.

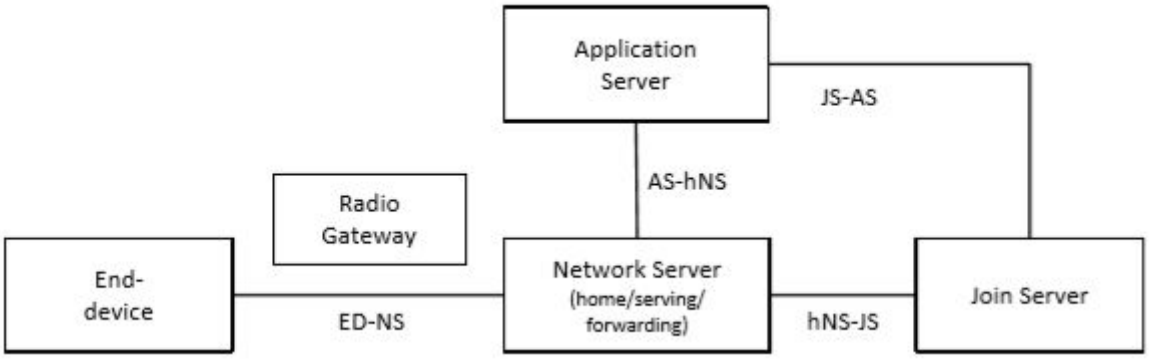

Figura 2. Diagrama de red LoRaWAN con acceso en red doméstica

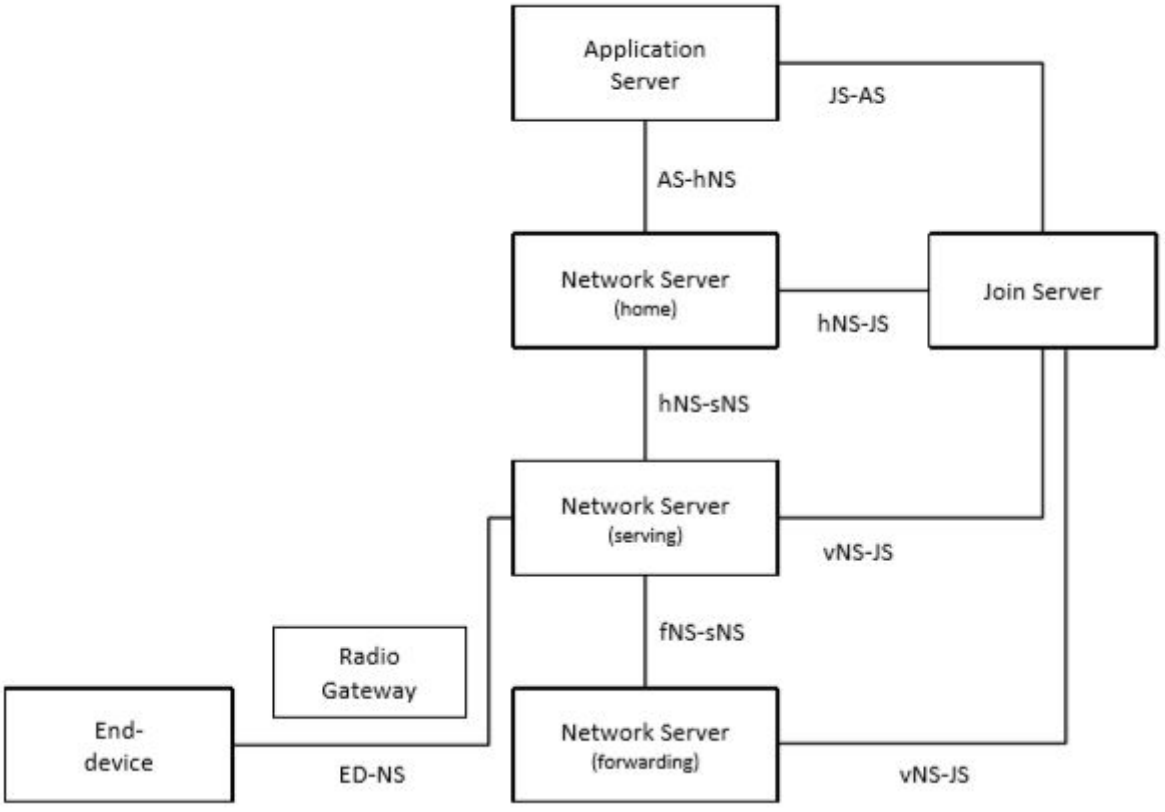

Figura 3. Diagrama de red LoRaWAN con acceso mediante itinerancia

2.2.1.1 Torre de protocolos

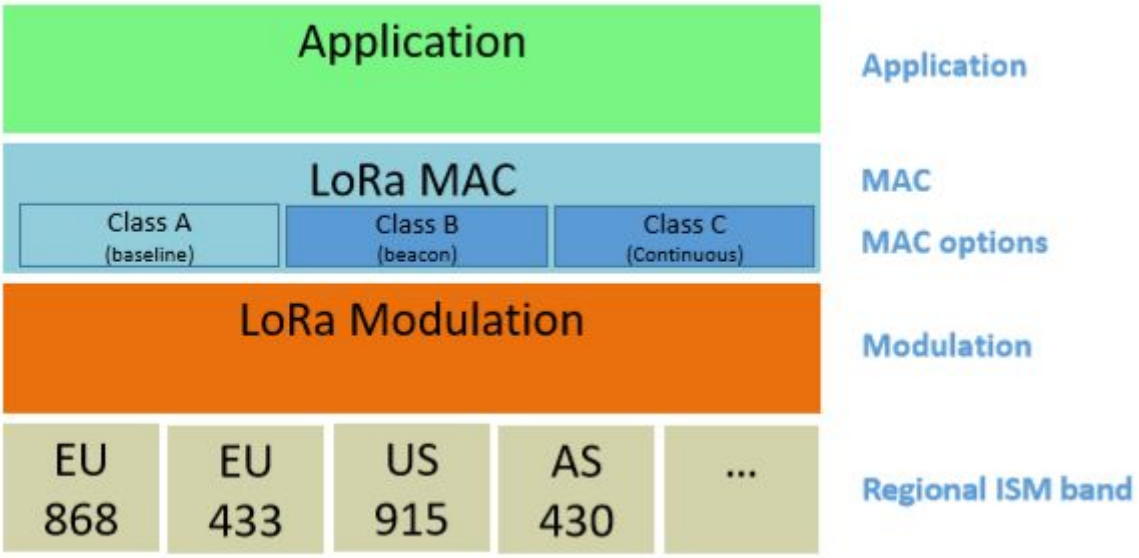

Figura 4. Torre de protocolos LoRaWAN

Como se observa en la figura, la torre de protocolos de LoRaWAN se divide, según el modelo OSI, en capa física, de enlace y de aplicación.

En la capa física se define la modulación LoRa y la banda de radiofrecuencia asignada a la región, que en el caso de este proyecto será la de 868Mhz de Europa. En el siguiente apartado se comentarán las características esenciales de esta banda y su modulación.

La capa de enlace está formada por el protocolo de control de acceso al medio (MAC) llamado LoRaWAN, que es para todos los dispositivos un protocolo de clase A, y además, puede tener características opcionales en otras clases para determinados dispositivos:

- Clase A (dispositivos finales bidireccionales): Los dispositivos finales de la clase A permiten comunicaciones bidireccionales, y en la transmisión del enlace ascendente es seguida por dos ventanas de recepción de enlace descendente. La ranura de transmisión programada por el dispositivo final se basa en un tiempo aleatorio (como método de contienda ALOHA). Esta clase es la que más energía ahorra, ya que solo espera hasta la segunda ventana para recibir el mensaje en el enlace descendente, y las comunicaciones de enlace descendente desde el servidor en cualquier otro momento tendrán que esperar hasta el siguiente enlace ascendente programado. Que según las normas gubernamentales no puede superar el 1% del duty cycle, esto quiere decir que el dispositivo final solo puede estar transmitiendo sobre el canal ascendente un 1% del total del tiempo que trabaja.
- Clase B (dispositivos finales bidireccionales con ranuras de recepción programadas): Los dispositivos de clase B amplían la clase A agregando ventanas de recepción programadas para los mensajes de enlace descendente. Y con balizas (*beacon*) sincronizadas por tiempo transmitidas por el *gateway*, los dispositivos abren periódicamente las ventanas de recepción, por lo que los dispositivos de esta clase gastan más potencia que en la anterior.
- Clase C (dispositivos finales bidireccionales con ranuras de recepción máximas): Los dispositivos de la clase C tiene la ventana de recepción siempre abierta, excepto cuando están transmitiendo. Debido a esto son los dispositivos que más energía consumen.

2.2.1.2 Parámetros de región

En Europa, la normativa sobre el espectro de radiofrecuencia ISM la impone la Instituto Europeo de Normas de Telecomunicaciones (ETSI). Y para LoRaWAN, la banda establecida por la norma está comprendida en el rango 863/870 MHz, pudiendo utilizar tanto modulación LoRa como FSK.

DataRate	Configuration	Indicative physical bit rate [bit/s]
0	LoRa: SF12 / 125 kHz	250
1	LoRa: SF11 / 125 kHz	440
2	LoRa: SF10 / 125 kHz	980
3	LoRa: SF9 / 125 kHz	1760
4	LoRa: SF8 / 125 kHz	3125
5	LoRa: SF7 / 125 kHz	5470
6	LoRa: SF7 / 250 kHz	11000
7	FSK: 50 kbps	50000

Figura 5. Configuraciones de canal LoRa de la banda ISM EU868

Las parámetros que un dispositivo final puede tomar en el enlace descendente/ascendente para una modulación LoRa y FSK son, respectivamente:

Parameter	Uplink value	Downlink value
Preamble size	8 symbols	
SyncWord	0x34 (Public)	
Header type	Explicit	
CRC presence	True	False
Coding Rate	4/5	
Spreading Factor	Defined by the Datarate, specified in each region	
Bandwidth		
IQ polarization	Not-inverted	Inverted

Figura 6. Parámetros de modulación LoRa de dispositivos finales

Parameter	Uplink value	Downlink value
Preamble size	5 bytes	
SyncWord	0xC194C1	
Bitrate	50000 bit/sec	
Tx frequency deviation	25kHz (SSB[1])	
Rx bandwidth	50kHz (SSB)	
Rx bandwidth AFC	80kHz (SSB)	
CRC presence	True (CRC-CCITT)	
Gaussian filter	BT = 1,0	
DC Free Encoding	Whitening Encoding	

Figura 7. Parámetros de modulación FSK de dispositivos finales

Los campos de los paquetes con modulación LoRa son:

8 Symbols	8 Symbols		*L* bytes (from PHDR)	2 Bytes
Preamble	PHDR	PHDR_CRC	PHYPayload	CRC (uplink only)

Figura 8. Cabeceras del paquete con modulación LoRa

Los campos de los paquetes con modulación de capa física FSK son:

Size (bytes)	5	3	1	*L bytes from PHYPayloadLength*	2
Packet Structure	Preamble	SyncWord	PHYPayloadLength	PHYPayload	CRC

Figura 9. Cabeceras del paquete con modulación FSK

La carga útil física (PHYPayload) encapsula los siguientes campos:

MHDR	MACPayload	MIC

or

MHDR	Join-Request	MIC

or

MHDR	Join-Response	MIC

Figura 10. Datos encapsulados en carga útil física (PHYPayload)

Y a nivel 2 (MAC), la carga útil (MACPayload) encapsula los siguientes campos:

FHDR	FPort	FRMPayload

Figura 11. Datos encapsulados en carga útil de nivel 2 OSI (MACPayload)

Donde la cabecera de trama FHDR encapsula la dirección del dispositivo final (DevAddr), un octeto de control de trama (FCtrl), un contador de trama de 2 octetos (FCnt) y hasta 15 octetos de opciones de trama (FOpts) utilizados para transportar comandos MAC.

Size (bytes)	4	1	2	0..15
FHDR	DevAddr	FCtrl	FCnt	FOpts

Figura 12. Datos encapsulados en cabecera de trama (FHDR)

Además, como ya se ha dicho, hay restricciones de ciclo de trabajo impuestas por la ETSI (< 1%), aunque no hay limitaciones de transmisión máxima o de tiempo de permanencia en el canal.

2.2.1.3 Métodos de activación del dispositivo final

Para que un *device* pueda ser utilizado en la red LoRaWAN, necesita completar el proceso de activación. Hay dos tipos de proceso:

- Activación sobre el aire (OTAA): Esta la forma preferida y la más segura de conectarse al servidor de aplicación. Los dispositivos realizan el procedimiento sobre la red mostrado en la figura del siguiente apartado, durante el cual se les asigna un DevAddr dinámico y se negocian las claves de seguridad con el dispositivo.
- Activación por personalización(ABP): Esta forma permite codificar el DevAddr y las claves de seguridad en el dispositivo final. Esta estrategia puede parecer más sencilla, ya que se omite el

procedimiento de combinación sobre la red, pero tiene algunas desventajas relacionadas con la seguridad.

La diferencia entre estos dos métodos es que las claves de seguridad de sesión y aplicación (NwkSKey, AppSKey), que son únicas para cada *device* en una sesión abierta, son regeneradas por cada activación OTAA, y en la activación ABP siempre son las mismas hasta que el usuario las cambie. Lo que conlleva un problema de seguridad.

2.2.1.4 Flujo de mensajes

Los diagramas de flujo de mensajes para activación OTAA y ABP son, respectivamente:

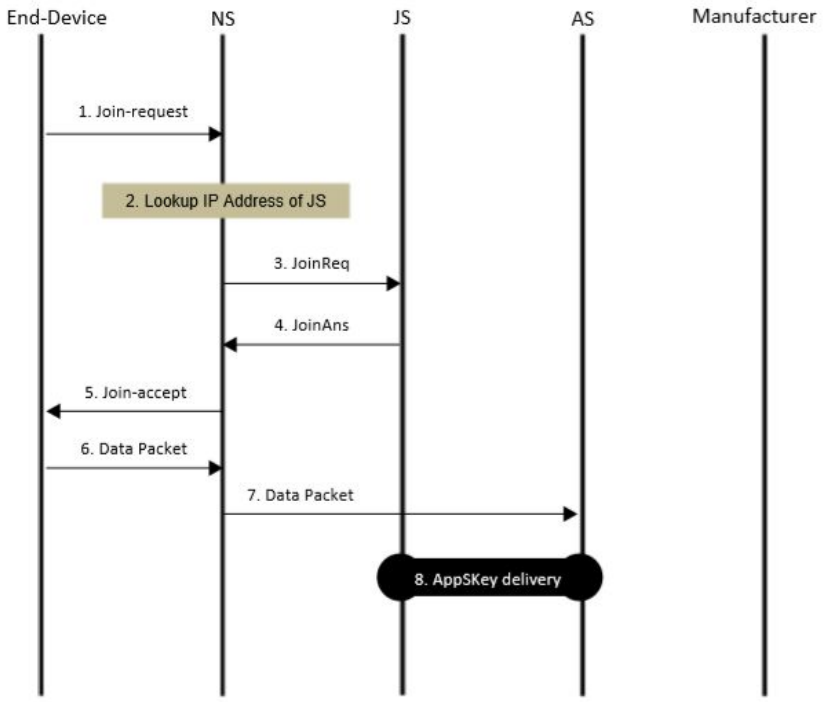

Figura 13. Diagrama de paso de mensajes para activación OTAA del dispositivo final

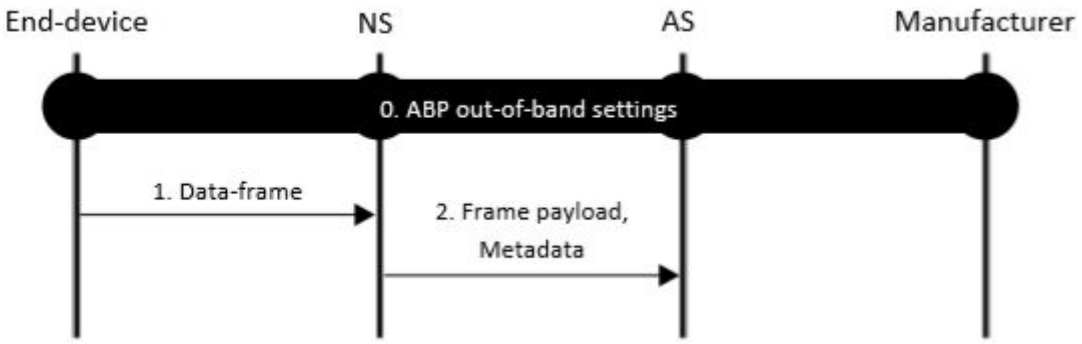

Figura 14. Diagrama de paso de mensajes para activación ABP del dispositivo final

Los siguientes diagramas muestran el procedimiento de envío de mensajes y cierre de sesión en una conexión itinerante:

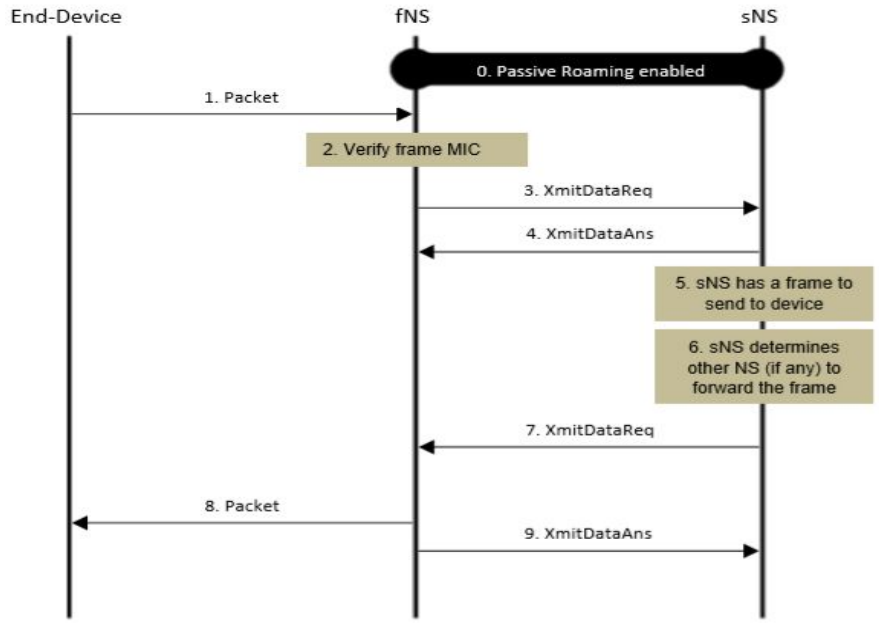

Figura 15. Diagrama de paso de mensajes de envío de mensaje

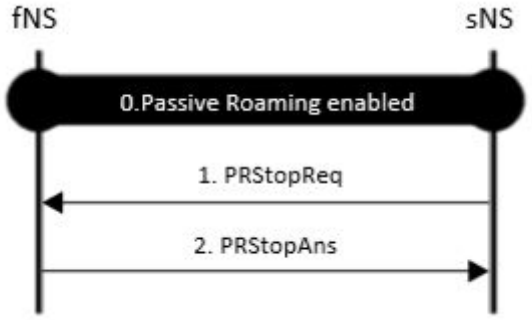

Figura 16. Diagrama de paso de mensajes de cierre de sesión de una conexión itinerante

Y por último, un ejemplo de conexión con el *device* conectado a la red doméstica mediante el método POST del protocolo de nivel de aplicación HTTP:

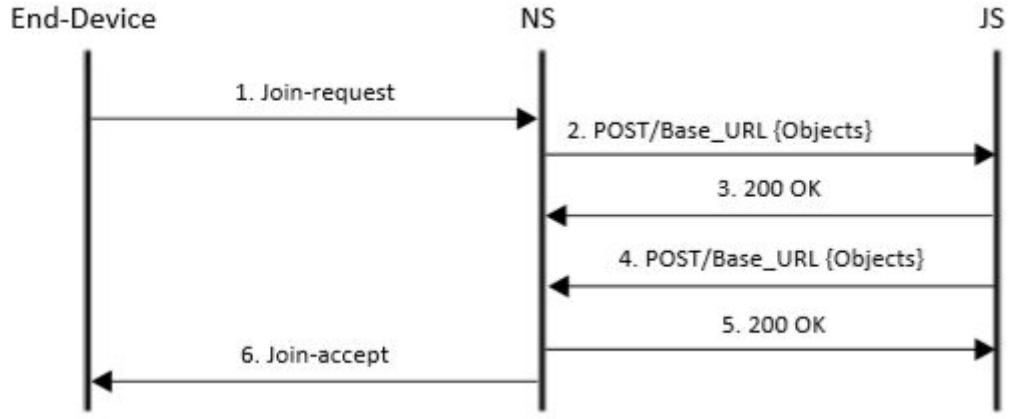

Figura 17. Diagrama de paso de mensajes de conexión mediante HTTP entre device y join server

2.2.2 TheThingsNetwork

TheThingsNetwork (TTN) es una web que proporciona a cualquier usuario una red abierta en la que montar aplicaciones IoT, además de ser un miembro contribuyente en LoRa Alliance®. TTN es una buena opción para el montaje de esta red, ya que, incluso a largo plazo, proporciona escalabilidad, fiabilidad y coste adecuados.

TTN proporciona el enrutamiento y procesado de paquetes LoRaWAN y está, en concreto, entre los *Gateways* y la aplicación creada por el usuario como se muestra en la siguiente imagen.

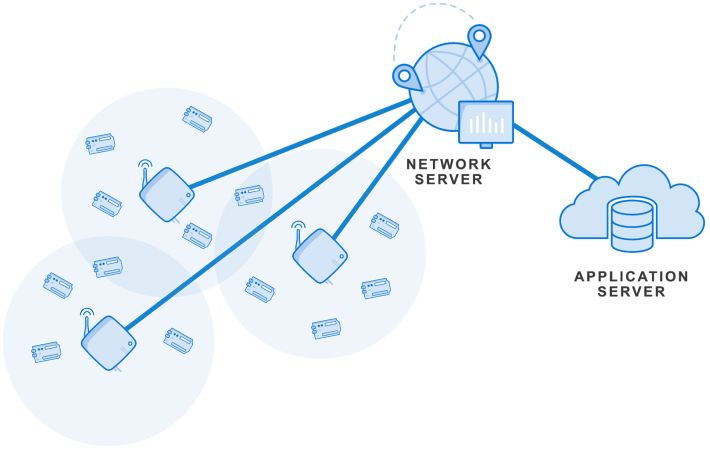

Figura 18. Arquitectura básica de LoRaWAN

Desde la página web de TheThingsNetwork se pueden configurar las aplicaciones, los dispositivos implicados en la aplicación, registrar gateways y tratar los datos recibidos con integraciones.

Para empezar, lo primero será crear una cuenta en la web, y después nos centraremos en el apartado CONSOLE, que es donde se configura lo expuesto en el anterior párrafo: https://console.thethingsnetwork.org/

2.2.2.1 Gateway

En el apartado de Gateways, estos se registran introduciendo un nombre de identificación, el protocolo de registro del gateway, una descripción, el plan de frecuencia de la región, el router con salida a internet que se conectará al gateway y el lugar geográfico donde se encuentra el gateway:

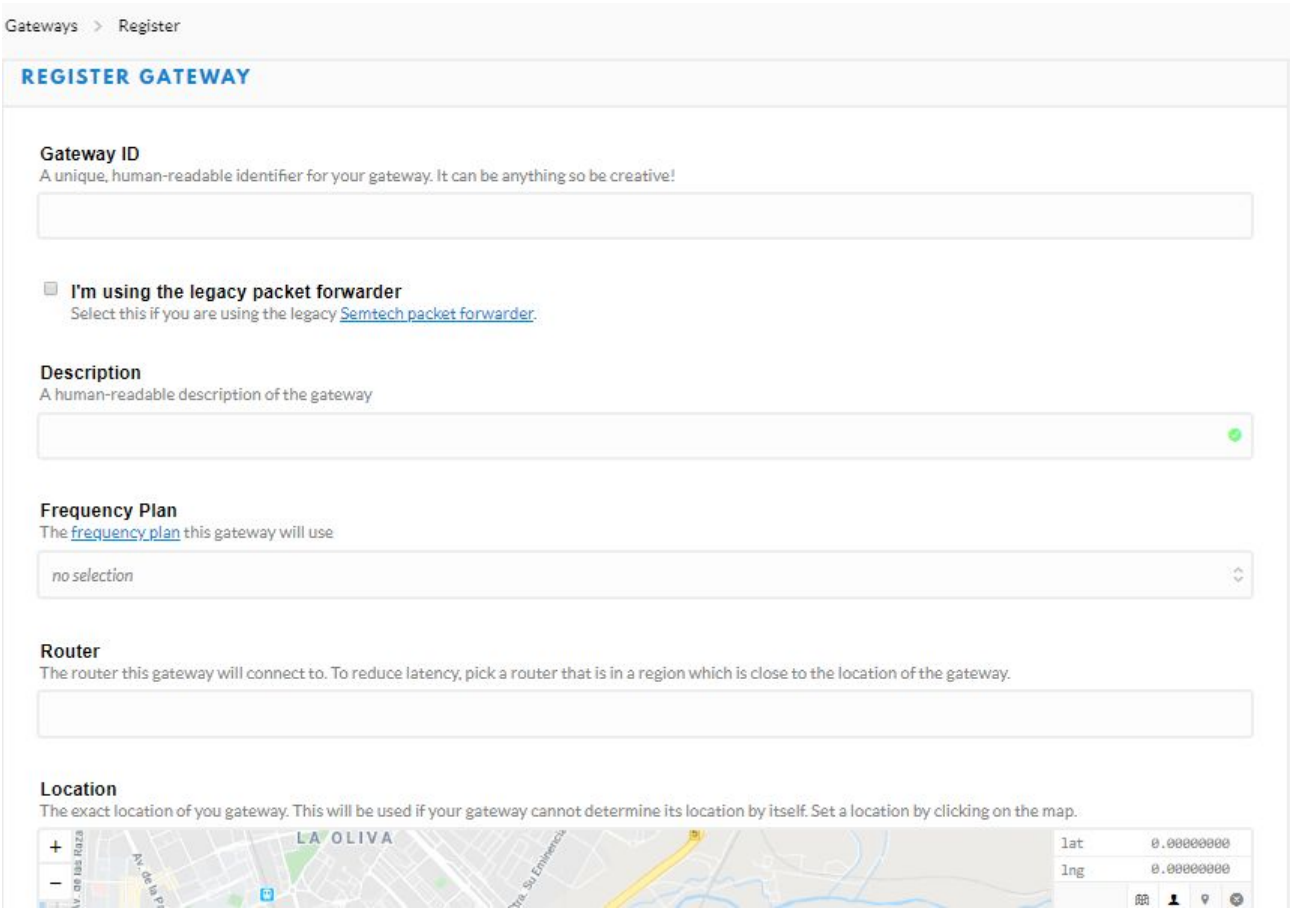

Figura 19. Pestaña de TTN para registro de gateway

2.2.2.2 Application

En el apartado de Applications, se añaden las aplicaciones a la red de TTN, y dentro de ellas estarán los devices. La aplicación se puede crear tanto a partir de APIs de TheThingsNetwork y diferentes SDKs como a través de la web. Internamente, una aplicación en TTN tendrá una API asociada a los datos y eventos recibidos de los devices y los datos enviados desde la aplicación a los devices, y esta API se utiliza normalmente bajo el protocolo *machine to machine* (M2M) MQTT. Además tendrá una API de manejo de aplicación y dispositivos, que se utiliza normalmente bajo HTTP. Al más bajo nivel (sin SDKs ni integraciones), estas APIs se comunican con la API del manejador de TTN, y su estructura sería la siguiente:

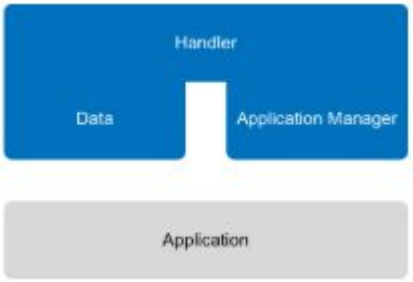

Figura 20. Estructura para el manejo de la aplicación

En el caso de la configuración desde la plataforma web, se utiliza la integración asociada a la plataforma, tal y como se explica en la siguiente imagen:

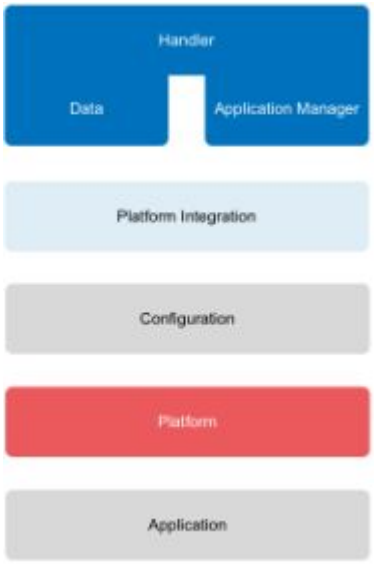

Figura 21. Estructura para el manejo y configuración de la aplicación y las integraciones desde TTN

2.2.2.3 Device

El dispositivo o *device* es el punto final de la red LoRaWAN. Este suele incorporar algún sensor y/o programa de un SDK como, por ejemplo, Arduino, para enviar datos y/o recibir datos desde la aplicación de TTN. Este dispositivos tendrá que tener un módulo de radio LoRa para conseguir enviar los datos hacia los Gateways, y al ser un módulo de muy poco tamaño, puede ser integrado en dispositivos verdaderamente pequeños.

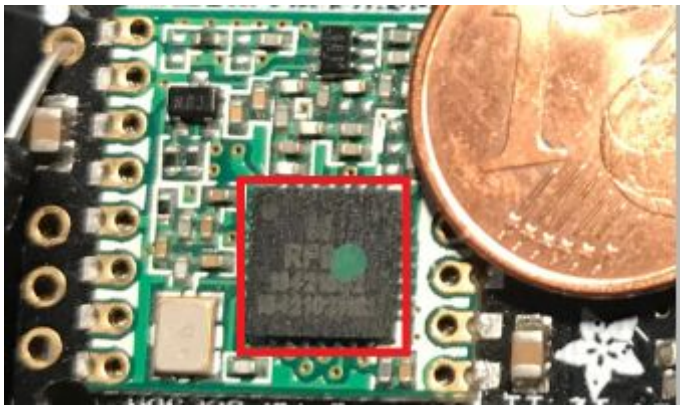

Figura 22. Transceptor LoRa

2.2.3 Microcontrolador Adafruit Feather m0 RFM9x LoRa Radio

Este microcontrolador de la empresa Adafruit es una solución correcta para este proyecto ya que cuenta con un transceptor de radio LoRa y su tamaño y peso son óptimos para su instalación en un espacio reducido (medidas: 51mm x 23mm x 8mm, peso: 5.8g).

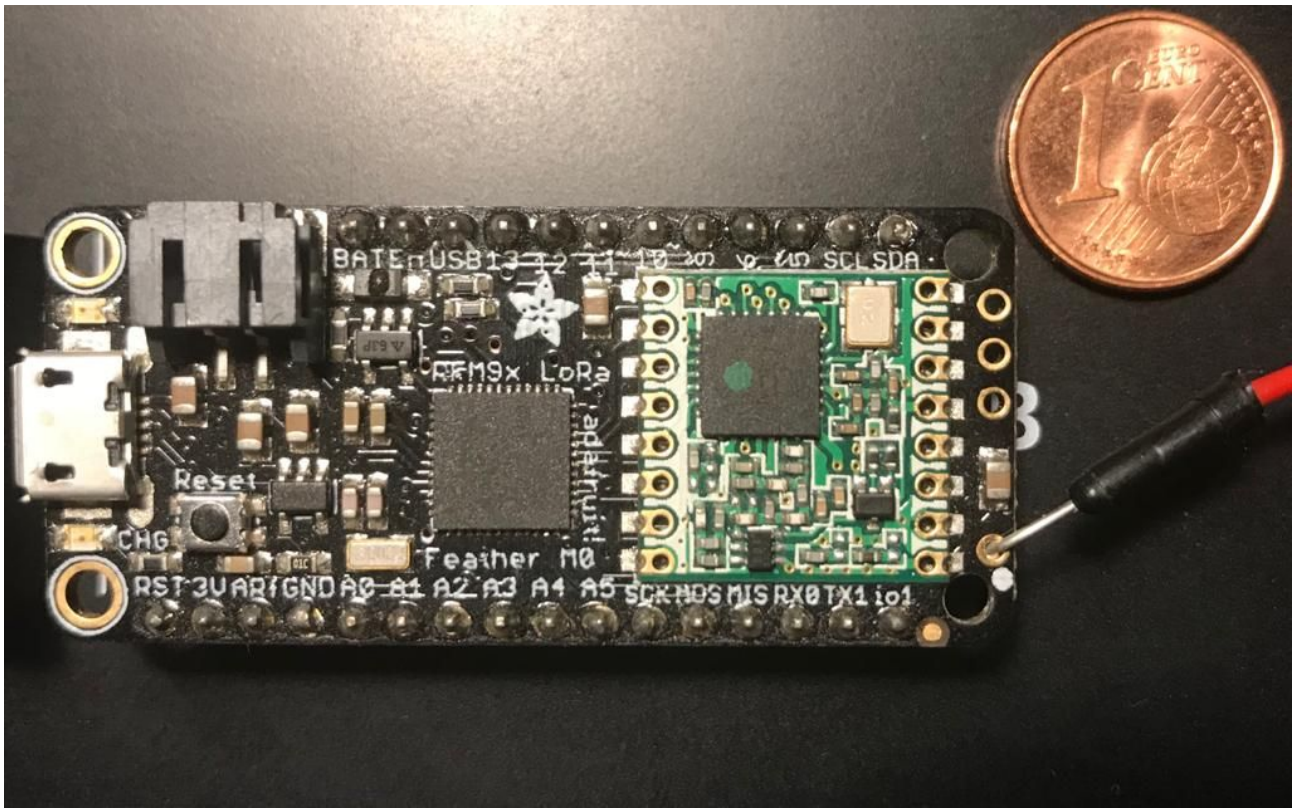

Figura 23. Microcontrolador Adafruit Feather m0 RFM9x LoRa Radio

El esquema de pines se muestra en la siguiente imagen:

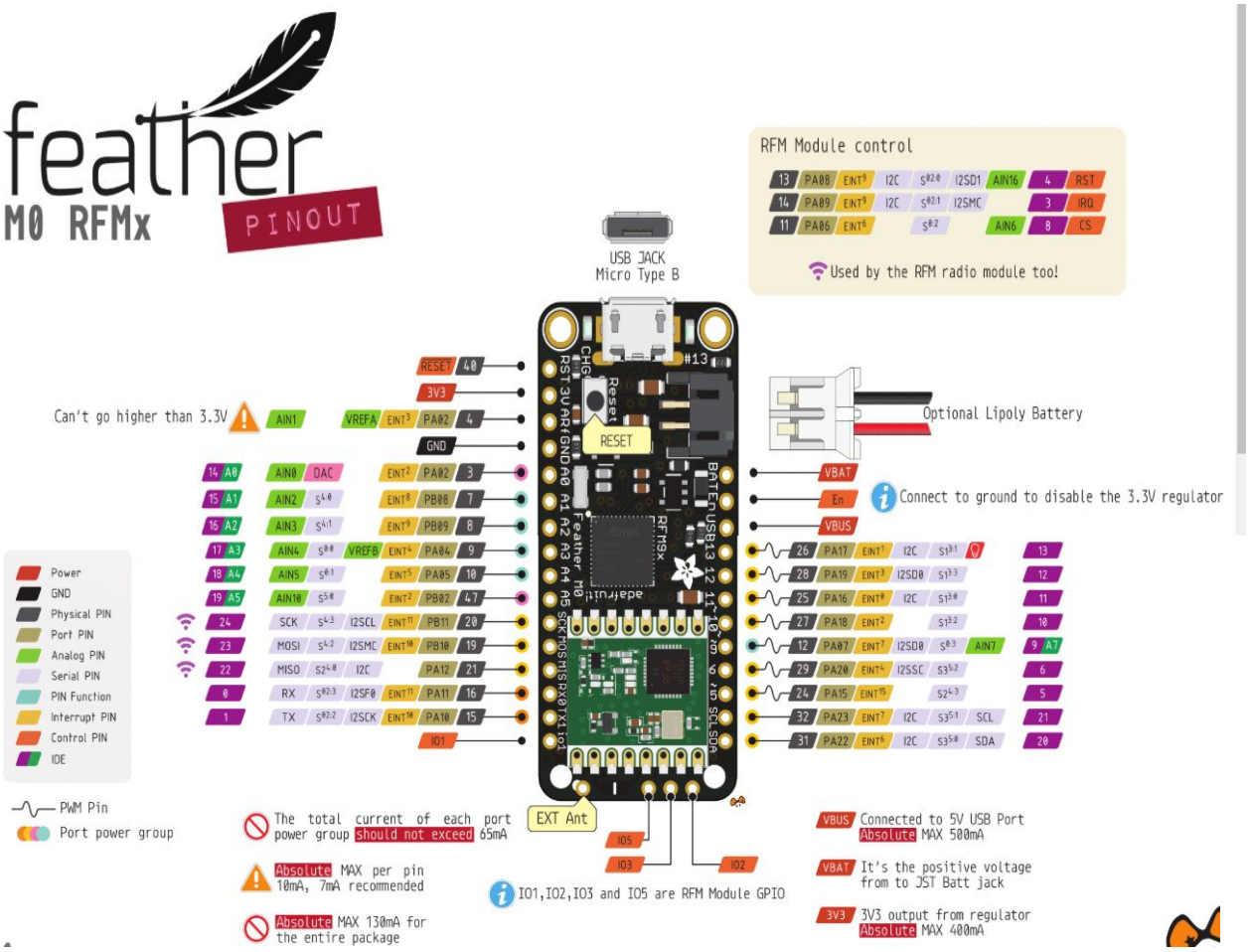

Figura 24. Esquema de pines del Adafruit Feather m0 RFM9x LoRa Radio

Este microcontrolador, como se muestra en el esquema, tiene pines digitales para comunicación i2s que serán necesarios en el montaje del sistema, ya que el sensor acústico será de estas características como se verá más adelante.

2.2.3.1 Procesador ATSAMD21G18 ARM Cortex M0

Este procesador funciona a una velocidad de reloj de 48 MHz y con lógica a 3.3 V. Cuenta con una memoria FLASH de 256 KB y una memoria RAM de 32 KB. El esquema de sus pines es el siguiente:

4.2 SAM D21G

4.2.1 QFN48 / TQFP48

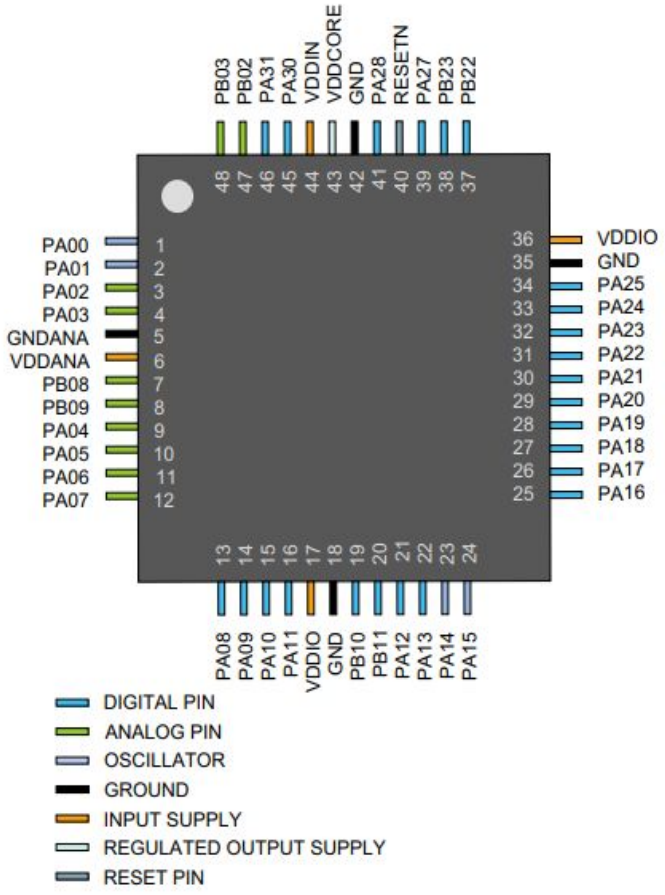

Figura 25. Esquema de pines del procesador ATSAMD21G

Y se solda en esta parte de la placa:

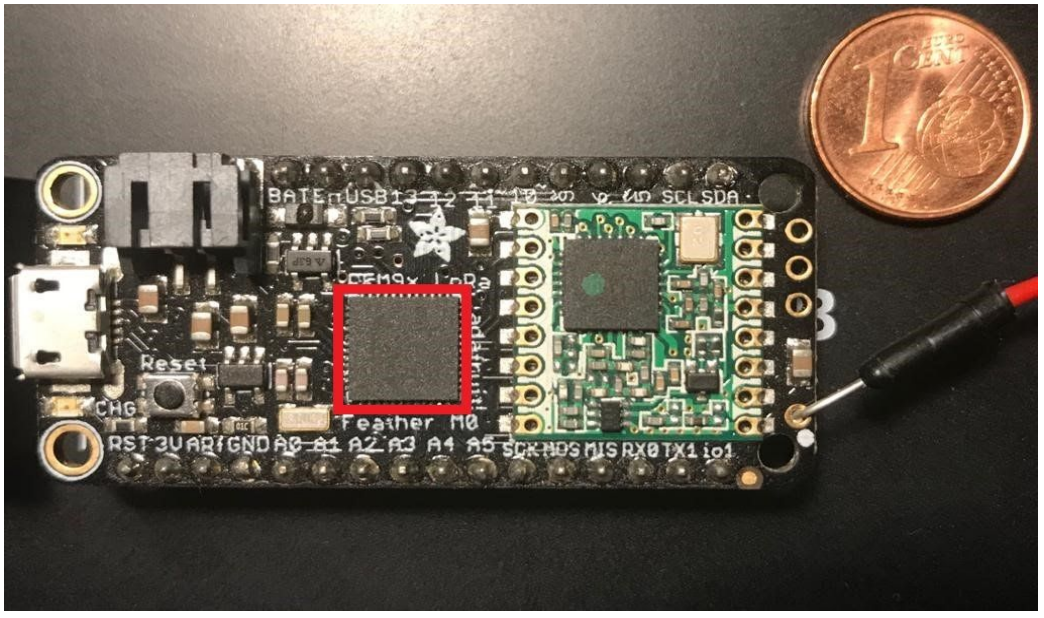

Figura 26. Módulo ATSAMD21G18 ARM Cortex M0 en la placa

2.2.3.2 Módulo de radio RFM9x LoRa

Lo que realmente interesa en este microcontrolador es su módulo de radio Long Range (LoRa) que transmite en la banda de los 868/915Mhz gracias al transceptor Semtech SX1276 basado en LoRa. Tiene una potencia de salida desde los +5 DBm hasta los +20 DBm, con una intensidad de 120 mA transmitiendo a +20 DBm de salida. Cuando está en modo sueño llega a una intensidad mínima de unos 300 uA, y cuando está escuchando para recibir por LoRa llega a unos 40 mA.

En la siguiente imagen se muestra dónde va soldado el transceptor Semtech SX1276 dentro del módulo RFM9x LoRa:

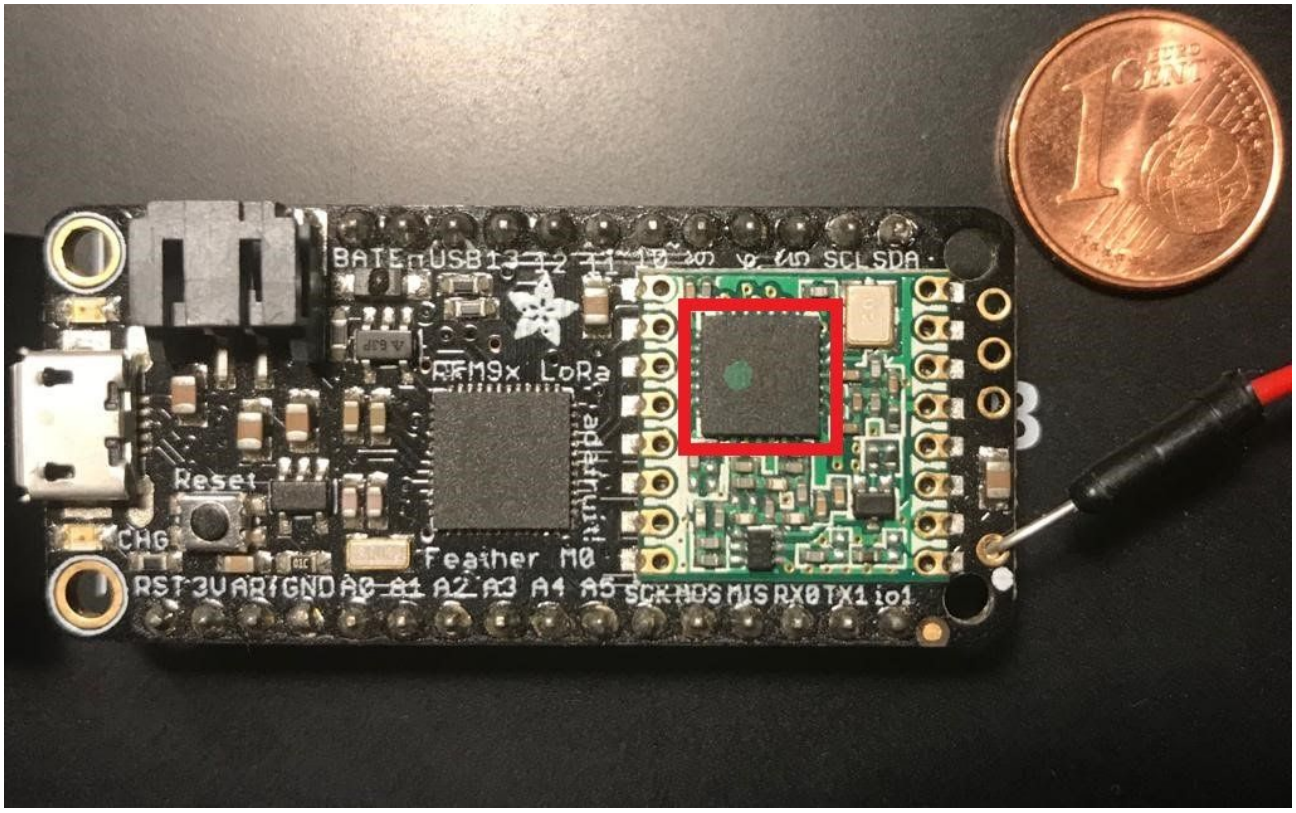

Figura 27. Transceptor LoRa Semtech SX1276

En las siguientes imágenes se muestra la arquitectura de pines de la semtech SX1276 y su significado [6]:

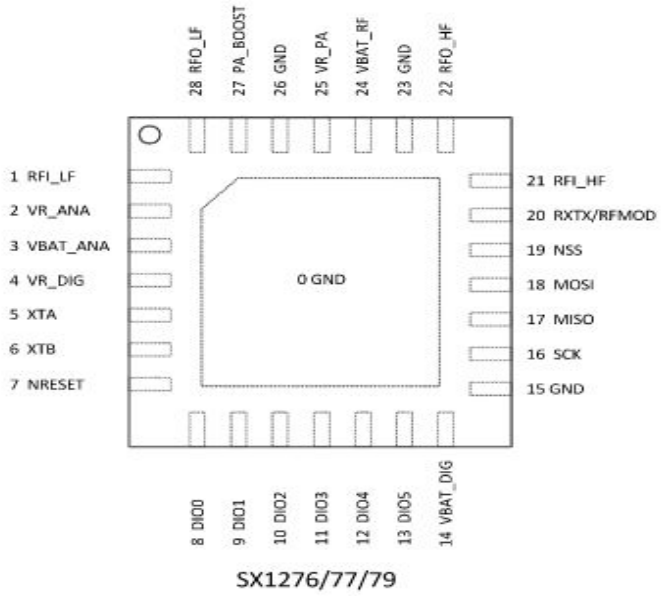

Figura 28. Arquitectura de pines del módulo Semtech SX1276

	SX1276/77/79/(78)	SX1276/77/79/(78)	SX1276/77/79/(78)
0	GROUND	-	Exposed ground pad
1	RFI_LF	I	RF input for bands 2&3
2	VR_ANA	-	Regulated supply voltage for analogue circuitry
3	VBAT_ANA	-	Supply voltage for analogue circuitry
4	VR_DIG	-	Regulated supply voltage for digital blocks
5	XTA	I/O	XTAL connection or TCXO input
6	XTB	I/O	XTAL connection
7	NRESET	I/O	Reset trigger input
8	DIO0	I/O	Digital I/O, software configured
9	DIO1/DCLK	I/O	Digital I/O, software configured
10	DIO2/DATA	I/O	Digital I/O, software configured
11	DIO3	I/O	Digital I/O, software configured
12	DIO4	I/O	Digital I/O, software configured
13	DIO5	I/O	Digital I/O, software configured
14	VBAT_DIG	-	Supply voltage for digital blocks
15	GND	-	Ground
16	SCK	I	SPI Clock input
17	MISO	O	SPI Data output
18	MOSI	I	SPI Data input
19	NSS	I	SPI Chip select input
20	RXTX/RF_MOD	O	Rx/Tx switch control: high in Tx
21	RFI_HF (GND)	I (-)	RF input for band 1 (Ground)
22	RFO_HF (GND)	O (-)	RF output for band 1 (Ground)
23	GND	-	Ground
24	VBAT_RF	-	Supply voltage for RF blocks
25	VR_PA	-	Regulated supply for the PA
26	GND	-	Ground
27	PA_BOOST	O	Optional high-power PA output, all frequency bands
28	RFO_LF	O	RF output for bands 2&3

Figura 29. Significado de pines del módulo Semtech SX1276

Y en el siguiente diagrama de bloques se explican los procesos entre los diferentes bloques de pines asociados a la recepción y transmisión de señal y a la comunicación SPI.

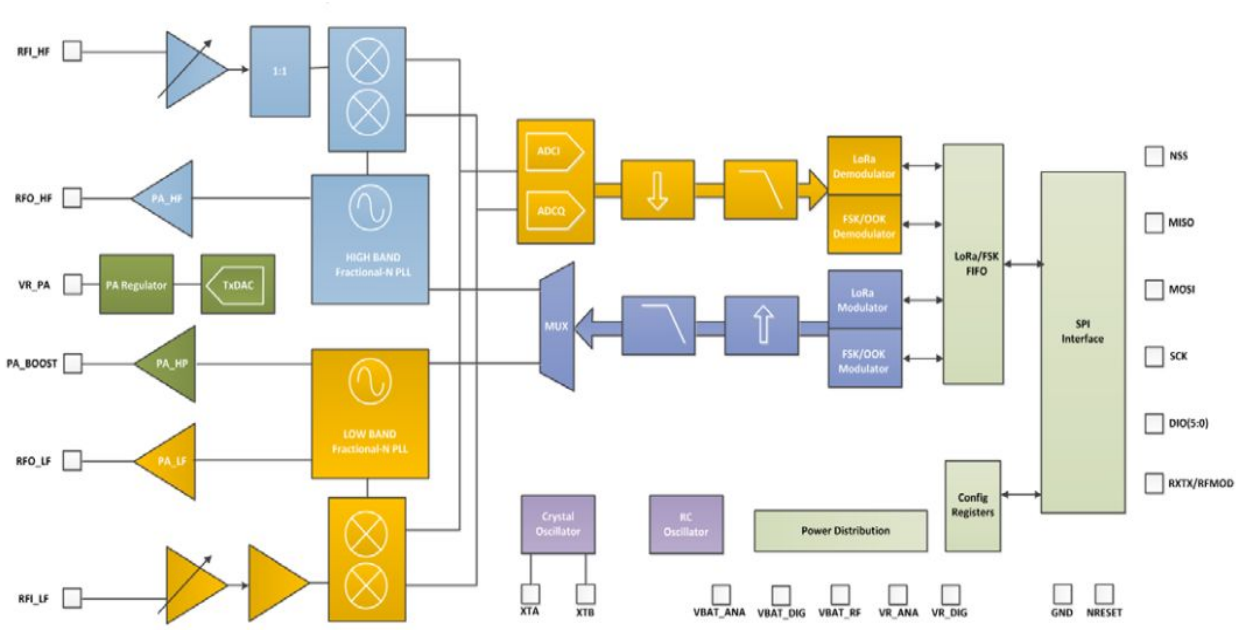

Figura 30. Procesos entre pines del Semtech SX1276

2.2.3.3 Solución clónica Feather m0 RFM9x LoRa Radio

El problema del microcontrolador de Adafruit es el precio, ya que el device es el elemento más caro en el sistema más simple posible (con un gateway de LoRaWAN ya funcionando y dando cobertura en la zona del cliente/dispositivo).

Para abordar este problema, desde la empresa se desarrolló un microcontrolador clónico a partir del PCB original de la Adafruit Feather m0 RFM9x LoRa Radio, que es público[5] y se muestra en la siguiente imagen:

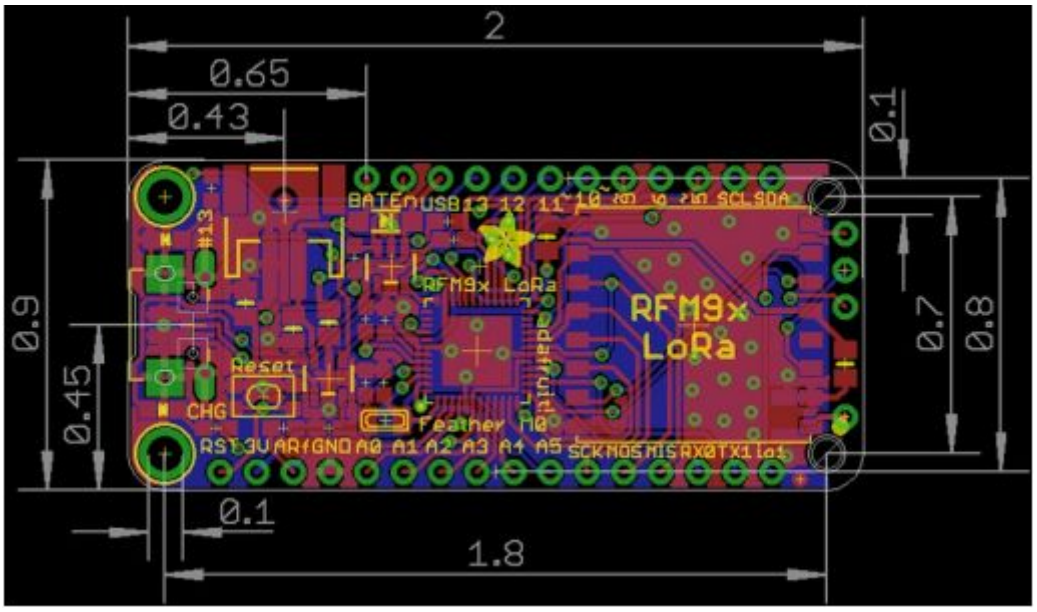

Figura 31. PCB de la Adafruit Feather m0 RFM9x LoRa Radio

Y soldando con precisión los pequeños pines de los módulos a la placa, esta queda de la siguiente manera:

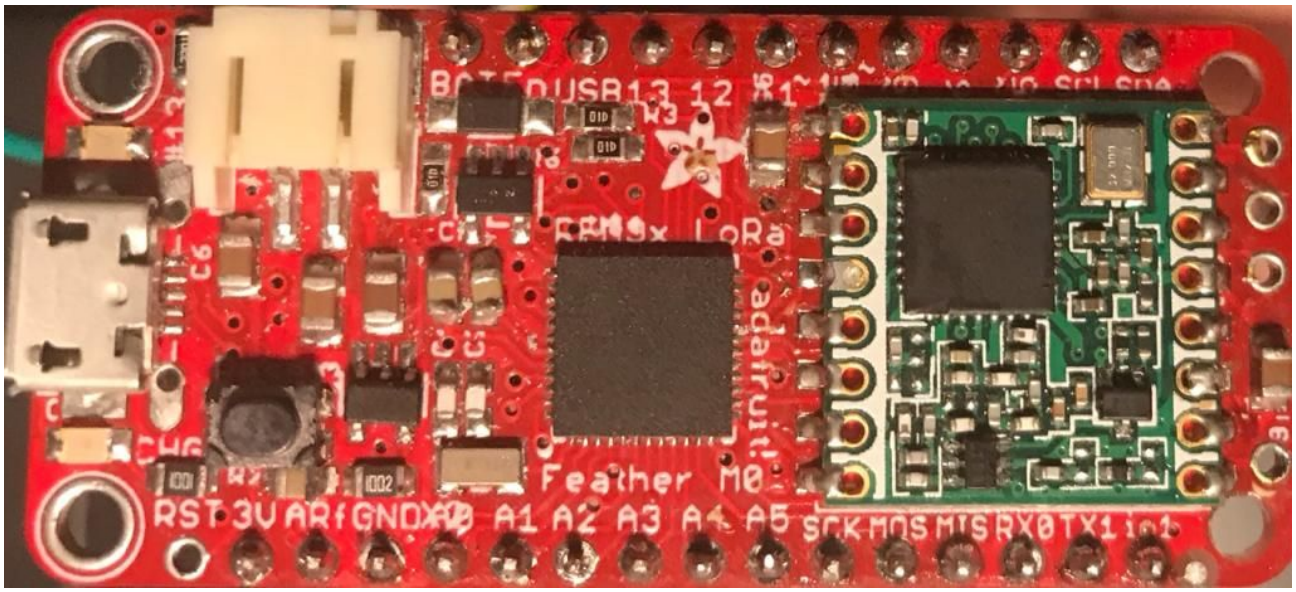

Figura 32. Placa clónica de la Adafruit Feather m0 RFM9x LoRa Radio

2.2.3.4 Diferencias y ventajas

La principal diferencia en esta solución, es el precio, ya que el de Adafruit cuesta 32€ y el clónico entre 6€ y 10€, dependiendo de la cantidad de PCB y módulos que la empresa encargue. Y para la empresa, esto conlleva una gran diferencia ya que ofrece una solución entre 22€ y 26€ más barata y se obtiene un margen de ventas amplio.

Al tener los mismos módulos que la Adafruit, su funcionamiento y arquitectura no varían, y a la hora de sincronizarla con el IDE de Arduino se verificará.

2.3. Escalabilidad y costes

TTN proporciona la red necesaria para la aplicación a partir del *gateway,* donde la escalabilidad es casi ilimitada (la del núcleo de la red y la red de retorno del 3G y Ethernet). La parte importante donde estudiar la escala del proyecto es desde los sensores hasta el *gateway,* donde el número de sensores que un solo *gateway* puede atender es de 8 (tantos como canales de subida tiene LoRaWAN). Sabiendo que estos sensores mandan información cada 5 minutos, el problema fundamental de escalabilidad se centra en los *gateways,* ya que son los dispositivos que más falta harían en la ciudad para dar señal en cualquier parte y el componente más caro en este sistema de comunicación. Por esta razón, la escalabilidad del proyecto se resume en la inversión de *gateways* LoRaWAN, ya sea por parte de empresas públicas, privadas o de la comunidad ciudadana, que por interés propio y por el bajo coste de estos dispositivos (desde los 20€), ahora mismo es el mayor inversor en muchas ciudades donde este concepto no está aún desarrollado.

Este proyecto se realiza con la idea de reducir lo máximo posible el coste del producto final, tanto del software como del hardware, sin perder la escalabilidad que nos proporciona la infraestructura de TTN. El coste se basará principalmente en el hardware, ya que el software utilizado para el desarrollo del producto será gratuito y la infraestructura de TTN también (excepto el *gateway*, como viene comentado en el párrafo anterior).

2.4. Proyectos de IoT sobre redes LoRaWAN

Hay un gran número de proyectos en marcha basados en LoRaWAN, tanto en el ámbito de *smart city* como en otros muchos. Por ejemplo, en agricultura, en gestión de construcción de edificios (*smart buildings*), en medición inteligente de electricidad, en medición del ambiente, en medición médica de pacientes y sistemas de alto riesgo, en los aparatos domésticos (*smart home*), en control industrial, etc. Donde todos estos ámbitos encuentran soluciones gracias a las ventajas del IoT y el protocolo LoRaWAN.

A continuación se muestran casos de proyectos en marcha, en concreto, sobre el tema de smart cities [10] que es sobre lo que se basa este TFG:

- Implementación de la red LoRaWAN para complementar el Wi-Fi en la ciudad de Seúl:

Figura 33. Portada del proyecto LoRaWAN complements citywide Wi-Fi network

Según el gobierno de Seúl, la Red Inteligente de Seúl, o "S-Net", proporcionará a todos los ciudadanos acceso a Internet mediante el despliegue de una nueva infraestructura de *smart city* ultraconectada. Los objetivos principales del proyecto S-Net son el establecimiento de una red municipal de banda ancha, el despliegue de redes Wi-Fi gratuitas y una mayor integración de la infraestructura de IoT basada en dispositivos LoRa. Para 2022, la ciudad planea desplegar hasta 1000 *gateways* basados en LoRaWAN para complementar el Wi-Fi municipal y proporcionar la base para las innovadoras aplicaciones de ciudad inteligente de próxima generación de Seúl. Estas aplicaciones incluyen estacionamiento inteligente, soluciones de alumbrado público interconectado que informan automáticamente a la policía al detectar una emergencia y una aplicación de geolocalización que utiliza la información de seguimiento de los niños y personas que sufren demencia para mantener a estos ciudadanos seguros. Además, el gobierno de la ciudad espera aumentar su recopilación y utilización de datos urbanos para abordar mejor las necesidades de sus ciudadanos.

- Semtech y X-TELIA implementan la solución LoRa para programas de horario de autobuses inteligentes:

Figura 34. Portada del proyecto X-TELIA implements LoRa for smart bus schedule signs

X-TELIA implementó una pantalla de visualización basada en LoRa que muestra los próximos horarios de salida de autobuses en las paradas. Las pantallas funcionan con energía solar y funcionan en la red pública LoRaWAN de X-TELIA. X-TELIA está actualizando las pantallas para mostrar los retrasos del autobús en tiempo real, aprovechando la tecnología LoRa para el seguimiento y los cambios de horario actualizados. El programa se encuentra entre los primeros en Canadá en utilizar la tecnología LoRa para el tránsito inteligente, incluido el seguimiento y la programación.

- La tecnología LoRa de Semtech mejora el sistema de gestión de residuos y congestión de estacionamiento de la Universidad de Canadá:

Figura 35. Portada del proyecto LoRa improves UBC´s parking & waste bin management

En todo el campus de la UBC (University of British Columbia), los sensores que utilizan los dispositivos LoRa de Semtech y la tecnología de radiofrecuencia inalámbrica (LoRa) se instalan en contenedores de basura y se incrustan en el pavimento para monitorear los datos en tiempo real para obtener eficiencia y reducir los costos operativos. Ambas soluciones de IoT están conectadas a la red basada en LoRaWAN de eleven-x y proporcionan datos en tiempo real a los administradores de UBC. En el sistema de gestión de residuos, las alertas se activan y se envían al personal de la UBC cuando se alcanza un nivel de residuos predeterminado por contenedor, lo que indica que se requiere la eliminación de residuos. Los niveles se supervisan en tiempo real a través de aplicaciones de terceros habilitadas para la nube, optimizando la respuesta de mantenimiento. La solución de estacionamiento inteligente de la UBC está diseñada para verificar qué espacios de estacionamiento se usan más y, específicamente, cuántos espacios están disponibles para personas con discapacidades. La Universidad planea utilizar los datos recopilados por los sensores de estacionamiento para evaluar la expansión de las áreas de estacionamiento para facilitar el acceso y aumentar la eficiencia.

- Semtech y Ubicquia iluminan las calles con una solución IoT basada en Smart Grid LoRa:

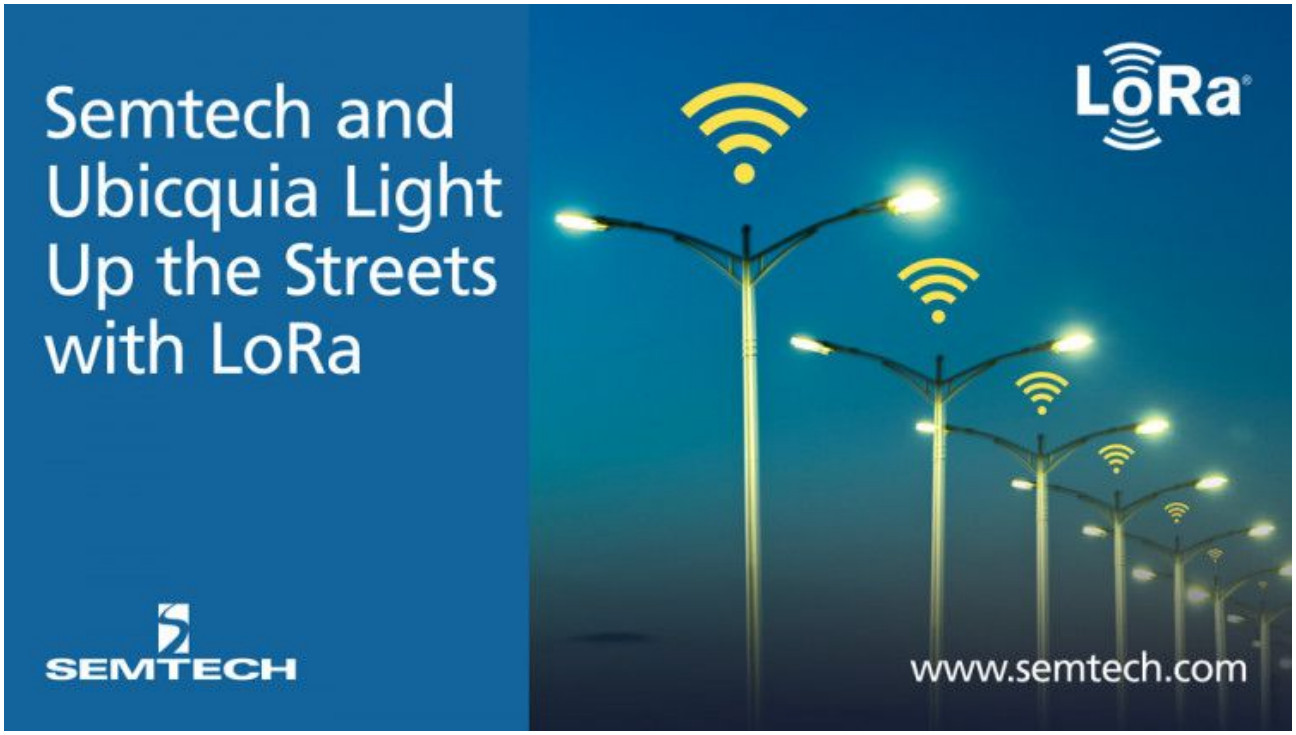

Figura 36. Portada del proyecto Semtech and Ubicquia ligth up the streets with LoRa

Las farolas representan una gran parte del presupuesto de energía asignado de una ciudad, en algunos casos alcanzando más del 40%. Los estudios sobre redes de energía estiman que una solución de Internet de las Cosas (IoT) de red inteligente que utiliza sensores, puertas de enlace y bombillas LED proporcionaría grandes ganancias financieras, así como datos valiosos. El nuevo módulo de *gateway* basado en LoRa de Ubicquia permite que la farola de Ubicell actúe como puerta de enlace y como cliente, manteniendo su funcionalidad central que incluye control de luz avanzado, medición de potencia de grado de servicio, detección de inclinación y vibración, conexión a sensores de terceros e integración de red.

- La tecnología LoRa de Semtech y la red basada en LoRaWAN de Senet con sensores de inundación para monitorear los niveles de agua:

Figura 37. Portada del proyecto LoRa technology leveraged in flood sensors to monitor water levels

Las soluciones de Green Stream utilizan la tecnología LoRa. Las soluciones de monitoreo de inundaciones de Green Stream están diseñadas utilizando sensores ultrasónicos comerciales y listos para usar y *gateways* habilitados para LoRa fáciles de implementar. Los datos se comunican a través de una red basada en LoRaWAN proporcionada por Senet, un proveedor líder de plataformas de servicios LoRaWAN basadas en la nube que permiten la construcción y la gestión de la conectividad IoT bajo demanda. Los sensores de inundación basados en Green Stream LoRa son autónomos y no requieren alimentación externa ni conexión de red por cable. Cada sensor es una unidad autónoma, resistente a la intemperie y alimentada por energía solar que viene con un soporte de montaje universal y un brazo de extensión. Estos sensores son lo suficientemente pequeños como para instalarse en la parte superior de los cruces peatonales, postes de electricidad o puentes. La robusta puerta de entrada del sensor se coloca sobre una masa de agua (lago, río, canal,...) o sobre tierra firme.

3 Diseño de la solución

3.1. Estudio del nivel de ruido urbano

Para saber en torno a qué niveles de ruido estudiar la fiabilidad del sensor, se estimará un mínimo y máximo posible en un entorno urbano.

.

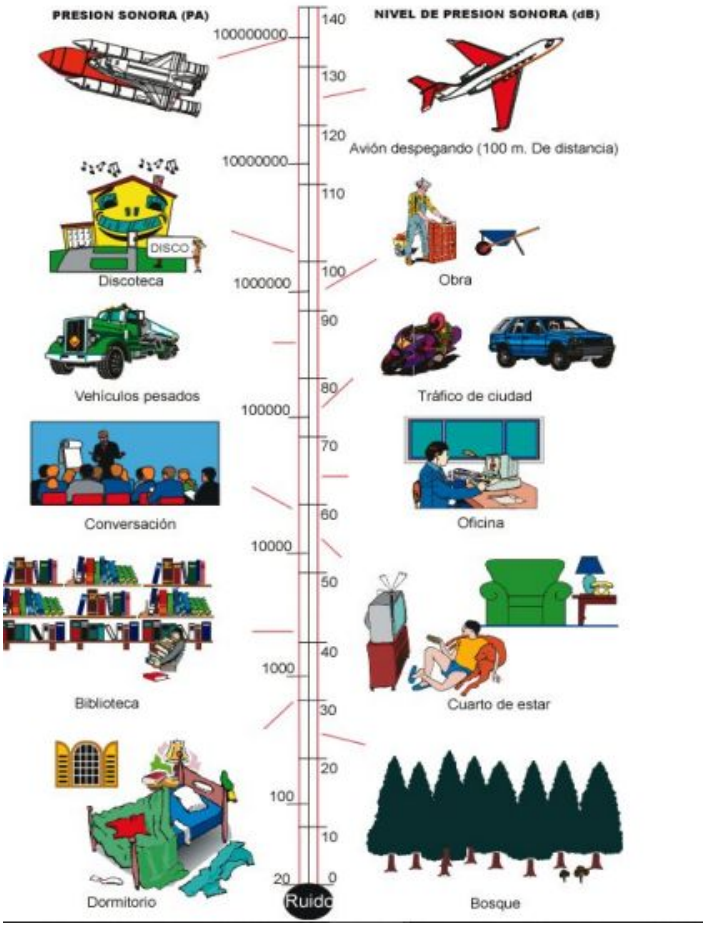

Figura 38. Ejemplos de nivel de ruido

Según el esquema mostrado [8], la media de ruido urbano podría oscilar en torno a los 60 decibelios, dependiendo del momento del día y el tráfico de vehículos en una determinada vía pública. En lo más alto se acotará el nivel al equivalente en una obra, que está sobre los 90 decibelios (umbral del dolor), y el nivel más bajo estará sobre los 30 decibelios cuando no hay apenas ruido en la calle, semejante al ruido en una habitación o en el bosque.

Por lo tanto, normalmente habrá un ruido de entre 40 decibelios (que puede ser el ruido de madrugada, con poco tráfico) y 70 (que puede ser el ruido del tráfico de vehículos). Suponiendo una calle de dimensiones medias, es decir, de dos carriles de tráfico en una zona sin atascos.

3.2. Funcionalidades

La funcionalidad principal del dispositivo es la medición de ruido en entornos urbanos como los mencionados en el apartado anterior. Los datos recogidos son enviados directamente a la aplicación de TTN para su visualización a tiempo real por los usuarios autorizados en TTN. Estos datos consistirán en la media, máximo y mínimo del ruido medido durante 5 minutos, donde el micrófono medirá una muestra cada segundo. Además se puede saber la geolocalización de cada dispositivo, ya que cada *device* se asocia a una localización en el proceso de registro en TTN.

El dispositivo puede estar instalado en exterior, por lo que debe estar protegido de las condiciones climáticas adversas que afecten al funcionamiento de este, sobre todo de la lluvia y la radiación solar. No se puede optar por la solución de una caja cerrada para la protección, ya que altera la señal acústica recibida por el sensor. Así que simplemente se añadirá al dispositivo una pieza de plástico en la parte superior capaz de reflejar la máxima radiación posible y que, obviamente, servirá también de protección contra la lluvia.

Este dispositivo será autónomo una vez instalado por el usuario en caso de que este lo tenga conectado a la corriente gracias a la función de recarga mediante USB. Y la batería, antena y micrófono son intercambiables por el usuario tanto para la mejora del device como para su mantenimiento. En el apartado 4 se estudiará la autonomía en caso de ser alimentado mediante pilas o batería LiPo.

El dispositivo se enlazará a TTN mediante las claves de la aplicación alojada en la web. En la implementación de la solución se explica cómo la librería de comunicación (LMiC) utiliza estas claves con la aplicación de TTN para dar de alta un dispositivo. Una vez dado de alta el dispositivo, se podrán empezar a visualizar los datos medidos por este en la interfaz web de la aplicación de TTN, y estos datos serán reenviados a una base de datos (*data storage*) para consultas posteriores.

Además, como línea de avance de este proyecto, los datos de TTN serán fácilmente manipulables para su envío a aplicaciones propias de la empresa (por ejemplo ThingsBoard, para la representación gráfica de los datos) gracias a las extensiones de TTN que sirven concretamente para esto.

3.3. Costes

En este apartado se mostrarán los precios de cada componente del producto, elegidos uno a uno para reducir el precio del proyecto lo máximo posible. El coste total del montaje de este dispositivo será el de software más el de hardware, y como se usa únicamente software libre, el coste será el de hardware. Se estimará el precio de cada componente, ya que el mercado rebaja el precio unitario del producto si se compra en mayor cantidad. Y como en este punto aún no se conoce la escala del proyecto para su venta al cliente, solo se comprará lo necesario para el montaje de un solo dispositivo de pruebas.

Debido a que el mayor coste es el del microcontrolador Adafruit, con un coste de alrededor de 30 euros, se opta por la solución de diseñar un prototipo clónico a este. Esto reducirá considerablemente el coste, de los 30 que cuesta el original a 6 que cuesta el clónico.

A continuación se listará el coste estimado de los componentes:

Microcontrolador_clónico:6€
Cables:2€
Antena: 1,50€
Batería LiPo 2500mAh: 9€
Sensor acústico: 1,5€

En resumen, el coste unitario del dispositivo sería de 11€ si se comercializase sin batería o de 20€ si se suministrase al cliente una de 2500mAh. Por lo que se ajusta bastante bien al precio que la empresa estimaba que iba a costar.

3.4. Arquitectura de la solución

En el proyecto se usará la arquitectura de red LoRaWAN para la comunicación entre el dispositivo con el sensor y el *Gateway* de TTN. El dispositivo contará con un módulo capaz de transmitir mensajes de radio LoRaWAN, por lo que se elige el microcontrolador Adafruit Feather m0 RFM9x LoRa Radio que cuenta con este módulo. El *Gateway* reenviará los datos sobre Ethernet, WiFi o red móvil (3G, 4G, ...) para su visualización en la aplicación correspondiente de la página web de TTN. Los detalles sobre los procedimientos y componentes internos de la red LoRaWAN están descritos en el apartado 2.2.1 y la arquitectura que ofrece TTN se explica en el apartado 2.2.2.

En el siguiente esquema se muestra la arquitectura general del proyecto.

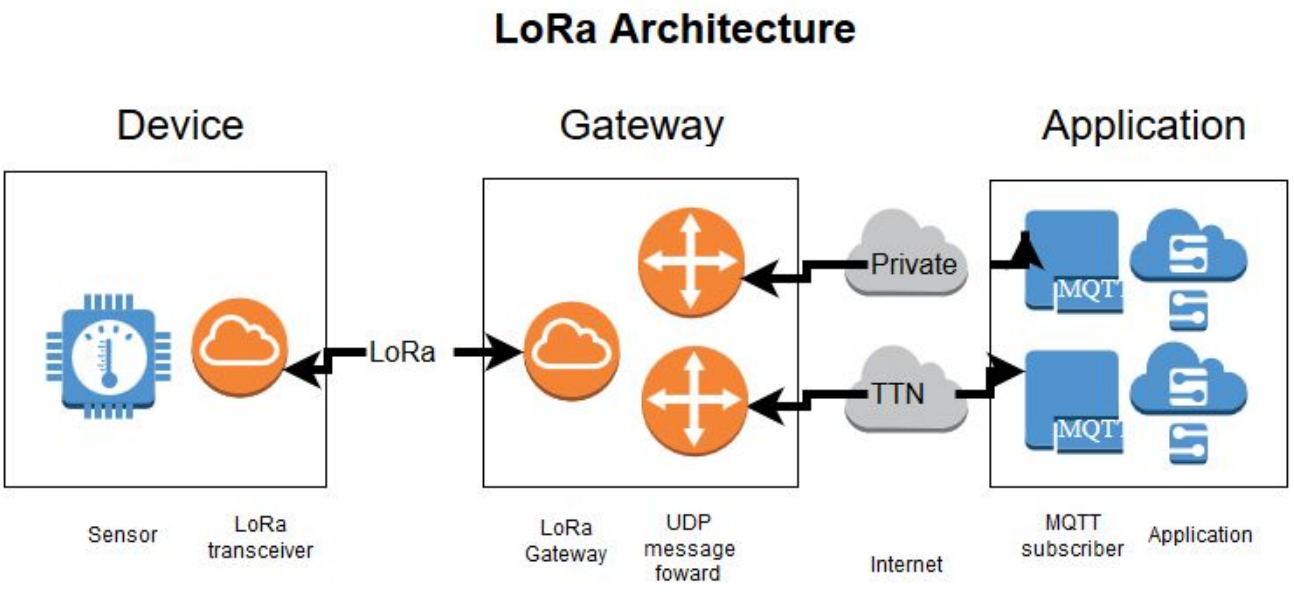

Figura 39. Esquema de arquitectura de la solución

El *device* contará con un sensor, que en este caso será un sensor acústico, capaz de medir datos del entorno para después ser enviados vía radio (LoRa). A continuación se muestra un diagrama de bloques que muestra de forma esquemática cómo se comporta la aplicación del *device.* La implementación se muestra en el apartado 4.

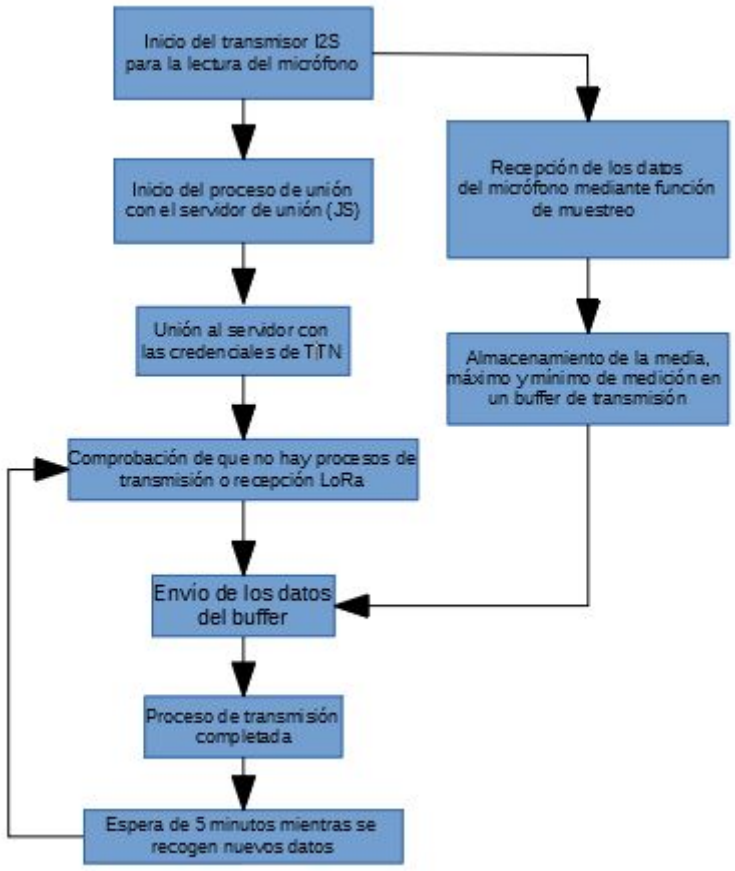

Figura 40. Diagrama de bloques de la aplicación

El transceptor LoRa se encargará de enviar estos datos a través de la red LoRa y hace posible la comunicación con el *gateway* LoRa. En este caso se usará el transceptor Semtech SX1276 y una antena Heltec capaz de enviar señales de radio a 868/915 MHz (frecuencia de radio LoRa).

El *gateway* LoRa será el componente de red encargado de encaminar los paquetes IP mediante UDP sobre el núcleo de la red. Y tras atravesar el núcleo de la red llegará mediante la red de TTN o mediante red privada a la aplicación deseada. En este caso se usa la red TTN.

En la aplicación a la que llegarán los datos de cada sensor se podrán manejar los datos o simplemente visualizarlos en la interfaz web de TTN. Gracias al protocolo de conectividad MQTT se publican las activaciones de los *devices* y los mensajes de la aplicación correspondiente, ya sea de una aplicación privada o, en el caso de esta solución, de la aplicación creada mediante TTN.

3.5. Plan de pruebas

El plan de pruebas consistirá, en primer lugar, en la verificación de la implementación diseñada. Se verificará que la implementación utilizada sirva para controlar los pines del microcontrolador para que funcione según el diseño de la implementación. Y tras las pruebas a nivel de hardware se verificará la conexión con la aplicación creada en TTN, y a su vez con la extensión de TTN (DataStorage) donde se guardará la información en una base de datos.

Una vez comprobados los puntos anteriores se empiezan a hacer las pruebas con el micrófono para validar que este recoja el nivel de ruido ambiental correcto, tanto en interior como en exterior. Y una vez se ha verificado totalmente la funcionalidad del dispositivo se diseña el plan de pruebas para las verificación en entorno urbano con la integración TTN Mapper, que consistirá en la medición del ruido en diferentes puntos de la ciudad y que estas ubicaciones queden registradas en la aplicación.

4 DESARROLLO DE LA SOLUCIÓN

4.1. Dispositivo

En este apartado se explica el desarrollo de todo lo relacionado con el dispositivo (*device*), distinguiendo entre software y hardware.

4.1.1. Software

4.1.1.1. Arduino

En esta sección se darán los pasos a seguir para la instalación y uso básico de Arduino IDE, que es el entorno de desarrollo usado para la programación del microcontrolador.

4.1.1.1.1 Instalación de Arduino IDE

La versión usada será la 1.8.12, y será instalada desde la web oficial de Arduino: https://www.arduino.cc/en/main/software , mediante el instalador de Windows mostrado en la siguiente captura:

Figura 41. Página de descarga del instalador de Arduino

Después, solo bastará con aceptar los términos de la licencia, seleccionar, al menos, las siguientes opciones de instalación:

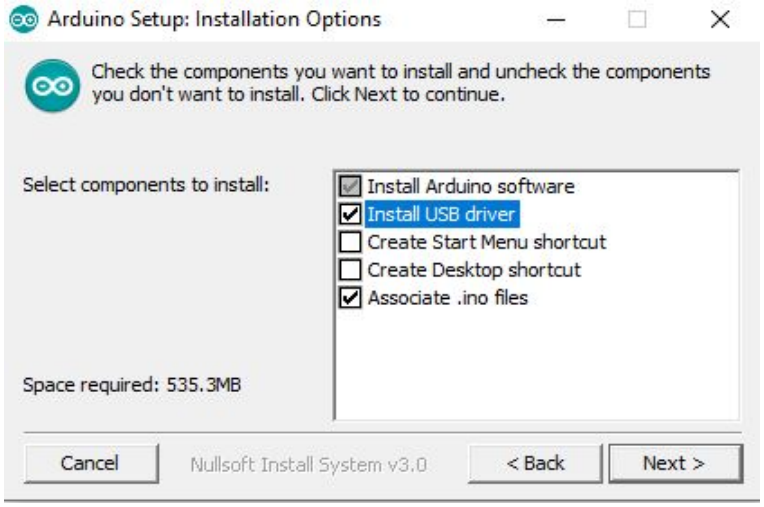

Figura 42. Opciones de instalación de Arduino

Y por último, asegurarse de que se instala el programa en la carpeta C:\Program Files (x86)\Arduino

4.1.1.1.2 Interfaz y funcionamiento básico

En la interfaz principal, la parte de "Herramientas" será desde la que se configurará el hardware a flashear, que será seleccionado en la lista desplegable "placa", también se podrá seleccionar el puerto de salida a la placa en "puerto" y abrir el monitor serie una vez la aplicación esté verificada e instalada en la placa. En la parte de "Archivos de programa." están a disposición los paquetes de librerías para su descarga o actualización.

Luego, los dos botones redondos superiores servirán para compilar el código ("verificar") y para subir el código a la placa ("subir"). Nótese que el botón de subir siempre ejecutará previamente el de verificar.

En la parte inferior está la consola donde se mostrará la salida del compilador, y debajo de la consola a la derecha se muestra el nombre de la placa y el puerto al que está conectada.

4.1.1.1.3 Estructura del proyecto de Arduino

Para comenzar, se situará el código dentro de un nuevo proyecto de Arduino. Una vez creado, dentro de la pestaña superior Archivo/preferencias, se introduce la localización donde se encontrará la carpeta de librerías, en este caso C:\Users\User\Documents\Arduino.

Figura 43. Localización del proyecto de Arduino

Dentro de la carpeta Arduino estará la carpeta de librerías, que debe de ser llamada libraries. Dentro de ella se colocarán las librerías con extensión .h , que deberán de estar contenidas en una carpeta del mismo nombre que la librería. Por ejemplo, my_credentials.h estará situada en la carpeta C:\Users\User\Documents\Arduino\libraries\my_credentials.

Además, en las preferencias hay que añadir la dirección del gestor de la placa para recoger información actualizada de esta cada vez que se abra el proyecto. Esta dirección se puede encontrar en la

web:https://github.com/arduino/Arduino/wiki/Unofficial-list-of-3rd-party-boards-support-urls#list-of-3rd-party-boards-support-urls .Mirando en la lista de esta web se identifica la URL para la placa Adafruit SAMD (Feather M0) y se añade en la casilla correspondiente de preferencias.

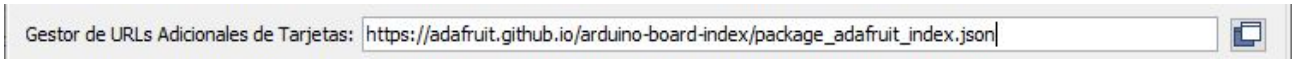

Figura 44. URL de gestor de tarjetas de Arduino

4.1.1.1.4 Gestor de tarjetas de Arduino

Para que se reconozca el Adafruit Feather M0 lo primero será instalar el paquete de tarjetas correspondiente dentro del gestor de tarjetas de Arduino (Herramientas/Placa/Gestor de tarjetas...).

Figura 45. Gestor de tarjetas

Después, solo habrá que seleccionar la tarjeta dentro de las del paquete instalado.

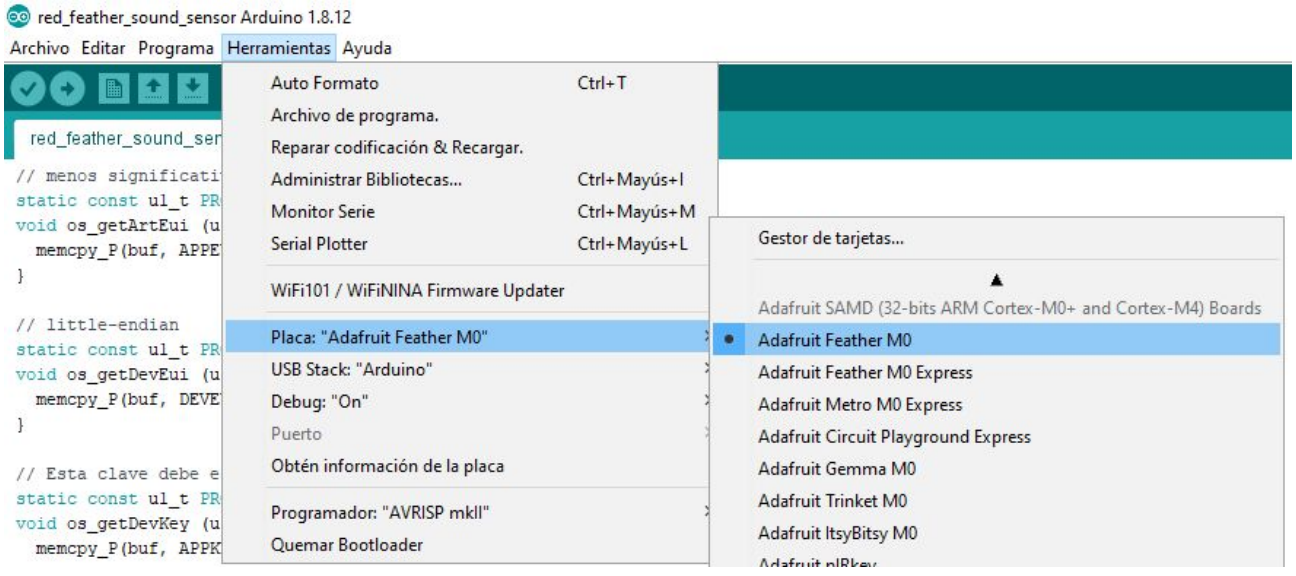

Figura 46. Selección de la placa Adafruit Feather M0

4.1.1.1.5 Instalación de librerías

Las librerías pueden ser añadidas al proyecto de dos formas, como se ha explicado en el subapartado anterior o mediante la pestaña Herramientas/Administrar Bibliotecas... del IDE. Este proyecto utilizará los dos métodos para importar las librerías. Se tendrá la librería de las credenciales de conexión de LoRaWAN en la carpeta C:\Users\User\Documents\Arduino\libraries\my_credentials y el resto de librerías serán instaladas desde el administrador de bibliotecas.

Las librerías relacionadas con el Arduino IDE y su lenguaje de programación (C) las reconocerá automáticamente el IDE al compilar sin necesidad de instalarlas manualmente. Se usarán las siguientes:

- Arduino.h: Necesario para que funcione el .cpp.
- stdio.h: Para las definiciones de macros, constantes y declaraciones de funciones de la biblioteca estándar del lenguaje de programación C (que es el lenguaje del Arduino IDE) para hacer operaciones estándar, de entrada y de salida, y además define tipos necesarios para dichas operaciones.
- math.h: Para operaciones matemáticas básicas.

Las librerías relacionadas con la conexión LoRaWAN se encuentran en el siguiente paquete de librerías:

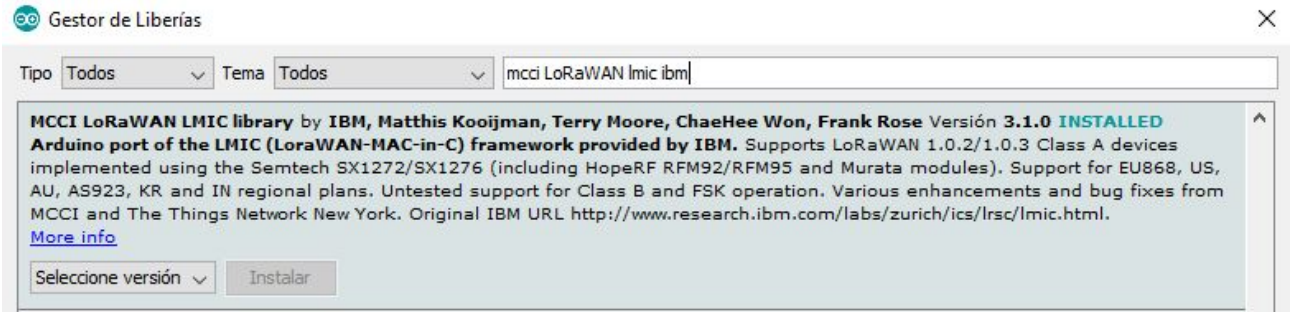

Figura 47. Instalación de librería LMiC

- lmic.h: Librería LoRaWAN-MAC-in-C, para funciones de conexión de capa de enlace LoRaWAN.
- hal/hal.h: Librería Hardware Abstraction Layer (Capa de abstracción de Hardware). Para portar fácilmente la librería LMIC a nuevas plataformas hardware.
- SPI.h: Librería de estándar de comunicaciones *Serial Peripheral Interface*. BUS para procesos de comunicación de interrupción de la placa.

El siguiente esquema de capas de protocolos muestra la interacción entre estas librerías/protocolos entre el *device/*hardware y el *gateway*.

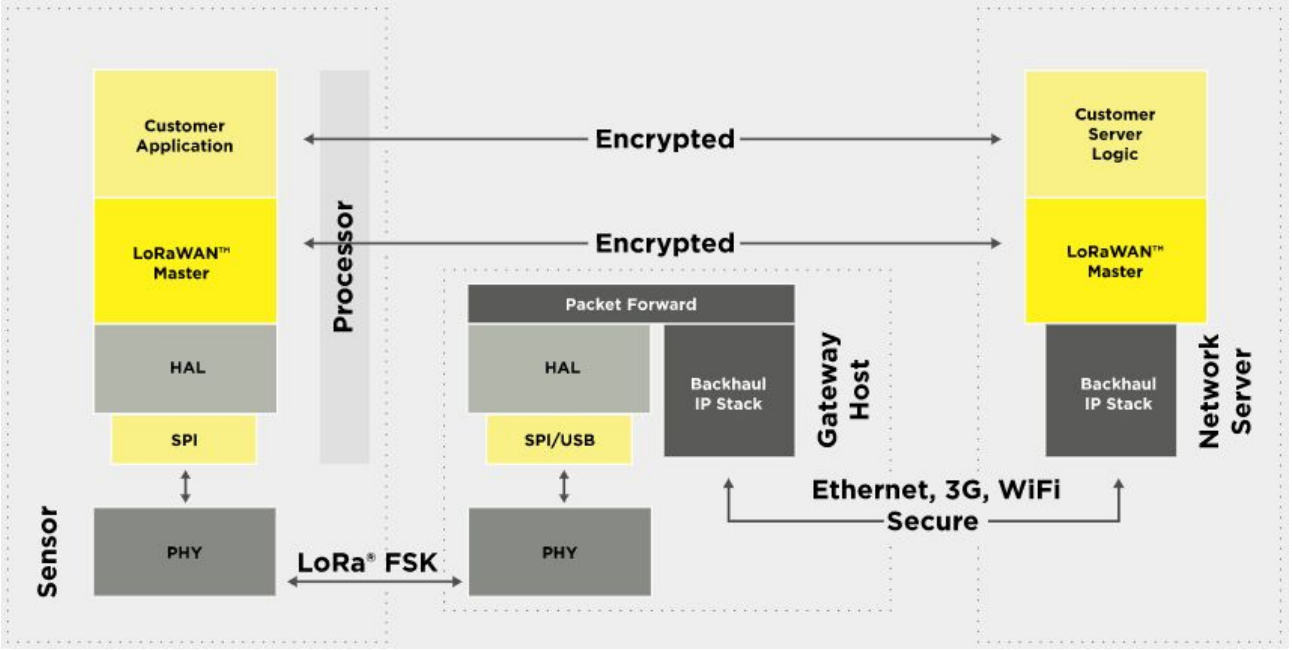

Figura 48. Interacción de las capas de protocolos/librerías entre device, gateway y network server (NS)

Para la comunicación I2S entre el Adafruit y el micrófono se utilizará la siguiente librería del gestor.

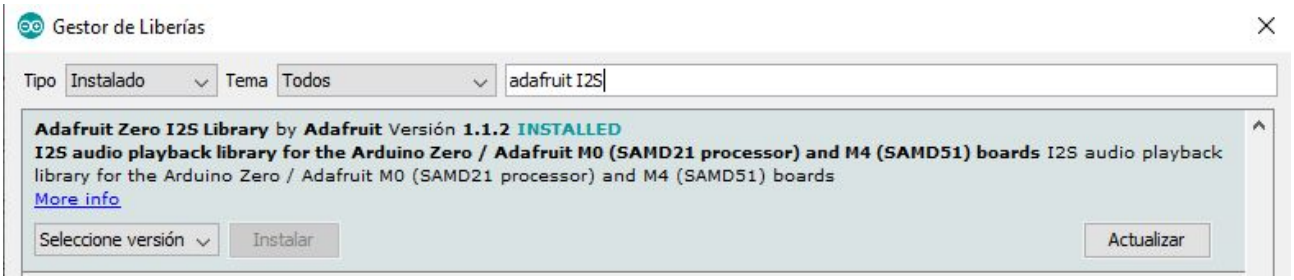

Figura 49. Instalación de librería I2S

Y, por último, para las funciones de ahorro de energía mediante *sleep mode* se utilizará la siguiente librería del gestor.

Figura 50. Instalación de librería Arduino Low Power

El código de las librerías más relevantes se encuentran en el anexo de este TFG.

4.1.1.1.6 Implementación de la solución

A continuación se observa el código del proyecto, cuyos comentarios proporcionan la descripción de los pasos seguidos. Para la realización y comprensión, se ha seguido la especificación del LMIC [9].

red_feather_sound_sensor

```
const int sample_delay = 1000;  //tiempo de muestreo en milisegundos (1 segundo)
const int N_samples = 60;  // Numero de muestras por minuto
const int sleepytime = 300000;  //5 minutos que pasara en modo sleep entre medida y medida del microfono


#include <Arduino.h>
#include <SPI.h>
#include <stdio.h>
#include <math.h>

//Librerias de LoRaWAN para conexión con TTN
#include <lmic.h>
#include <hal/hal.h>

//Libreria para uso de I2S
#include <Adafruit_ZeroI2S.h>

//Libreria para modo sleep
#include <ArduinoLowPower.h>

// Claves de acceso de LoRaWAN
#include <my_credentials.h>


/*
El comportamiento predeterminado es calcular los valores min, max y avg
de 60 muestras a lo largo de un minuto (una muestra por segundo) y luego dormir durante 5 minutos.
Eso proporciona 12 mediciones por hora, 288 por día, 2016 por semana, etc.
*/


#define SAMPLERATE_HZ 44100 //Velocidad de lectura de I2S en Hz (44100 es la típica)

// Variables para el muestreo del sonido
float maximum;// Mayor sonido medido
float maximum2;// Segundo mayor sonido medido, porque la primera medida tras despertar es erronea
float minimum;// Menor sonido medido
float average;// Media de todas las muestras

// Variables de LoRa
char TTN_response[30];// Variable para recibir datos de TTN
static osjob_t sendjob;// Variable de trabajo LMIC

// Mapeo de pines del Adafruit para LMIC
const lmic_pinmap lmic_pins = {
  .nss = 8,
  .rxtx = LMIC_UNUSED_PIN,
  .rst = LMIC_UNUSED_PIN,
  .dio = {3, 6, LMIC_UNUSED_PIN},
};
```

```
// Este EUI debe estar en formato little-endian, por lo tanto, el byte
// menos significativo va el primero
static const u1_t PROGMEM APPEUI[8] = myAPPEUI;
void os_getArtEui (u1_t* buf) {
  memcpy_P(buf, APPEUI, 8);
}

// little-endian
static const u1_t PROGMEM DEVEUI[8] = myDEVEUI;
void os_getDevEui (u1_t* buf) {
  memcpy_P(buf, DEVEUI, 8);
}

// Esta clave debe estar en formato big-endian
static const u1_t PROGMEM APPKEY[16] = myAPPKEY;
void os_getDevKey (u1_t* buf) {
  memcpy_P(buf, APPKEY, 16);
}

int32_t samplesxxx=128; //Cantidad de muestras para obtener una correcta

Adafruit_ZeroI2S i2s; //Creacion de variable de la libreria I2S



// Funcion de muestreo
void sampling(int sample_delay, int N_samples)
{
  digitalWrite(11, LOW); //Configuracion del pin 11 para recibir datos del canal izquierdo
  average = 0;

  float sound;
  int32_t left, right;
  int i, j;

  // Lee samplesxxx muestras, una cada 1 ms:
  int32_t samples[samplesxxx];
  for (j = 0; j <= N_samples; j++)
  {
    for (int i = 0; i < samplesxxx; i++)
    {
        i2s.read(&left, &right);
        delay(1);
        // conversión a 18 bits con signo
        left >>= 14;
        samples[i] = left;
    }
```

```
// Media de las muestras
float meanval = 0;
for (int i = 0; i < samplesxxx; i++)
{
  meanval += samples[i];
}
meanval /= samplesxxx;

// restar media a todas las muestras para obtener una salida 'normalizada'
for (int i = 0; i < samplesxxx; i++)
{
  samples[i] -= meanval;
}

// Encontrar la muestra mayor y menor
float maxsample, minsample;
minsample = 100000;
maxsample = -100000;
for (int i = 0; i < samplesxxx; i++)
{
  minsample = min(minsample, samples[i]);
  maxsample = max(maxsample, samples[i]);
}


//Medida final en decibelios
sound = 10 * log(maxsample - minsample);

Serial.println(sound);

//Se le suma la medida a la variable de media
average += sound;

//Seleccion de la medida mayor y menor
if (j == 1)
{
// Se le asigna inicialmente el valor obtenido a maximos y minimo
  maximum = sound;
  maximum2 = sound;
  minimum = sound;
}
else if (sound < minimum)
{
// Si el sonido es menor que el menor sonido medido,
//entonces este es el nuevo menor
  minimum = sound;
}
```

```
    else if (sound > maximum)
    {
    // Si el sonido es mayor que el mayor sonido medido,
    //entonces este es el nuevo mayor

      maximum = sound;
    }
     else if (sound > maximum2 && sound < maximum)
    {
      // Si el sonido es mayor que el maximo 2 medido y menor que el maximo medido,
      // será el maximo 2
      maximum2 = sound;
    }
    // Retraso para periodo de muestreo
    delay(sample_delay);
  }

  // Valores acumulados en media entre las N_samples medidas
  average=average/j;

  //Impresion de valores finales por monitor serie
  Serial.print("average = "); Serial.println(average);
  Serial.print("minimum = "); Serial.println(minimum);
  Serial.print("maximum = "); Serial.println(maximum2);


// Funcion para el envio de los datos
void report_data(osjob_t* j) {

  // Variable para mandar los datos (entera sin signo de 8 bytes)
  // 21B = 6B * 3 float + 2B ';' + 1B \0
  static uint8_t message[21] = "";

  // Buffer para almacenar los datos
  // y variable para intermediar conversion
  String buff;
  char bufferance[21];

  Serial.print("report_data(): OS Timestamp [");
  Serial.print(os_getTime());
  Serial.print("]: ");
  Serial.println("taking samples from sensor");

  // Llamada a la funcion sampling obtener muestreo
  // y convertir datos del sensor
  sampling(sample_delay, N_samples);

  // Conversion de datos de float a string
  buff = (String) maximum2 + ";" + (String) minimum + ";" + (String) average;
```

```
  // Conversion del string buff al conjunto de char bufferance
  // y se copia a la variable message
  buff.toCharArray(bufferance, 21) ;
  memcpy(message, bufferance, sizeof(bufferance));

  // Comprobacion de que no hay procesos de transmision o recepcion LoRa
  if (LMIC.opmode & OP_TXRXPEND) {
    Serial.println("OP_TXRXPEND: not sending");
    Serial.println("OP_TXRXPEND: There is already another TX/RX job running");
  } else {

    // Preparacion del envio para el siguiente intento
    // de la tecnica de contienda ALOHA
    LMIC_setTxData2(1, message, sizeof(message) - 1, 0);

    // Aqui, cuando finaliza el envio, el evento EV_TXCOMPLETE sera manejado por la funcion onEvent ()

    // Comprobacion del envio por monitor serie,
    // se compara el tiempo con el anterior y
    // se muestra la carga util enviada
    Serial.print("report_data(): OS Timestamp [");
    Serial.print(os_getTime());
    Serial.print("]: ");
    Serial.print("uplink message payload : ");
    Serial.println(buff);

  }
}


// Manejador de eventos del LMIC (LoRaWAN-MAC-in-C)
void onEvent (ev_t ev) {
  Serial.print("onEvent(): OS Timestamp [");
  Serial.print(os_getTime());
  Serial.print("]: ");

// Funcion switch que conmuta segun el evento ev
  switch (ev) {

    // Evento tras realizar transmision
    case EV_TXCOMPLETE:
      Serial.println("EV_TXCOMPLETE (includes waiting for RX windows)");
      if (LMIC.txrxFlags & TXRX_ACK) {
        Serial.println("TXRX_ACK: Received ack");
      }

      // En caso de implementar una respuesta a este envio
      // desde TTN al dispositivo...
      if (LMIC.dataLen) {
        int i = 0;
        // datos recibidos en una de las 2 ranuras de recepcion
        // negociadas tras la transmision
        Serial.print("Data Received: ");
        Serial.write(LMIC.frame + LMIC.dataBeg, LMIC.dataLen);
        Serial.println();
        Serial.println(LMIC.rssi);
```

```
    for ( i = 0 ; i < LMIC.dataLen ; i++ )
      TTN_response[i] = LMIC.frame[LMIC.dataBeg + i];
    TTN_response[i] = 0;
  }
  // Cuando termina el evento, se llama de nuevo al manejador
  os_setCallback (&sendjob, report_data);

  // Para dormir 5 minutos tras el evento de transmision
  // y ahorrar en consumo de bateria
  Serial.println("Going to sleep");
  LowPower.deepSleep(sleepytime);
  Serial.println("Waking up!");

  break;

// Evento de proceso de union con el Join Server
case EV_JOINING:
  Serial.println("EV_JOINING: -> Joining...");
  break;



// Evento de verificacion de union con el JS
case EV_JOINED:
  Serial.println("EV_JOINED");

  // Se deshabilita la validación de verificación de enlace
  // (se habilita automaticamente durante la union, pero TTN no lo admite en este momento)
  LMIC_setLinkCheckMode(0);

  // Empezando a leer el microfono y enviando el dato cuando termina
  Serial.println("Welcome, I´m JOINED");
  os_setCallback (&sendjob, report_data);

  break;

// En caso de datos recibidos
case EV_RXCOMPLETE:
  Serial.println("EV_RXCOMPLETE");
  break;

// En caso de que no se haya recibido ninguna confirmación del servidor de red (NS)
// durante un período prolongado de tiempo.
// Las transmisiones aún son posibles pero su recepción es incierta.
case EV_LINK_DEAD:
  Serial.println("EV_LINK_DEAD");
  break;
```

```
    // Caso para evento desconocido
    default:
      Serial.println("Unknown event");
      break;
  }

}


// Arduino empieza a correr desde aqui...
void setup() {

  // Inicializando puerto serie
  Serial.begin(115200);
  delay(5000);
  Serial.println("I2S Audio Tone Generator");

   // Comprobacion del inicio del transmisor I2S
  if (!i2s.begin(I2S_32_BIT, SAMPLERATE_HZ)) {
    Serial.println("Failed to initialize I2S transmitter!");
    while (1);
  }
  //Se habilita el pin de recepcion de I2S
  i2s.enableRx();


  // El pin 11 en modo control del canal de salida (L/R)
  // del microfono
  pinMode(11,OUTPUT);

  Serial.println("setup(): Serial starting");

  // Inicio de LMIC
  os_init();

  // Se resetea el estado del LMIC,
  // y las sesiones o transmisiones pendientes se descartan.
  LMIC_reset();
  LMIC_setClockError(MAX_CLOCK_ERROR * 1 / 100);


  // Configuracion de los canales utilizados por TTN después de la funcion
  // LMIC_setSession (llamado implicitamente por el evento EV_JOINING y EV_JOINED)
  // parametros del canal: Numero|frecuencia|rango de velocidad de datos(data rate)|banda.
  LMIC_setupChannel(0, 868100000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);
  LMIC_setupChannel(1, 868300000, DR_RANGE_MAP(DR_SF12, DR_SF7B), BAND_CENTI);
  LMIC_setupChannel(2, 868500000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);
  LMIC_setupChannel(3, 867100000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);
  LMIC_setupChannel(4, 867300000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);
  LMIC_setupChannel(5, 867500000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);
  LMIC_setupChannel(6, 867700000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);
  LMIC_setupChannel(7, 867900000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);
  LMIC_setupChannel(8, 868800000, DR_RANGE_MAP(DR_FSK,  DR_FSK),  BAND_MILLI);
```

```
  // Deshabilitar verificacion de conexion de red
  LMIC_setLinkCheckMode(0);

  // TTN usa SF9 para la segunda ventana de recepcion
  LMIC.dn2Dr = DR_SF9;

  // Data rate y potencia de transmision en decibelios, con cualquiera
  // de estos 3 consigue el envio. Se elige el factor de
  // propagacion 9 (BW 125KHz), que utiliza un ancho de banda inferior al 7(BW 250KHz),
  // para hacer que llegue más lejos la señal al ser el gateway menos sensible al ruido
  // (aunque tarde mas en enviar el dato).
  //LMIC_setDrTxpow(DR_SF11,14);
  //LMIC_setDrTxpow(DR_SF7,14);
  LMIC_setDrTxpow(DR_SF9,14);

  // Comienzo de intento de union a la red
  LMIC_startJoining();

}

void loop() {
  // Llama al proceso runloop del LMIC dentro de este
  // bucle infinito para que sucedan los eventos continuamante
  os_runloop_once();
}
```

Figura 51. Código del firmware

4.1.2. Hardware

4.1.2.1. Microcontrolador Adafruit Feather m0 RFM9x LoRa Radio

4.1.2.1.1 Conexionado de pines del Adafruit Feather m0 RFM9x LoRa Radio

Para que funcione correctamente, primero hace falta añadir un cable entre el pin digital 6 y el pin digital asociado al RFM io1 [7], ya que el io1 está conectado internamente al pin 3 (A0), que es el utilizado como pin de radio GPIO0 o de solicitud de interrupción.

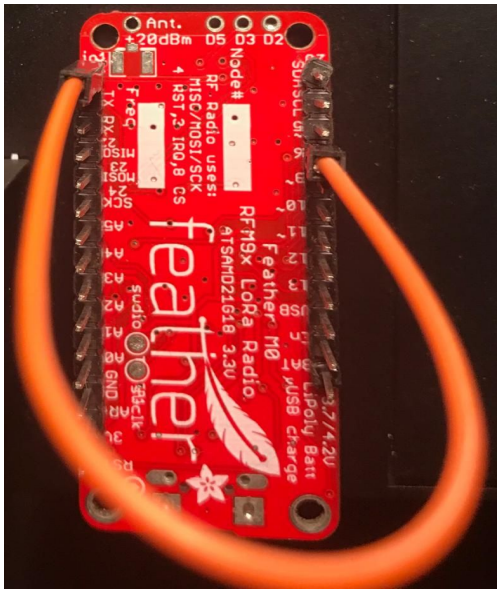

Figura 52. Conexión de pines 3-6

A continuación se conectarán los pines del micrófono al Adafruit.

Los pines GND y VDD estarán conectados, respectivamente, a los pines GND y 3V, que son los pines de tierra y de tensión de alimentación a 3,3 voltios.

Para la conexión de los pines asociados a la comunicación I2S emplearemos las instrucciones de la librería que se utilizará para ello, Adafruit_ZeroI2S.h. Que indica que el pin WS de selección de palabra se conecta al pin digital 0 (I2SF0/RX, cable blanco de la imagen inferior), el pin SCK de de reloj de datos en serie se conecta al pin digital 1 (I2SCK/TX, cable rojo), y el pin SD de salida de datos en serie se conecta al pin digital 9 (I2SD0, cable azul). La conexión del pin de selección de canal L/R irá al pin 11 (cable verde), como se explicará más adelante en la implementación del código.

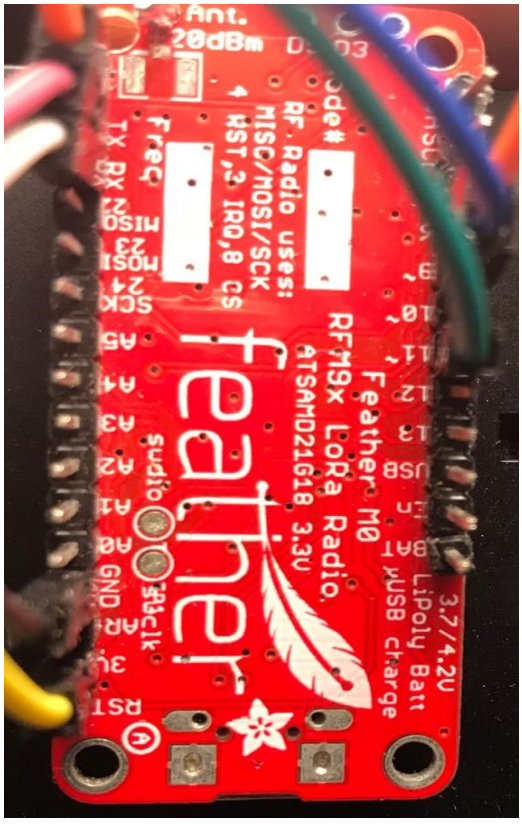

Figura 53. Conexión de pines con el micrófono

4.1.2.1.2 Flasheo del microcontrolador

Para instalar el firmware al microcontrolador, se conectará este por puerto USB y el Arduino IDE lo reconocerá tras el paso anterior en un puerto COM específico. Se podrá comprobar el estado del dispositivo, las propiedades del controlador del dispositivo, los eventos ocurridos en la conexión y las características del puerto COM en la pestaña de puertos (COM y LPT) del administrador de dispositivos de Windows.

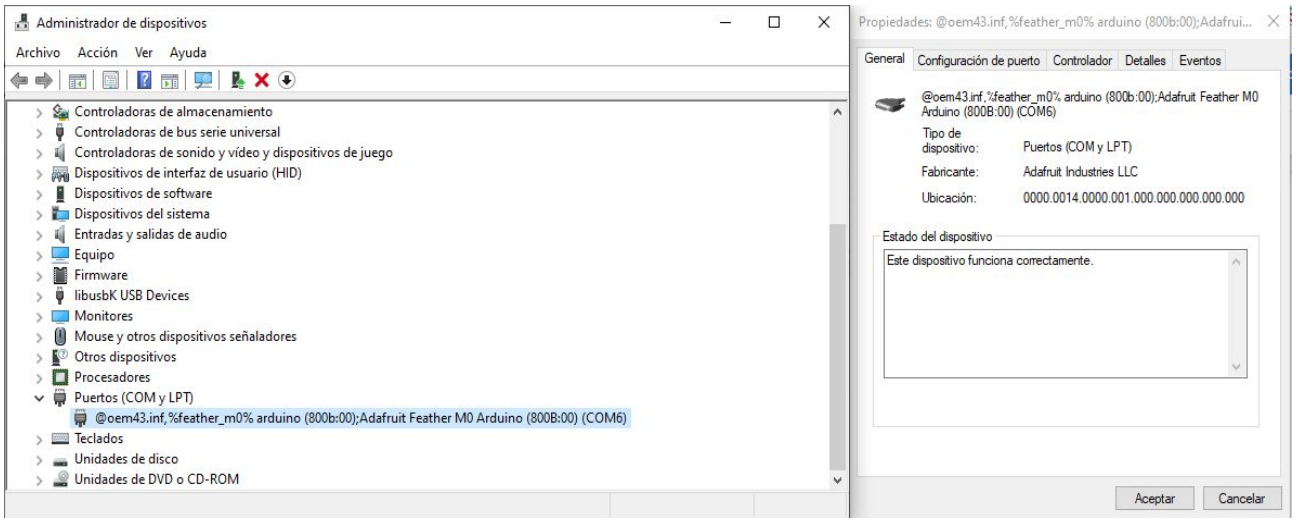

Figura 54. Comprobación del dispositivo conectado al puerto COM

Para *flashear* la placa con el código del proyecto de Arduino, esta tendrá que estar en modo *bootloader.* Esto significa que estará en un modo específico en el que la placa no correrá ningún código cargado anteriormente y estará esperando la subida desde el Arduino IDE de un nuevo firmware. Para que el Adafruit entre en este modo hay que presionar dos veces seguidas el botón de reset y este cambiará automáticamente al puerto de bootloader, que es otro puerto COM distinto al de *run* mode.

Ya que está en *bootloader* mode, se pulsa el botón "subir" del Arduino IDE y el firmware será compilado e instalado en la placa.

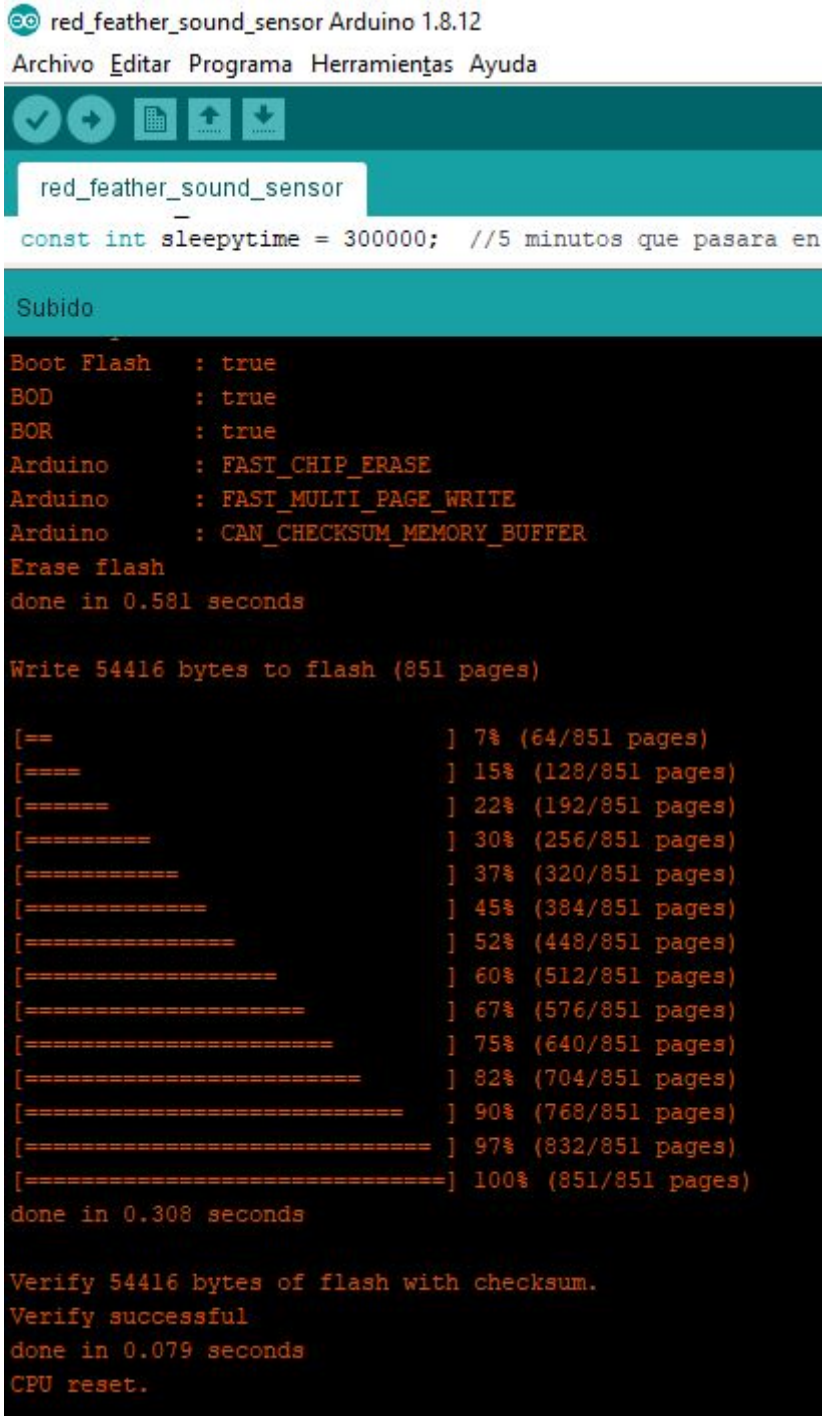

Figura 55. Compilación e instalación del firmware

Tras la subida, el Adafruit cambiará de nuevo al puerto COM asociado al *run mode* de forma automática y se podrá comprobar desde el monitor serie el funcionamiento del programa.

4.1.2.2. Elementos de conexionado

Para hacer las pruebas necesarias, se usará el microcontrolador interconectado sobre una placa de pruebas o *protoboard,* o se interconectarán sus pines directamente con cables.

4.1.2.2.1 Protoboard

La *protoboard* utilizada es del modelo de 630 puntos, que se muestra en la siguiente imagen.

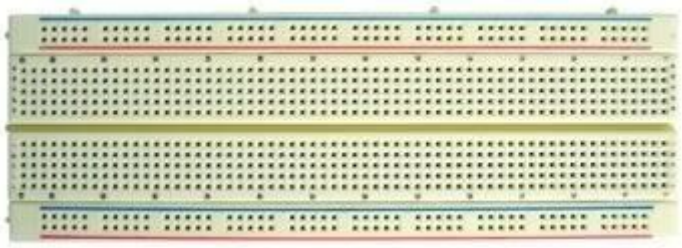

Figura 56. Protoboard

4.1.2.2.2 Cables

Los cables son especiales para interconectar pines de placas de pruebas entre sí, y se llaman cables puente. Para las pruebas con la placa se usarán cables puente hembra-hembra cuando no se utilice la *protoboard* y cables puente hembra- hembra y macho-hembra cuando sí.

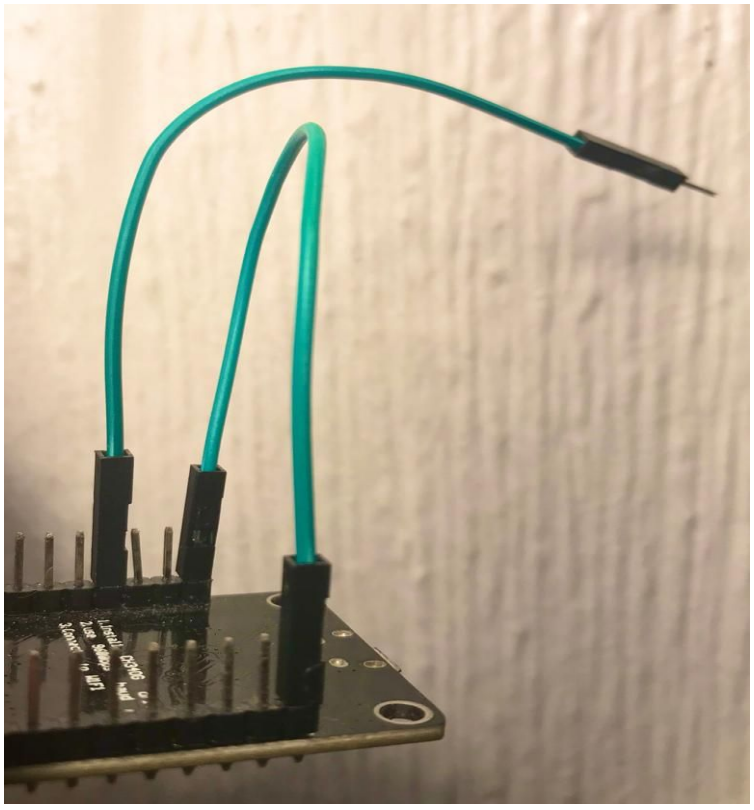

Figura 57. Cables puente

4.1.2.3 Antena

Para elegir la antena, se tendrá en cuenta que tendrá que recibir la señal LoRa a una frecuencia de 868/915 MHz, por lo que se usará una que reciba en este rango exacto y que no se exceda en tamaño y tiempo. Y la antena Heltec de 868-915 MHz se ajusta a los requisitos.

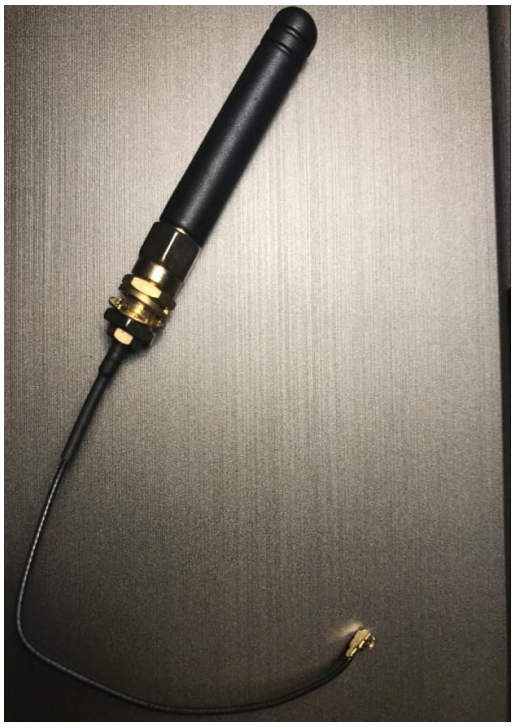

Figura 58. Antena Heltec 868-915 MHz

Como alternativa, el fabricante da la opción de poner una antena de cuarto de onda. También es efectiva y se ahorra en precio al desarrollar el producto final. Esta antena consiste únicamente en un cable cuya longitud será la longitud de onda entre 4. Como se transmite a 868 MHz, se puede deducir a partir de la fórmula $c = \lambda \cdot f$ que la longitud de onda es 300000000(m/s) / 868000000(1/s o Hz) = 0,3456 metros. Por lo tanto, la antena tendrá una longitud de 34,56cm/4 = 8,64cm.

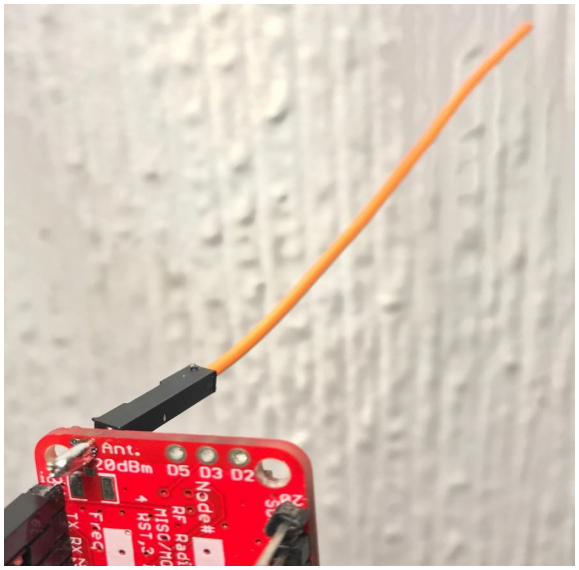

Figura 59. Antena de cuarto de onda soldada a la placa

4.1.2.4. Fuente de alimentación

La fuente de alimentación podrá ser tanto a pilas en serie como a batería, así que se estudiarán los dos casos para ver cuál de las dos soluciones aportan una mayor autonomía a la placa, sabiendo que esta transmitirá durante 70ms a una intensidad máxima de 145 mA a +20 DBm, y el resto del tiempo puede estar escuchando a 13 mA o en *sleep mode* a 11 mA.

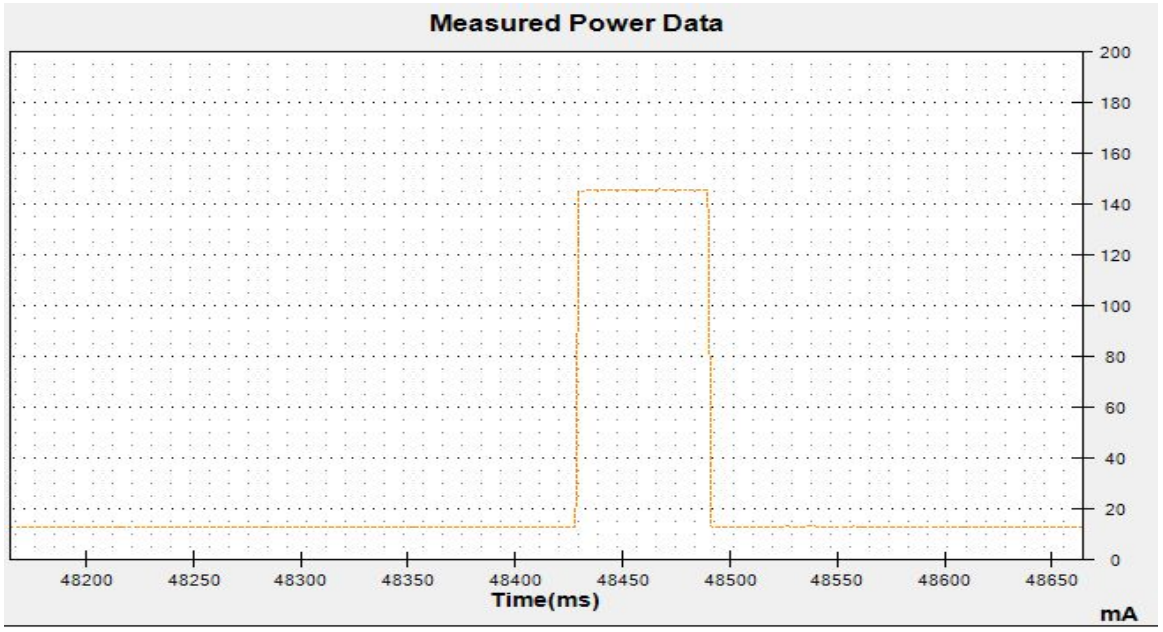

Figura 60. Consumo durante la transmisión de datos del Adafruit

4.1.2.4.1 Batería Lipo

Figura 61. Batería LiPo

Si se utiliza una batería LiPo de 3,7 V y 2500 mAh (más del doble de la batería de 1100 mAh con la que se realizarán las pruebas), se cubrirá el consumo a 3,3 V del microcontrolador. Y haciendo cálculos de consumo a partir de la gráfica anterior y de la fórmula Wbateria / Wconsumida tendremos el tiempo de vida.

Wbatería será la potencia de la batería, luego 3,7V * 2,5Ah = 9,25Wh.

Wconsumida será la potencia consumida, que resulta de multiplicar los 3,7V que da la batería por el consumo medio Imicro. La gráfica de la intensidad del microcontrolador indica que los envíos duran 70ms y en ese tiempo hay 145mA de intensidad. Y como el resto del tiempo hasta que pasen 5 minutos estará solo escuchando o dormido, pasarán entre 11 y 13 mA. Por lo que los 145mA se pueden despreciar por el corto periodo de tiempo que el Adafruit pasa enviando comparado con el que se pasa sin enviar. Entonces,

quedaría un consumo Wconsumida= 0,013A * 3,7V = 0,048W.

Por último, el resultado del tiempo de vida de la batería LiPo es de 9,25Wh / 0,048W = 192h u 8 días.

Este resultado sería el calculado con las condiciones más estrictas, y como se verá en el apartado de la ejecución del proyecto, se podrán emplear librerías para dejar la placa en modo sueño y poder modificar la potencia de transmisión.

4.1.2.4.2 Pila

Figura 62. Pilas en serie

Se elegirá un conjunto de 3 pilas alcalinas de tipo AAA de 1,5 V en serie para alimentar la placa a 3,3 V. El tiempo de vida de la pila también dependerá de la capacidad, que para las pruebas serán de 1000 mAh. Para calcular el tiempo de duración se tendrán en cuenta los mismos parámetros que con la batería, teniendo en cuenta que el conjunto de pilas da 4,5 V a 1000mAh. El cálculo resulta 7 días y 21 horas.

A partir de aquí, se puede razonar a simple vista que una batería LiPo sale más rentable. El pack de 4 pilas cuesta alrededor de 1,5 €, que en un mes serían 4 recargas de 3 pilas, es decir, 3 packs que saldrían a 4,5€. Y como una batería LiPo de 2500 mAh 3,7V cuesta alrededor de 9 €, cuando tenga 2 meses de uso quedará amortizada.

4.1.2.4.3 Comparación de soluciones de alimentación

Una de las ventajas del Adafruit Feather m0 RFM9x LoRa Radio es que cuando una batería recargable está enchufada a la vez que el USB, la entrada USB alimenta la placa y a la vez recarga la batería. Y esto podría servir como sistema de alimentación ininterrumpida (SAI) en el caso de que el dispositivo esté siempre a través de alimentación USB. O también podría dar una mayor comodidad y beneficio en el caso de que siempre funcione con batería y el cliente tenga que ir recargándolos, ya que las pilas serían más costosas a medio-largo plazo como se ha demostrado. Y al final hay que hacer un balance entre la duración, precio y comodidad de recarga, por lo que la alimentación mediante batería LiPo con un tiempo de vida de 8 días y función de recarga mediante USB será la solución empleada.

4.1.2.5. Sensor Acústico

El sensor de ruido será de salida digital, con micrófono omnidireccional y comunicación con microcontrolador mediante I2S. Es necesario que sea onmidireccional ya que los unidireccionales captan el sonido en un rango menor de distancia. Y como el omnidireccional tiene un mayor rango, también tiene más rango para captar ruido ambiental, que se solucionará con el calibrado.

En concreto, será el micrófono INMP441 de la marca TECNOIOT.

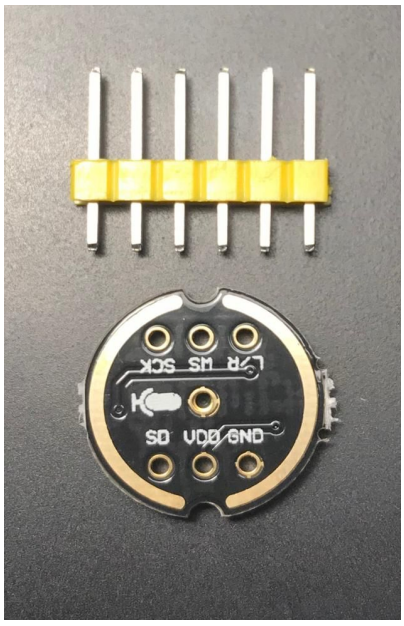

Figura 63. Micrófono INMP441

Se ha elegido esta opción ya que la diferencia de precio entre un micrófono digital y otro analógico es considerable, y el único trabajo necesario para adaptar este micro de bajo coste será calibrarlo a partir de un sonómetro de alta precisión.

Los pines tienen el siguiente significado:

SCK es el reloj de datos en serie de I2S.
WS es la selección de palabra de datos en serie de I2S.
L/R es la selección de canal de emisión izquierdo o derecho.
SD es la salida de datos en serie de I2S.
VDD es la fuente de entrada.
GND es la tierra.

4.1.2.5.1 Calibrado del sensor acústico

El siguiente paso será calibrar el micrófono omnidireccional INMP441. Para ello se utilizará la recta de corrección, que es una gráfica que muestra la diferencia en la respuesta en frecuencia entre la medición del micrófono a calibrar y el sonómetro. Esto permitirá calibrar la respuesta en frecuencia del micrófono, aunque la respuesta en fase no se podrá calibrar con este método.

Lo primero será encontrar una fuente sonora válida, es decir, con una buena respuesta en frecuencia y ganancia para el tono con el que se realizarán las mediciones.

Lo siguiente será medir el sonido a la vez con el sonómetro y el micrófono a calibrar, a la misma distancia y reproduciendo en la fuente un tono de 1 KHz a un volumen que supere, al menos, en 20 decibelios al ruido ambiente, e ir subiendo el volumen hasta el máximo nivel que se quiera calibrar. En este caso, el máximo será el estudiado más adelante tras medir el ruido urbano. Por último, solo bastará con introducir los datos numéricos de la recta de corrección del excel al software que calibre mediante esta al micrófono.

4.1.2.6 Sonómetro

El sonómetro se utilizará para realizar pruebas en el medio urbano. Servirá tanto para entender el rango de decibelios en diferentes entornos (antes de calibrar el micrófono) como para validar las pruebas finales.

Se usará el sonómetro digital de marca BENETECH modelo GM1356, que mide en un rango de 30 a 130 decibelios con un error de ±1,5 decibelios.

Figura 64. Sonómetro BENETECH GM1356

4.2. LoRa Gateway

4.2.1 Pasarela TheThingsNetwork

El *gateway* de TTN no será necesario adquirirlo para las pruebas, ya que en la ciudad de Sevilla se disponen de cinco de ellos obtenidos por personas anónimas que los dejan a disposición para su uso público de la red LoRaWAN.

Figura 65. Gateways LoRaWAN en la ciudad de Sevilla

Se estudiará, por ejemplo, el módulo de gateway RAK831, que es uno de los que más cobertura ofrece en el centro de Sevilla.

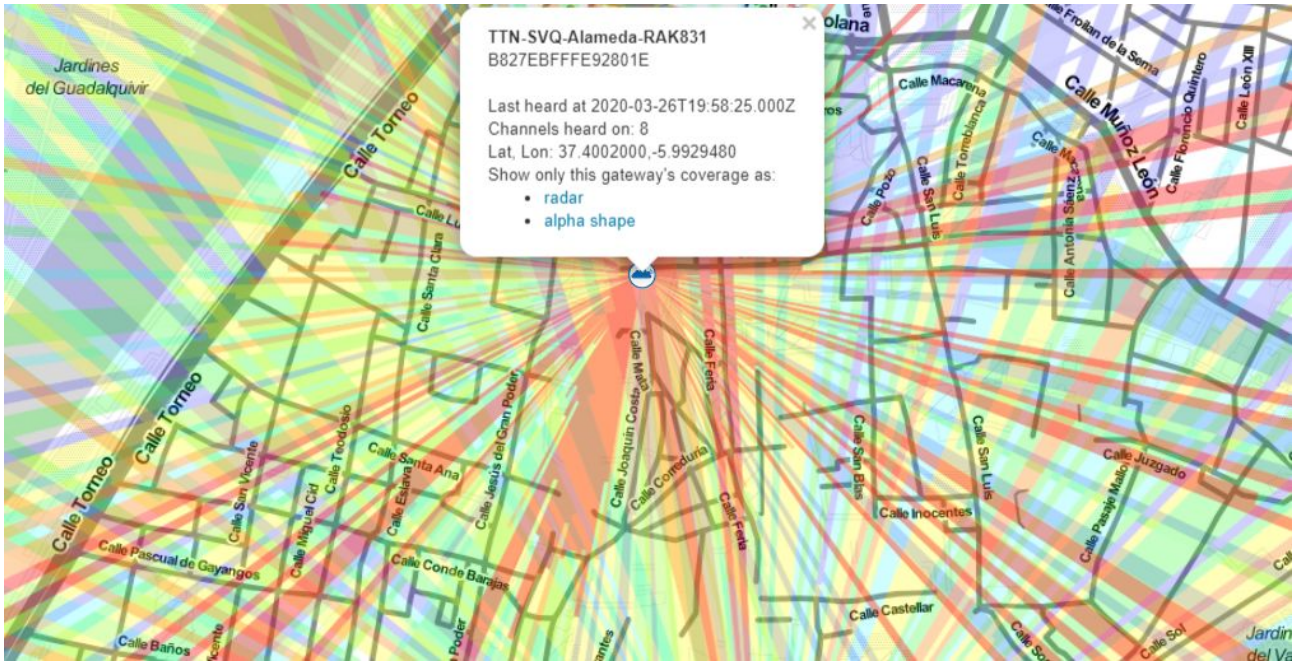

Figura 66. Localización del gateway RAK831 estudiado

Este gateway ofrece una cobertura de hasta 2 kilómetros en un medio urbano denso y hasta 15 km sin obstáculos, por ejemplo, poniendo la antena en un sitio elevado de la ciudad para que haya línea de visión directa con los dispositivos. Y como se puede demostrar al observar los test de TTN Mapper sobre este gateway, la distancia de cobertura puede llegar a ser mucho mayor, llegando hasta los 77 kilómetros (correspondiente a la línea azul más larga de la figura inferior).

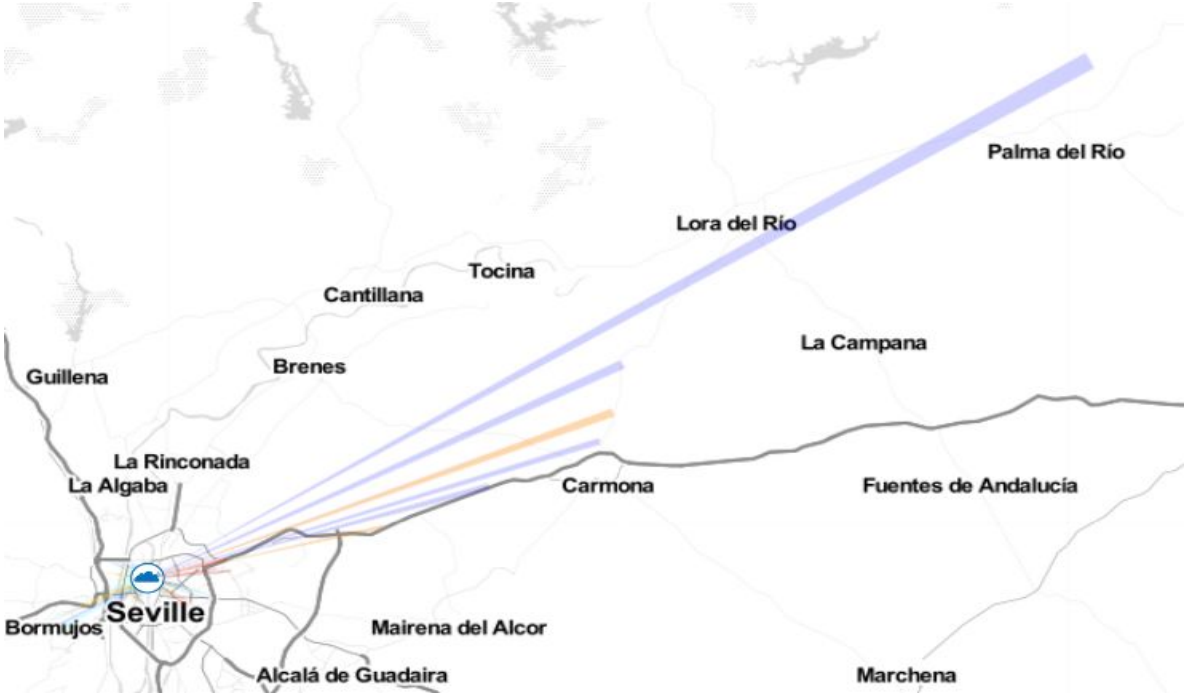

Figura 67. Mapa de cobertura del gateway RAK831 estudiado

4.3. Aplicación

Este apartado contiene todo lo relacionado con el desarrollo de la aplicación en TTN (*Application*), incluidas las extensiones TTN Mapper y DataStorage para los test de cobertura y el almacenamiento de los datos, respectivamente.

4.3.1. Configuración TheThingsNetwork

En la plataforma web de TTN se pueden configurar los parámetros asociados al servidor de aplicación y sus correspondientes dispositivos. Además, se podrán emplear integraciones dentro de la aplicación para diversos usos de los datos recibidos (almacenamiento en base de datos, envío de los datos sobre HTTP a otro destino diferente a TTN, etc.).

4.3.1.1 Configuración de la aplicación

Para empezar, lo primero será crear una cuenta en TTN para poder utilizar sus herramientas. Y después, añadir la aplicación en la región correspondiente (ttn-handler-eu) que contendrá el *device* utilizado en el proyecto. Al crearlo también hace falta introducir el application ID, que servirá para diferenciarlo de otras aplicaciones en el mismo servidor.

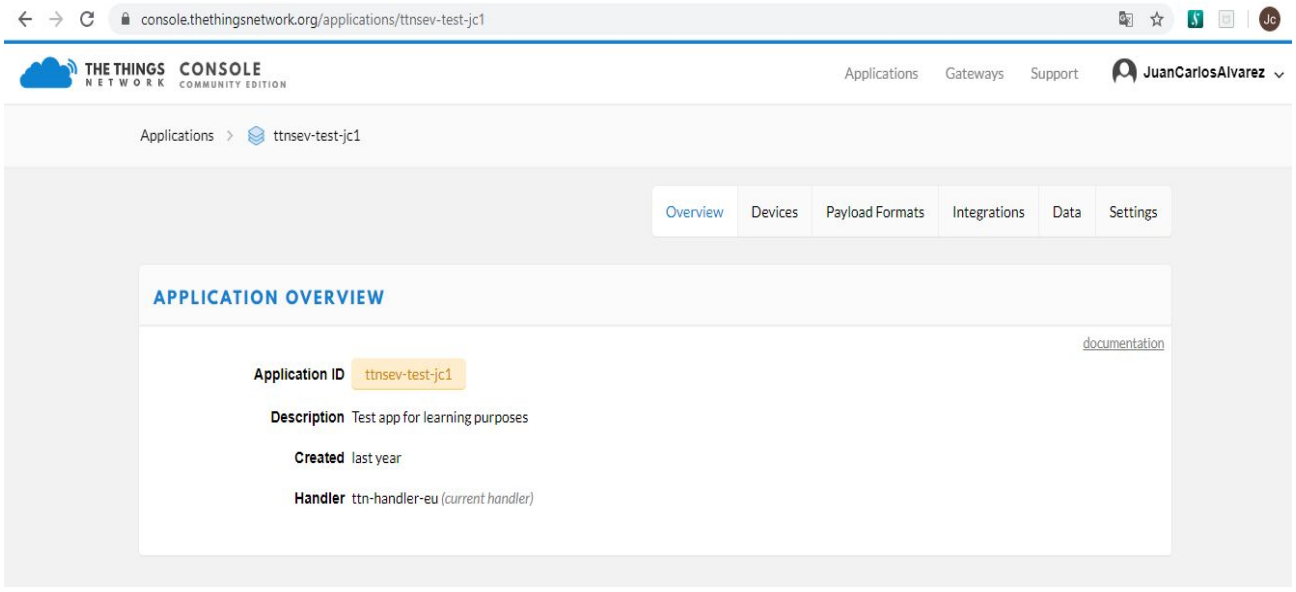

Figura 68. Aplicación creada en TTN

A partir de este punto, ya se pueden añadir devices, configurar el formato de payloads recibidos, administrar integraciones y observar el tráfico de datos que atraviesa la aplicación.

4.3.1.2 Configuración del device

A continuación, se añadirá el device desde el apartado correspondiente de la aplicación y se le asignará un device ID.

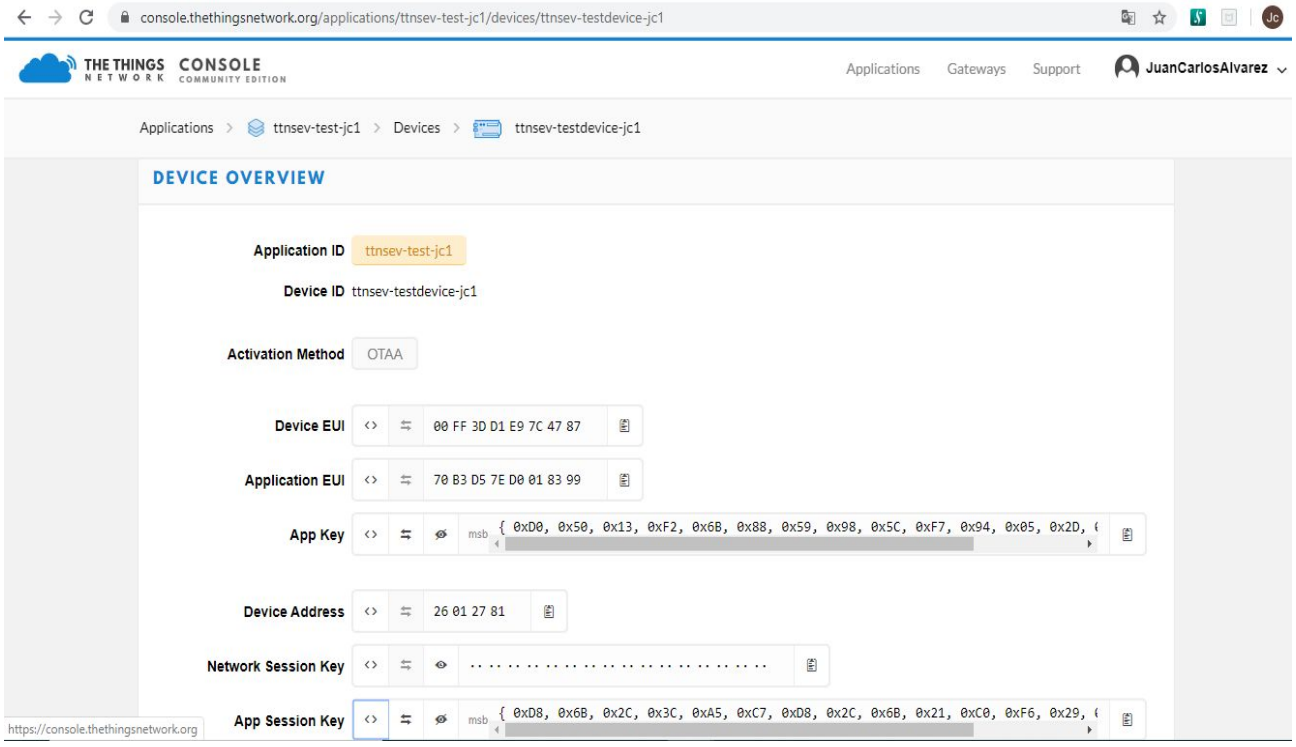

Figura 69. Página de configuración del device en TTN

En los ajustes, se puede elegir el método de activación del *device,* y como ya se ha comentado anteriormente, dentro del método ABP se puede generar un AppSKey y un NwkSKey introduciéndolo directamente o dejando que la propia aplicación la genere automáticamente. Además, el device Address lo asignará el NS.

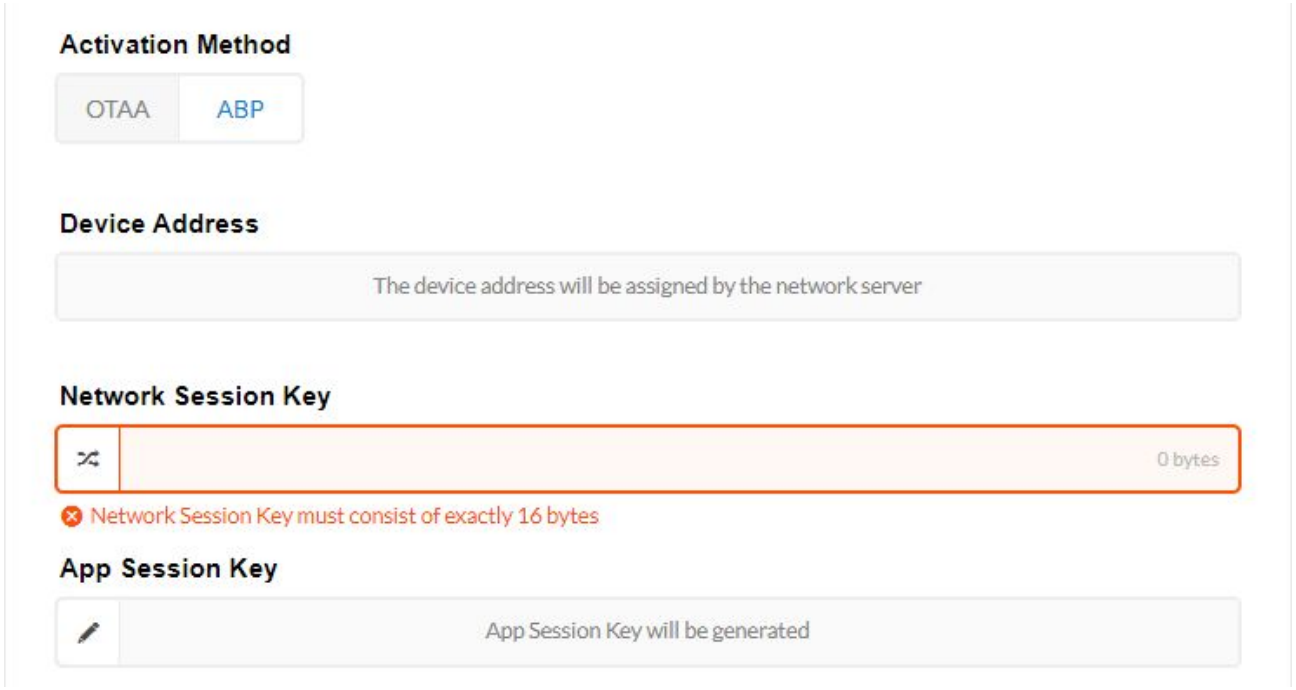

Figura 70. Opciones de configuración con método de activación ABP

Para el método OTAA, que será el utilizado por las razones de seguridad descritas anteriormente, sólo hará falta generar la AppKey, ya sea automáticamente o manualmente. El resto de claves se generarán en procesos de negociación en la red LoRaWAN como los descritos en el apartado anterior. Estas claves se pueden

consultar en el menú principal del dispositivo (primera imagen de este subapartado).

Figura 71. Opciones de configuración con método de activación OTAA

4.3.1.3 Configuración Data Storage

Los datos serán almacenados en la base de datos proporcionada por la integración *Data Storage*. Esta integración se añade desde el apartado de integraciones de la aplicación en la web de TTN. Cuando esté añadida, se accederá a la web de la base de datos pulsando *go to platform* dentro de la integración. Y una vez dentro, se introduce el *access key* de la aplicación tras pulsar *authorize,* autorizando así a la base de datos a acceder a los datos recibidos en la aplicación.

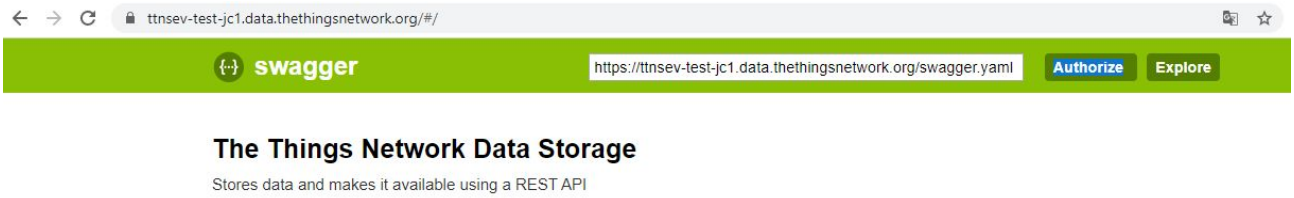

Figura 72. Página de la integración Data Storage de TTN

El *access key* se encuentra al final del *overview* de la aplicación de TTN.

4.3.2. TTN Mapper

Para la comprobación de la cobertura del dispositivo es útil la integración TTN Mapper, que usa los datos de GPS de un dispositivo para comprobar el RSSI y el SNR de la señal LoRa recibida. Además, tanto en su página web (https://ttnmapper.org/) como en su aplicación se puede ver el mapa de cobertura y los Gateways con su información básica.

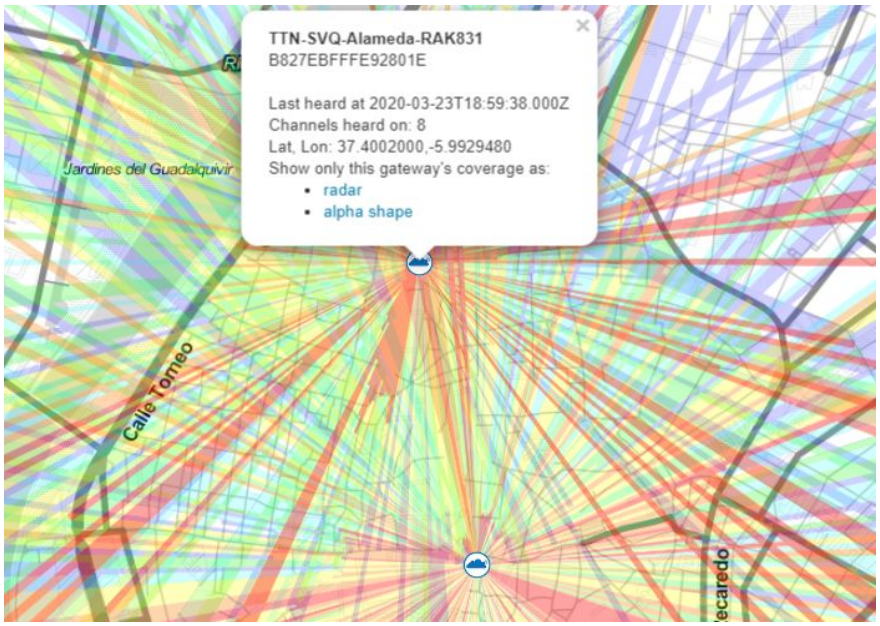

Figura 73. Mapa de cobertura de TTN Mapper

4.3.2.1 Instalación y uso

Para usar TTN Mapper en sitios donde no haya ordenador, línea de internet o el dispositivo LoRaWAN no tenga GPS, se usará la aplicación móvil de TTN Mapper para Android o IOS, y gracias a la cobertura GPS del móvil se podrán transmitir los valores obtenidos por el device a la plataforma de TTN Mapper. Este método será útil a la hora de probar la cobertura en diferentes sitios públicos de la ciudad.

La aplicación está disponible en las tiendas de Android y IOS, Google Play y App Store, respectivamente. Para conectar el device con esta aplicación, solo hará falta introducir en la sección *configure* ciertos parámetros obtenidos de la aplicación en la web de TTN. Esto es así ya que desde el *backend* o respaldo de la plataforma de TTN se mandan los datos de cobertura (obtenidos de la conexión con el device) mediante MQTT a la aplicación de TTN Mapper. Para verificar la conexión con TTN se realiza el *Test Config*.

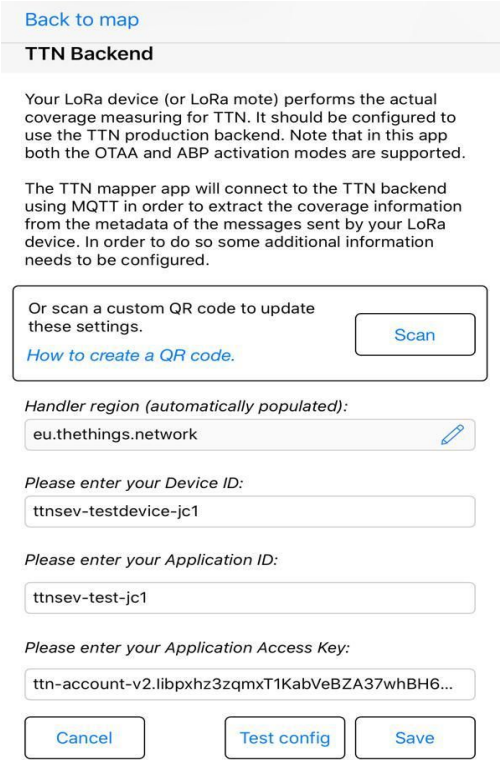

Figura 74. Pantalla de configuración de aplicación móvil de TTN Mapper

4.3.2.2 Pruebas de conexión con TTN Mapper

La integración de TTN Mapper permite tener una base de datos pública en la que diferentes usuarios han hecho pruebas de señal con sus respectivos *devices* en diferentes puntos de la ciudad. Estas pruebas indican los parámetros de señal en cada envío de datos a TTN necesarios para evaluar la cobertura en un sitio concreto. Esto servirá para saber los lugares donde conviene situar el *device* y para sacar conclusiones sobre la cobertura en sitios concretos de la ciudad.

Como el *device* no tiene función de geolocalización GPS, se deben realizar los test de cobertura acompañados del dispositivo móvil con la aplicación de TTN Mapper como se explica en el correspondiente apartado de esta memoria.

Las pruebas de cobertura se realizan desde un sitio cercano al *gateway* LoRa y se repetirán mientras se amplía la distancia con este. En este caso se realizan desde el *gateway* situado en las oficinas de Pod Group, junto a la Plaza de la Encarnación de Sevilla, y se tomará un rumbo hacia el otro *gateway* del centro de Sevilla, situado en la Alameda de Hércules.

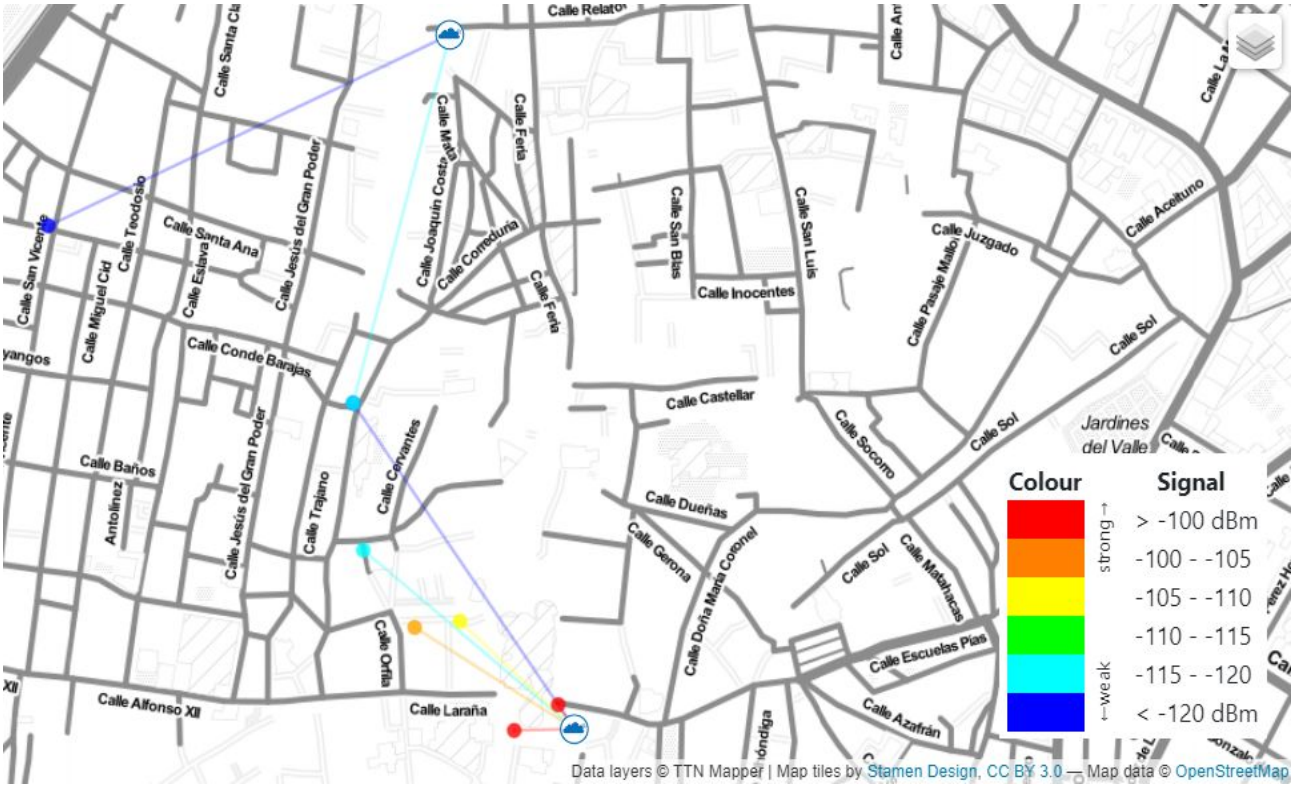

Figura 75. Mapa en TTN Mapper de pruebas de cobertura

Se realiza el test de cobertura en estos 7 lugares distinguiendo el color de la señal conforme a la leyenda de la derecha de la imagen, que indica la potencia en dBm de la señal recibida en el *device* (RSSI). Se puede observar cómo la señal recibida es menor cuanta más distancia hay entre este y el gateway, llegando a enviar los datos de medición a TTN siempre y cuando el RSSI es superior a -120 dBm.

Cabe destacar que la arquitectura en esta parte de Sevilla se conforma principalmente a base de edificios antiguos con paredes de piedra de gran anchura, lo que justifica la pérdida de señal a distancias menores que las expuestas en la especificación de LoRa debido a la gran capacidad de aislamiento y absorción de estas paredes. Se puede observar cómo a unos 200 metros del primer test se pierden 22 dBm de RSSI y a unos 300 metros se pierden 35 dBm hasta llegar a los -118 dBm.

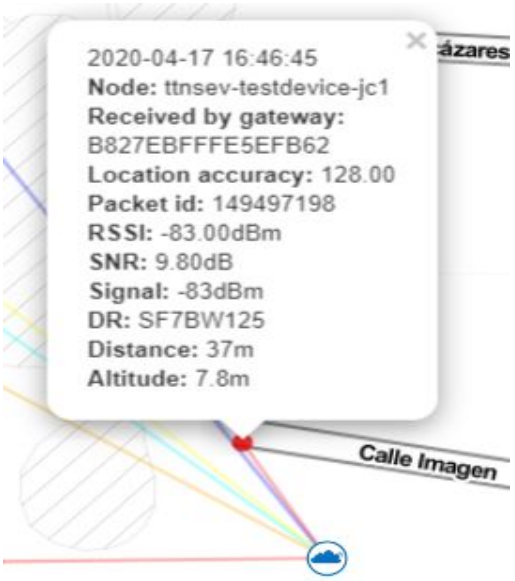

Figura 76. Primer estudio de test de cobertura

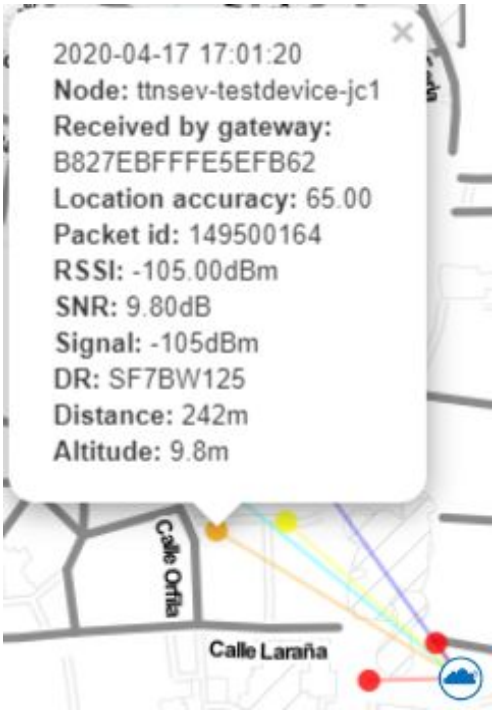

Figura 77. Segundo estudio de test de cobertura

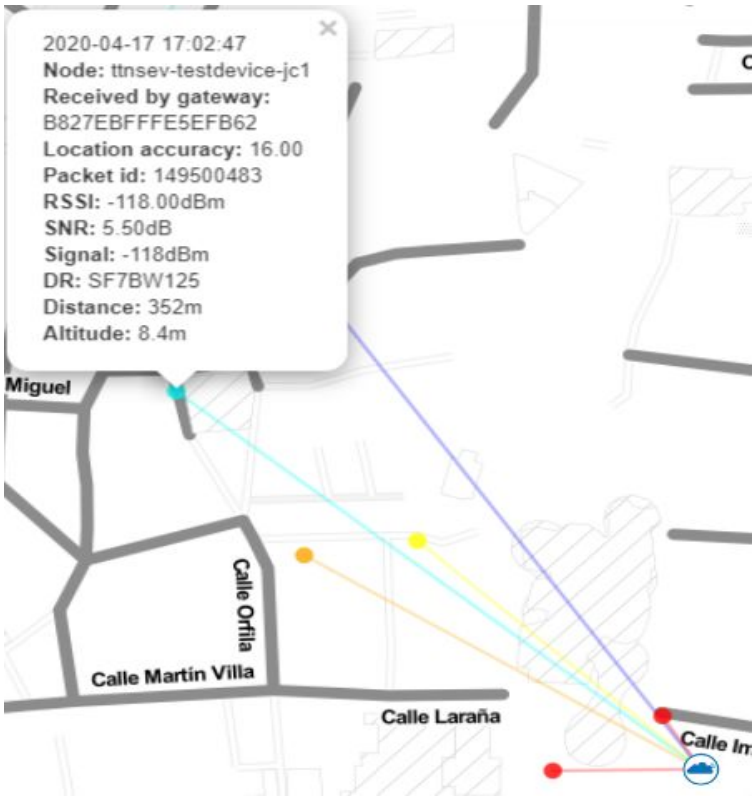

Figura 78. Tercer estudio de test de cobertura

En la penúltima prueba, ya llega más potencia de señal desde el *gateway* de la Alameda de Hércules y el device deja de mandar los datos al de la oficina (a medio kilómetro), ya que el nivel de señal de este está por debajo de los -120 dBm y la relación señal ruido es demasiado baja.

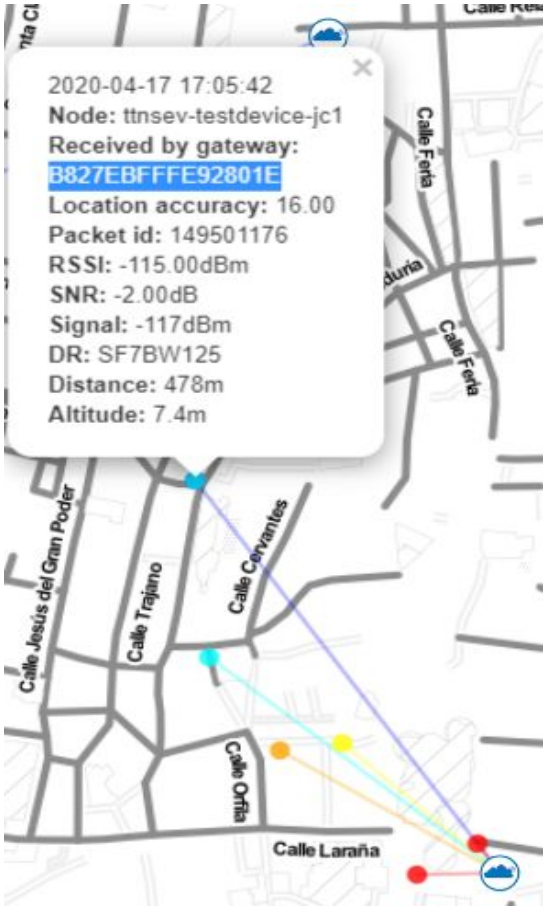

Figura 79. Cuarto estudio de test de cobertura

Se sabe que se usa el *gateway* de la Alameda de Hércules porque en el campo del test *Received by gateway* está su identificador.

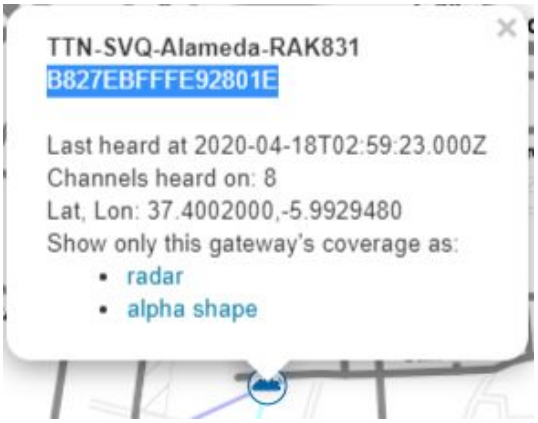

Figura 80. Identificador del gateway RAK831 situado en la Alameda de Hércules

4.3.2.3 Conclusiones

Esta herramienta puede ser útil a la hora de diseñar una infraestructura de gateways LoRa en medio urbano. Y, en la ciudad de Sevilla, que hay una gran variedad arquitectónica con predominio de casas con gruesos muros de piedra en algunas zonas céntricas, la señal llega de forma más débil que en otros sitios. Debido a esto, es de gran utilidad esta aplicación para tener una base de datos pública de la cobertura y realizar una gran cantidad de pruebas a nivel comunitario en estos sitios, donde es difícil determinar en una zona el nivel de señal recibido y la relación señal-ruido provocado tanto por estos problemas arquitectónicos como por problemas de interferencia de señal.

5 PRUEBAS Y VALIDACIÓN

5.1. Plan de pruebas

Tras las configuraciones y pruebas realizadas en el apartado anterior, hay que verificar que los diferentes puntos del sistema de comunicación del proyecto funcionen correctamente, es decir, desde el Arduino IDE utilizado para flashear el microcontrolador hasta el almacenamiento de los datos medidos en DataStorage. Esto es la conexión entre Arduino y el microcontrolador, la conexión del microcontrolador con el *gateway* de la red LoRa, la conexión del *gateway* en la red interna de TTN, la conexión entre la aplicación de TTN y la extensión de la base de datos.

Por último, para terminar de validar la funcionalidad del dispositivo, se verificarán los datos de nivel de ruido recogidos en el desarrollo del proyecto y se verificarán comprobando si son acordes al entorno urbano de cada ubicación.

5.2. Verificación de conexión

5.2.1 Verificación del puerto serie

Para verificar la conexión con el puerto serie, solo hace falta mirar la parte inferior derecha, que indicará el nombre del dispositivo y el puerto COM al que está conectado correctamente.

Si se intenta subir el proyecto en modo *run,* este no se subirá y empezará a cambiar de puerto de forma aleatoria y dará diversos fallos. Así que es de principal importancia subir el proyecto en modo *bootloader* y no empezar a monitorizar por puerto serie hasta que no se observe abajo a la derecha que ha cambiado al puerto COM de modo *run.*

Por otra parte, cabe la posibilidad de que el puerto serie no se comunique con el Adafruit debido a que el controlador no está instalado correctamente en Windows o que no está actualizado. Esto se soluciona instalándo el controlador desde https://learn.adafruit.com/adafruit-arduino-ide-setup/windows-driver-installation si no lo ha detectado tras la instalación del paquete de tarjetas, y en otro caso se soluciona actualizando el controlador, desde la sección del controlador del administrador de dispositivos de Windows. Donde aparecerá en la pestaña de puertos COM al conectar la placa mediante USB.

5.2.2 Verificación de la red LoRa

Para verificar la conexión de la red LoRa entre el dispositivo y el Gateway se pueden observar los eventos de la librería LMIC ocurridos durante el funcionamiento del programa del Adafruit, que se muestran a través del monitor serie como se explica en el código del firmware.

El *device* procederá al principio con el proceso de activación, que ocurrirá en el evento EV_JOINING mientras encuentra el *Join Server* y EV_JOINED cuando ya se ha unido a él y está listo para el envío de datos al *Application Server* (TTN). Tras unirse, el sensor empezará a recoger las muestras de ruido ambiental.

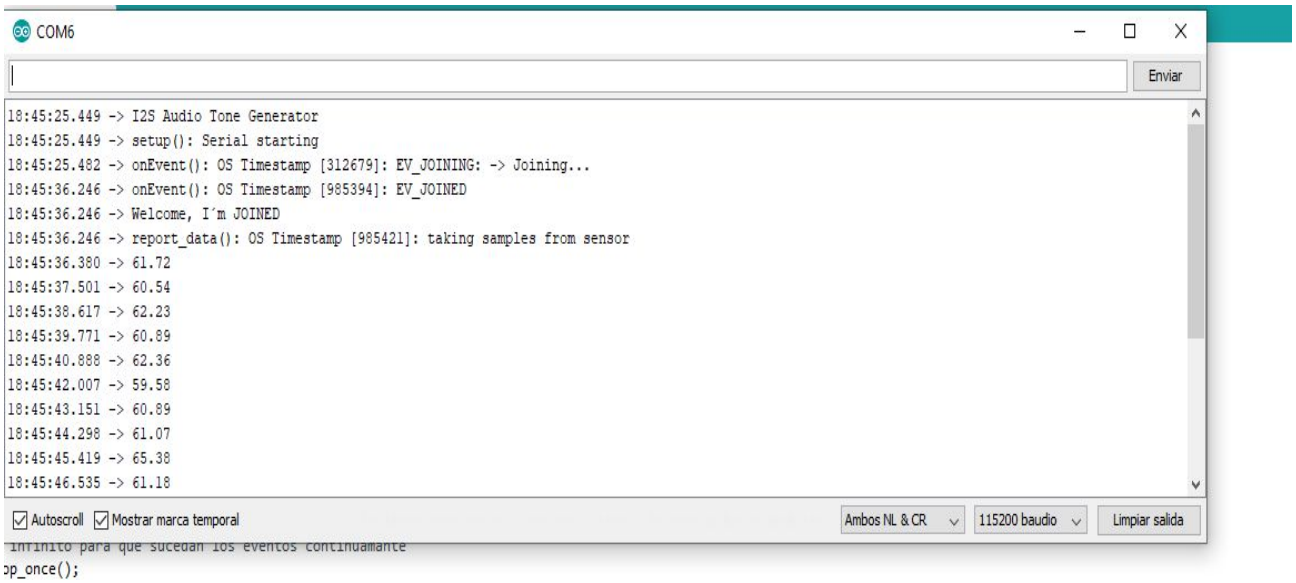

Figura 81. Activación del device y comienzo de medición de datos

Tras recoger las muestras, se procede a enviar los datos de media, máximo y mínimo sonido en el *payload* del mensaje del canal de subida, activándose el evento EV_TXCOMPLETE una vez se haya enviado correctamente. Y por último, se echa a dormir la placa hasta el siguiente envío.

```
18:46:45.246 -> average = 65.02
18:46:45.246 -> minimum = 59.27
18:46:45.246 -> maximum = 83.61
18:46:45.246 -> report_data(): OS Timestamp [5299320]: uplink message payload : 83.61;59.27;65.02
18:46:47.359 -> onEvent(): OS Timestamp [5431613]: EV_TXCOMPLETE (includes waiting for RX windows)
18:46:47.359 -> Going to sleep
```

Figura 82. Envío de datos y paso a modo sleep

5.2.3 Verificación en TheThingsNetwork

Cuando se detecta la primera comunicación del Adafruit con TTN se puede comprobar en el *status* del *device* cuántos segundos han pasado desde que ocurrió, y a partir de esto ya se puede empezar a comprobar la pestaña *Data* para ver los mensajes que han sido recibidos en la plataforma.

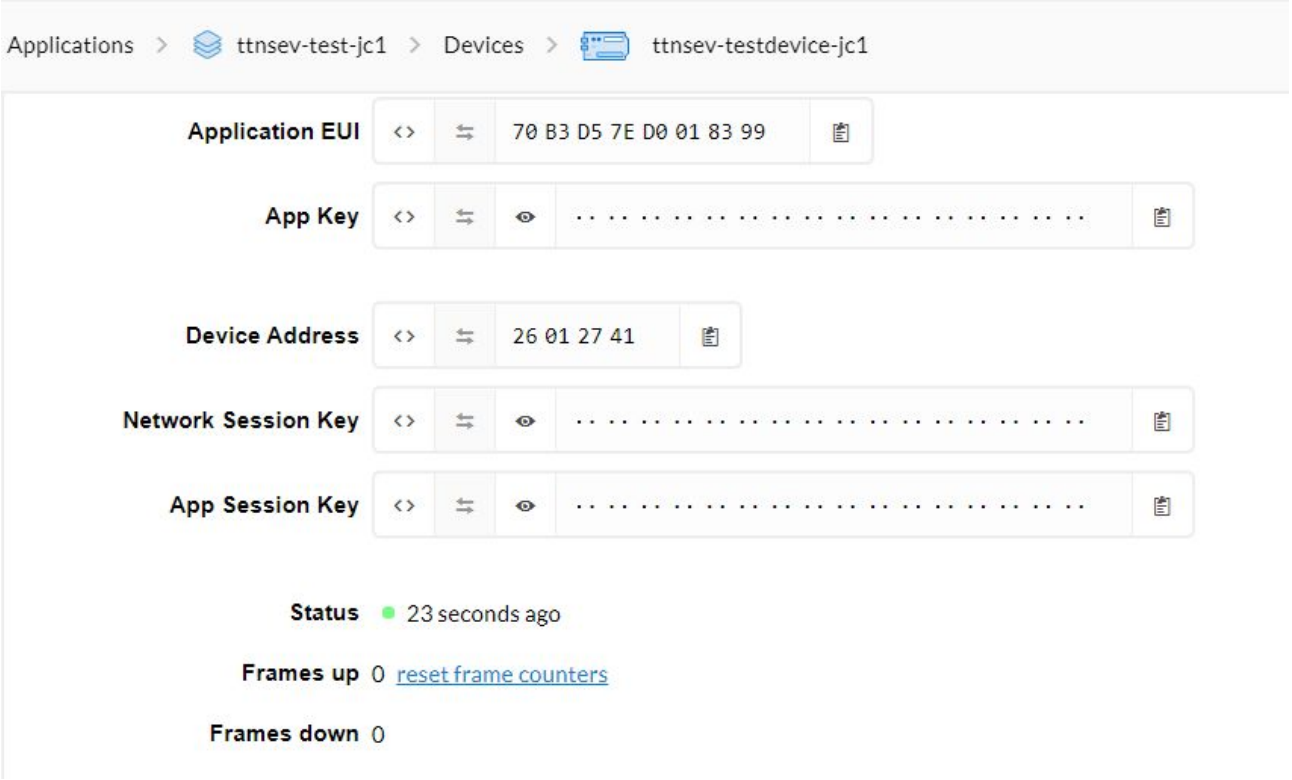

Figura 83. Comprobación del status del device

En la pestaña *Data* se observan las características de todos los datos de subida, tanto de su contenido como de los parámetros de la red LoRaWAN de la configuración. Además, también se muestra el tiempo estimado en aire de la señal LoRa entre el Adafruit y el *gateway* y los datos de cobertura estudiados en la parte de TTN Mapper (RSSI y SNR).

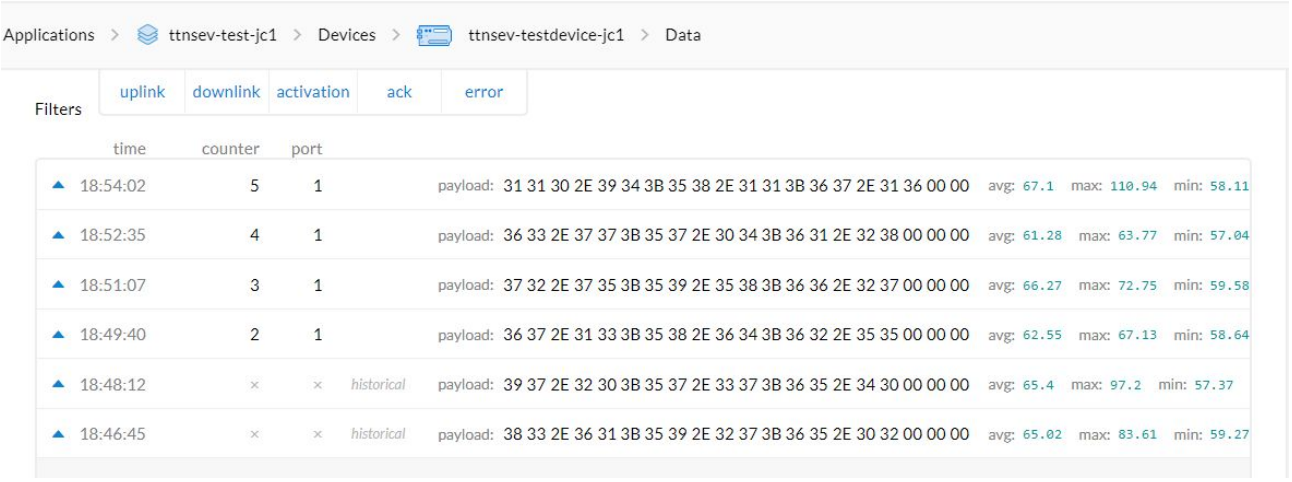

Figura 84. Comprobación de datos subidos del device a TTN

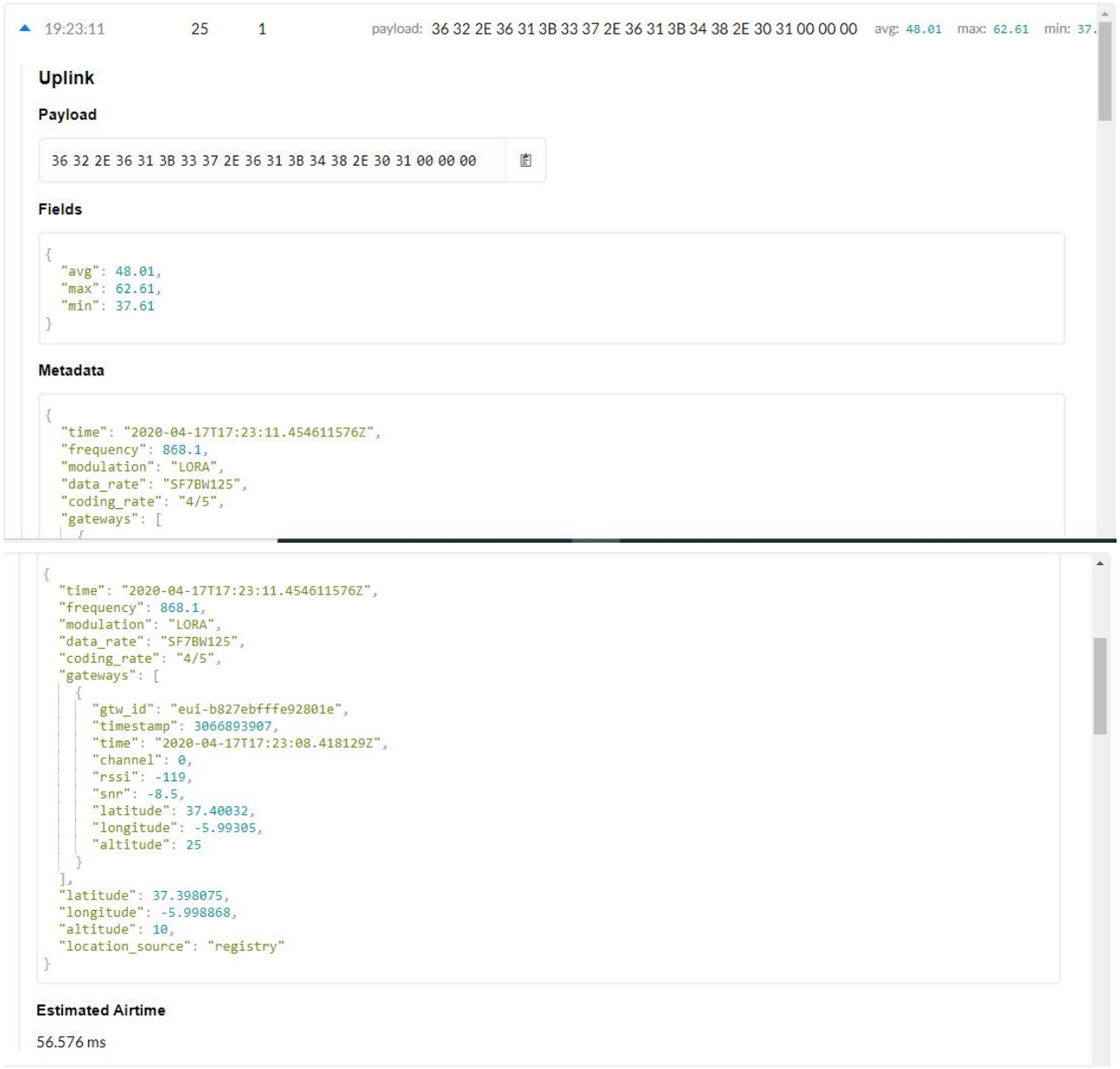

Figura 85. Carga útil, campos y metadatos recibidos en un envío concreto

5.2.4 Verificación en Data Storage

El hecho por el que se utiliza esta integración es porque los datos que recoge la pestaña *Data* del device es reseteada cada vez que se accede a TTN, por lo que no se podría consultar los datos de otros días si el cliente lo desease. En cambio, esta integración permite consultar datos de hasta 7 días atrás.

Cuando se inserta el espacio de tiempo de datos a visualizar, por ejemplo, en la sección *Query* y se pulsa *try it out!*, se obtiene el comando curl y la dirección web necesarios para la respuesta a esta solicitud.

Figura 86. Selección en Data Storage del periodo a obtener los datos recibidos

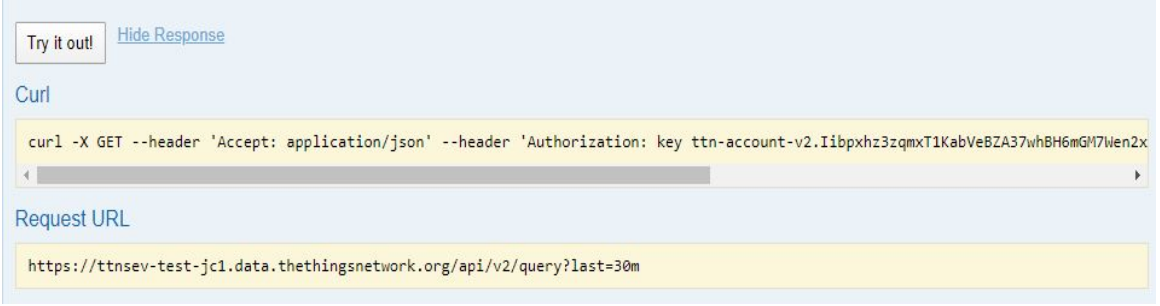

Figura 87. Métodos para obtener datos del Data Storage

Además, se pueden ver directamente estos datos en formato JSON en el cuerpo de la respuesta, que se muestra debajo de la URL de respuesta.

Figura 88. Visualización en formato JSON de los datos recibidos

5.3. Verificación de datos

Para la prueba realizada se han subido las medidas cada minuto y medio aproximadamente para tomar mayor número de muestras que en el caso real (cada 5 minutos). Y así poder hacer una estimación más amplia a partir de más muestras.

En el inicio de la prueba, en la plaza de la Encarnación de Sevilla, se midieron medias de entre 60 y 70 decibelios al lado del doble carril de vehículos, que queda verificado ya que el tráfico en este punto era constante, aunque al ser solo de dos carriles el ruido ambiental no es molesto. Y algunas medidas llegan puntualmente hasta los 110 decibelios de máximo, que puede ser debido al ruido motocicletas con el tubo de escape ruidoso o algún vehículo pesado, como por ejemplo el camión de la basura o el autobús urbano.

Response Body


```
    "avg": 66.27,
    "device_id": "ttnsev-testdevice-jc1",
    "max": 72.75,
    "min": 59.58,
    "raw": "NzIuNzU7NTkuNTg7NjYuMjcAAAA=",
    "time": "2020-04-17T16:51:07.722347805Z"
  },
  {
    "avg": 61.28,
    "device_id": "ttnsev-testdevice-jc1",
    "max": 63.77,
    "min": 57.04,
    "raw": "NjMuNzc7NTcuMDQ7NjEuMjgAAAA=",
    "time": "2020-04-17T16:52:35.174858507Z"
  },
  {
    "avg": 67.1,
    "device_id": "ttnsev-testdevice-jc1",
    "max": 110.94,
    "min": 58.11,
    "raw": "MTEwLjk0OzU4LjExOzY3LjE2AAA=",
```


Figura 89. Primera verificación de datos medidos

Cuando la prueba continuó hacia calles más estrechas con menos tráfico, la media de las medidas descendió y estaba comprendida entre los 52 y los 60 decibelios, que se debía a conversaciones entre ciudadanos y los electrodomésticos de los bajos de las viviendas en funcionamiento que se podían escuchar a través de las ventanas abiertas. El pico a 90 se podría deber a cualquier vehículo, y es elevado ya que al ser la calle estrecha el ruido del escape reverbera entre las paredes de esta en mayor medida en que lo haría cuando la calle es ancha.

```
{
  "avg": 52.24,
  "device_id": "ttnsev-testdevice-jc1",
  "max": 80.52,
  "min": 40.07,
  "raw": "ODAuNTI7NDAuMDc7NTIuMjQAAAA=",
  "time": "2020-04-17T16:59:52.5766238Z"
},
{
  "avg": 60.76,
  "device_id": "ttnsev-testdevice-jc1",
  "max": 77.5,
  "min": 46.05,
  "raw": "NzcuNTA7NDYuMDU7NjAuNzYAAAA=",
  "time": "2020-04-17T17:01:20.036951118Z"
},
{
  "avg": 60.15,
  "device_id": "ttnsev-testdevice-jc1",
  "max": 90.66,
  "min": 46.05,
```


Figura 90. Segunda verificación de datos medidos

Por último, más lejos del centro de la ciudad, en callejones aún más estrechos donde apenas hay circulación de peatones y vehículos la media baja a los 44-52 decibelios, que se traduce al ruido que podría haber en una clase, con el único ruido producido por el viento y los vecinos en el interior de sus viviendas realizando labores domésticas.

Response Body


```
{
  "avg": 44.25,
  "device_id": "ttnsev-testdevice-jc1",
  "max": 52.42,
  "min": 36.89,
  "raw": "NTIuNDI7MzYuODk7NDQuMjUAAAA=",
  "time": "2020-04-17T17:20:16.673033001Z"
},
{
  "avg": 48.01,
  "device_id": "ttnsev-testdevice-jc1",
  "max": 62.61,
  "min": 37.61,
  "raw": "NjIuNjE7MzcuNjE7NDguMDEAAAA=",
  "time": "2020-04-17T17:23:11.454611576Z"
},
{
  "avg": 52.22,
  "device_id": "ttnsev-testdevice-jc1",
  "max": 65.18,
  "min": 37.38,
```


Figura 91. Tercera verificación de datos medidos

6 CONCLUSIONES FINALES

6.1. Conclusiones del proyecto

Esta aplicación puede servir, hablando del interés personal del ciudadano, a la hora de gestionar la calidad de vida ya que en ciudades grandes puede resultar incómodo el hecho de mudarse a un entorno con una elevada contaminación acústica. Por otra parte, esta información también puede ser utilizada por el Ayuntamiento de la ciudad para impulsar planes de acción en materia de la contaminación acústica. Ya sea mediante sensibilización y educación contra el ruido, movilidad sostenible, actuaciones de control del ruido provocado por el ocio nocturno, etc. Esto al final se traduce en un incremento de la calidad de vida, que es conseguido como se ha demostrado en este proyecto sin una infraestructura costosa y difícil de mantener como podría ser la de las grandes operadoras de telecomunicación.

También cabe destacar la sencillez con la que el cliente puede instalar su dispositivo. El cliente recibirá un paquete en el que vendrá el dispositivo totalmente montado con el firmware de la aplicación ya instalado. No necesita ningún conocimiento sobre programación, señales o electrónica, y las únicas instrucciones que recibirá serán recomendaciones sobre dónde situar su dispositivo para que alcance la cobertura necesaria de la red LoRa y los credenciales de TTN asignados al dispositivo en concreto. Lo más difícil que tendrá que realizar será copiar estos credenciales en la librería credentials.h, editándose con cualquier editor de texto que el cliente disponga. Ya a partir de TTN, la aplicación y su integración correspondiente para volcar los datos a la web será gestionada por la empresa que comercialice este producto.

Como última conclusión, ver que el cliente se pone en contacto directo con este dispositivo, su funcionamiento y sus ventajas, por lo que, indirectamente, realiza un aprendizaje sobre IoT y *smart city.* Esto podría animar a la sociedad del futuro al estudio y familiarización sobre estas tecnologías, haciendo comprender empíricamente a los ciudadanos que su calidad de vida puede mejorar tanto como el desarrollo tecnológico del momento lo permita.

6.2. Líneas de avance

Este proyecto no acaba aquí, ya que este TFG se ha basado únicamente en el diseño técnico. Y a partir de este punto habría que seleccionar una base de datos en la que los datos persistan más de una semana (en el caso del TFG basta con una semana para, únicamente, comprobar y verificar datos) y hacer un diseño de página web donde el cliente pueda ver gráficos temporales sobre los datos subidos a TTN de los *devices* repartidos por la ciudad. Y como estos dispositivos no tienen geolocalización, esta página web debe permitir al cliente asociar el device ID de su dispositivo a la localización física de este. El resultado final de esta web es un mapa a disposición pública con valores a tiempo real de la contaminación acústica medida por cada cliente.

La gran ventaja de este producto final sería el bajo precio de venta al público, ya que como se ha visto en este TFG el desarrollo técnico tiene un bajo coste gracias a las características del hardware y la red. Principalmente lo que le da una buena imagen a este proyecto es la red utilizada, ya que LoRa se ajusta a las premisas de red IoT de sensores y es aprovechada técnica y económicamente a la hora de monitorizar cualquier parámetro medible mediante sensores. Esto se traduce en la eficiencia a la hora de crear cualquier aplicación, que en el caso de este TFG estaría dentro del concepto de *smart city* o ciudad inteligente.

REFERENCIAS

[1] OSI: «IoT, el universo conectado,» https://www.osi.es/es/actualidad/blog/2018/06/27/iot-el-universo-conectado , 27 de Junio de 2018.

[2] LoRaWAN® Specification Documents: https://lora-alliance.org/lorawan-for-developers

[3] TheThingsNetwork: https://www.thethingsnetwork.org/docs/

[4] Estudio Things Matter: https://iot.telefonica.com/es/whats-new/multimedia/estudio-things-matter-2019/

[5] Adafruit Feather M0 RFM9x LoRa Radio: https://learn.adafruit.com/adafruit-feather-m0-radio-with-lora-radio-module

[6] Semtech SX1276 datasheet: https://semtech.my.salesforce.com/sfc/p/#E0000000JelG/a/2R0000001OKs/Bs97dmPXeatnbdoJNVMIDaKDlQz8q1N_gxDcgqi7g2o

[7] Conexionado básico de Adafruit: https://learn.adafruit.com/the-things-network-for-feather/arduino-wiring

[8] Física del sonido: https://www.miteco.gob.es/es/calidad-y-evaluacion-ambiental/temas/atmosfera-y-calidad-del-aire/contaminacion_acustica_tcm30-185098.pdf

[9] Especificación de IBM LoRaWAN in C (LMIC) : https://lora-developers.semtech.com/resources/tools/basic-mac/welcome-basic-mac/

[10] Aplicaciones basadas en LoRaWAN: https://www.semtech.com/lora/lora-applications

ANEXO: CÓDIGO DE LIBRERÍAS

- Librería principal LMiC:

```
/*
 * Copyright (c) 2014-2016 IBM Corporation.
 * Copyright (c) 2016 Matthijs Kooijman.
 * Copyright (c) 2016-2019 MCCI Corporation.
 * All rights reserved.
 *
 *  Redistribution and use in source and binary forms, with or without
 *  modification, are permitted provided that the following conditions are met:
 *   * Redistributions of source code must retain the above copyright
 *     notice, this list of conditions and the following disclaimer.
 *   * Redistributions in binary form must reproduce the above copyright
 *     notice, this list of conditions and the following disclaimer in the
 *     documentation and/or other materials provided with the distribution.
 *   * Neither the name of the <organization> nor the
 *     names of its contributors may be used to endorse or promote products
 *     derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL <COPYRIGHT HOLDER> BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

//! @file
//! @brief LMIC API

#ifndef _lmic_h_
#define _lmic_h_

#include "oslmic.h"
#include "lorabase.h"

```

```
#if LMIC_DEBUG_LEVEL > 0 || LMIC_X_DEBUG_LEVEL > 0
# if defined(LMIC_DEBUG_INCLUDE)
#   define LMIC_STRINGIFY_(x) #x
#   define LMIC_STRINGIFY(x) LMIC_STRINGIFY_(x)
#   include LMIC_STRINGIFY(LMIC_DEBUG_INCLUDE)
# endif
#  ifdef LMIC_DEBUG_PRINTF_FN
     extern void LMIC_DEBUG_PRINTF_FN(const char *f, ...);
#  endif // ndef LMIC_DEBUG_PRINTF_FN
#endif

// if LMIC_DEBUG_PRINTF is now defined, just use it. This lets you do anything
// you like with a sufficiently crazy header file.
#if LMIC_DEBUG_LEVEL > 0
# ifndef LMIC_DEBUG_PRINTF
//  otherwise, check whether someone configured a print-function to be used,
//  and use it if so.
#   ifdef LMIC_DEBUG_PRINTF_FN
#     define LMIC_DEBUG_PRINTF(f, ...) LMIC_DEBUG_PRINTF_FN(f, ## __VA_ARGS__)
#     ifndef LMIC_DEBUG_INCLUDE // If you use LMIC_DEBUG_INCLUDE, put the declaration in there
        void LMIC_DEBUG_PRINTF_FN(const char *f, ...);
#     endif // ndef LMIC_DEBUG_INCLUDE
#   else // ndef LMIC_DEBUG_PRINTF_FN
//    if there's no other info, just use printf. In a pure Arduino environment,
//    that's what will happen.
#     include <stdio.h>
#     define LMIC_DEBUG_PRINTF(f, ...) printf(f, ## __VA_ARGS__)
#   endif // ndef LMIC_DEBUG_PRINTF_FN
# endif // ndef LMIC_DEBUG_PRINTF
# ifndef LMIC_DEBUG_FLUSH
#   ifdef LMIC_DEBUG_FLUSH_FN
#     define LMIC_DEBUG_FLUSH() LMIC_DEBUG_FLUSH_FN()
#   else // ndef LMIC_DEBUG_FLUSH_FN
//    if there's no other info, assume that flush is not needed.
#     define LMIC_DEBUG_FLUSH() do { ; } while (0)
#   endif // ndef LMIC_DEBUG_FLUSH_FN
# endif // ndef LMIC_DEBUG_FLUSH
#else // LMIC_DEBUG_LEVEL == 0
// If debug level is zero, printf and flush expand to nothing.
# define LMIC_DEBUG_PRINTF(f, ...)      do { ; } while (0)
# define LMIC_DEBUG_FLUSH()             do { ; } while (0)
#endif // LMIC_DEBUG_LEVEL == 0

//
// LMIC_X_DEBUG_LEVEL enables additional, special print functions for debugging
// RSSI features. This is used sparingly.
#if LMIC_X_DEBUG_LEVEL > 0
#  ifdef LMIC_DEBUG_PRINTF_FN
#    define LMIC_X_DEBUG_PRINTF(f, ...) LMIC_DEBUG_PRINTF_FN(f, ## __VA_ARGS__)
#  else
#    error "LMIC_DEBUG_PRINTF_FN must be defined for LMIC_X_DEBUG_LEVEL > 0."
#  endif
#else
#  define LMIC_X_DEBUG_PRINTF(f, ...)  do {;} while(0)
#endif

#ifdef __cplusplus
extern "C"{
#endif

// LMIC version -- this is ths IBM LMIC version
#define LMIC_VERSION_MAJOR 1
#define LMIC_VERSION_MINOR 6
#define LMIC_VERSION_BUILD 1468577746

// Arduino LMIC version
#define ARDUINO_LMIC_VERSION_CALC(major, minor, patch, local)   \
    ((((major)*UINT32_C(1)) << 24) | (((minor)*UINT32_C(1)) << 16) | (((patch)*UINT32_C(1)) << 8) | (((local)*UINT32_C(1)) << 0))

#define ARDUINO_LMIC_VERSION    ARDUINO_LMIC_VERSION_CALC(3, 1, 0, 0)   /* v3.1.0.0 */

#define ARDUINO_LMIC_VERSION_GET_MAJOR(v)   \
    ((((v)*UINT32_C(1)) >> 24u) & 0xFFu)
```

```
112
113 #define ARDUINO_LMIC_VERSION_GET_MINOR(v)   \
114     ((((v)*UINT32_C(1)) >> 16u) & 0xFFu)
115
116 #define ARDUINO_LMIC_VERSION_GET_PATCH(v)   \
117     ((((v)*UINT32_C(1)) >> 8u) & 0xFFu)
118
119 #define ARDUINO_LMIC_VERSION_GET_LOCAL(v)   \
120     ((v) & 0xFFu)
121
122 //! Only For Antenna Tuning Tests !
123 //#define CFG_TxContinuousMode 1
124
125 // since this was annouunced as the API variable, we keep it. But it's not used,
126 // MAX_LEN_FRAME is what the code uses.
127 enum { MAX_FRAME_LEN      =  MAX_LEN_FRAME };   //!< Library cap on max frame length
128
129 enum { TXCONF_ATTEMPTS    =   8 };   //!< Transmit attempts for confirmed frames
130 enum { MAX_MISSED_BCNS    =  (2 * 60 * 60 + 127) / 128 };   //!< threshold for dropping out of class B, triggering rejoin requests
131                                      // note that we need 100 ppm timing accuracy for
132                                      // this, to keep the timing error to +/- 700ms.
133 enum { MAX_RXSYMS         = 350 };   // Stop tracking beacon if sync error grows beyond this. A 0.4% clock error
134                                      // at SF9.125k means 512 ms; one sybol is 4.096 ms,
135                                      // so this needs to be at least 125 for an STM32L0.
136                                      // And for 100ppm clocks and 2 hours of beacon misses,
137                                      // this needs to accomodate 1.4 seconds of error at
138                                      // 4.096 ms/sym or at least 342 symbols.
139
140 enum { LINK_CHECK_CONT    =  0  ,    // continue with this after reported dead link
141        LINK_CHECK_DEAD    =  32 ,    // after this UP frames and no response to ack from NWK assume link is dead (ADR_ACK_DELAY)
142        LINK_CHECK_UNJOIN_MIN = LINK_CHECK_DEAD + 4,         // this is the minimum value of LINK_CHECK_UNJOIN if we parameterize
143        LINK_CHECK_UNJOIN  =  LINK_CHECK_DEAD + (3 * 240),   // after this many UP frames and no response, switch to join (by default)
144        LINK_CHECK_INIT    = -64 ,    // UP frame count until we ask for ack (ADR_ACK_LIMIT)
145        LINK_CHECK_OFF     =-128 };   // link check disabled
146
147 enum { TIME_RESYNC        = 6*128 }; // secs
148 enum { TXRX_GUARD_ms      =  6000 };  // msecs - don't start TX-RX transaction before beacon
149 enum { JOIN_GUARD_ms      =  9000 };  // msecs - don't start Join Req/Acc transaction before beacon
150 enum { TXRX_BCNEXT_secs   =     2 };  // secs - earliest start after beacon time
151 enum { RETRY_PERIOD_secs  =     3 };  // secs - random period for retrying a confirmed send
152
153 #if CFG_LMIC_EU_like // EU868 spectrum ================================================
154
155 enum { MAX_CHANNELS = 16 };      //!< Max supported channels
156 enum { MAX_BANDS    =  4 };
157
158 enum { LIMIT_CHANNELS = (1<<4) };   // EU868 will never have more channels
159 //! \internal
160 struct band_t {
161     u2_t     txcap;     // duty cycle limitation: 1/txcap
162     s1_t     txpow;     // maximum TX power
163     u1_t     lastchnl;  // last used channel
164     ostime_t avail;     // channel is blocked until this time
165 };
166 TYPEDEF_xref2band_t; //!< \internal
167
168 struct lmic_saved_adr_state_s {
169     u4_t        channelFreq[MAX_CHANNELS];
170     u2_t        channelMap;
171 };
172
173 #elif CFG_LMIC_US_like  // US915 spectrum ================================================
174
175 enum { MAX_XCHANNELS = 2 };      // extra channels in RAM, channels 0-71 are immutable
176
177 struct lmic_saved_adr_state_s {
178     u2_t        channelMap[(72+MAX_XCHANNELS+15)/16];  // enabled bits
179     u2_t        activeChannels125khz;
180     u2_t        activeChannels500khz;
181 };
182
183 #endif // ================================================================================
184
185 typedef struct lmic_saved_adr_state_s   lmic_saved_adr_state_t;
186
```

```
// Keep in sync with evdefs.hpp::drChange
enum { DRCHG_SET, DRCHG_NOJACC, DRCHG_NOACK, DRCHG_NOADRACK, DRCHG_NWKCMD, DRCHG_FRAMESIZE };
enum { KEEP_TXPOW = -128 };


#if !defined(DISABLE_PING)
//! \internal
struct rxsched_t {
    dr_t     dr;
    u1_t     intvExp;   // 0..7
    u1_t     slot;      // runs from 0 to 128
    rxsyms_t rxsyms;
    ostime_t rxbase;
    ostime_t rxtime;    // start of next spot
    u4_t     freq;
};
TYPEDEF_xref2rxsched_t;  //!< \internal
#endif // !DISABLE_PING


#if !defined(DISABLE_BEACONS)
//! Parsing and tracking states of beacons.
enum { BCN_NONE    = 0x00,   //!< No beacon received
       BCN_PARTIAL = 0x01,   //!< Only first (common) part could be decoded (info,lat,lon invalid/previous)
       BCN_FULL    = 0x02,   //!< Full beacon decoded
       BCN_NODRIFT = 0x04,   //!< No drift value measured yet
       BCN_NODDIFF = 0x08 }; //!< No differential drift measured yet
//! Information about the last and previous beacons.
struct bcninfo_t {
    ostime_t txtime;  //!< Time when the beacon was sent
    u4_t     time;    //!< GPS time in seconds of last beacon (received or surrogate)
    s4_t     lat;     //!< Lat field of last beacon (valid only if BCN_FULL set)
    s4_t     lon;     //!< Lon field of last beacon (valid only if BCN_FULL set)
    s1_t     rssi;    //!< Adjusted RSSI value of last received beacon
    s1_t     snr;     //!< Scaled SNR value of last received beacon
    u1_t     flags;   //!< Last beacon reception and tracking states. See BCN_* values.
    //
    u1_t     info;    //!< Info field of last beacon (valid only if BCN_FULL set)
};
#endif // !DISABLE_BEACONS

// purpose of receive window - lmic_t.rxState
enum { RADIO_RST=0, RADIO_TX=1, RADIO_RX=2, RADIO_RXON=3, RADIO_TX_AT=4, };
// Netid values /  lmic_t.netid
enum { NETID_NONE=(int)~0U, NETID_MASK=(int)0xFFFFFF };
// MAC operation modes (lmic_t.opmode).
enum { OP_NONE     = 0x0000,
       OP_SCAN     = 0x0001, // radio scan to find a beacon
       OP_TRACK    = 0x0002, // track my networks beacon (netid)
       OP_JOINING  = 0x0004, // device joining in progress (blocks other activities)
       OP_TXDATA   = 0x0008, // TX user data (buffered in pendTxData)
       OP_POLL     = 0x0010, // send empty UP frame to ACK confirmed DN/fetch more DN data
       OP_REJOIN   = 0x0020, // occasionally send JOIN REQUEST
       OP_SHUTDOWN = 0x0040, // prevent MAC from doing anything
       OP_TXRXPEND = 0x0080, // TX/RX transaction pending
       OP_RNDTX    = 0x0100, // prevent TX lining up after a beacon
       OP_PINGINI  = 0x0200, // pingable is initialized and scheduling active
       OP_PINGABLE = 0x0400, // we're pingable
       OP_NEXTCHNL = 0x0800, // find a new channel
       OP_LINKDEAD = 0x1000, // link was reported as dead
       OP_TESTMODE = 0x2000, // developer test mode
       OP_UNJOIN   = 0x4000, // unjoin and rejoin on next engineUpdate().
};
// TX-RX transaction flags - report back to user
enum { TXRX_ACK    = 0x80,   // confirmed UP frame was acked
       TXRX_NACK   = 0x40,   // confirmed UP frame was not acked
       TXRX_NOPORT = 0x20,   // set if a frame with a port was RXed, clr if no frame/no port
       TXRX_PORT   = 0x10,   // set if a frame with a port was RXed, LMIC.frame[LMIC.dataBeg-1] => port
       TXRX_LENERR = 0x08,   // set if frame was discarded due to length error.
       TXRX_PING   = 0x04,   // received in a scheduled RX slot
       TXRX_DNW2   = 0x02,   // received in 2dn DN slot
       TXRX_DNW1   = 0x01,   // received in 1st DN slot
};

```

```
// Event types for event callback
enum _ev_t { EV_SCAN_TIMEOUT=1, EV_BEACON_FOUND,
             EV_BEACON_MISSED, EV_BEACON_TRACKED, EV_JOINING,
             EV_JOINED, EV_RFU1, EV_JOIN_FAILED, EV_REJOIN_FAILED,
             EV_TXCOMPLETE, EV_LOST_TSYNC, EV_RESET,
             EV_RXCOMPLETE, EV_LINK_DEAD, EV_LINK_ALIVE, EV_SCAN_FOUND,
             EV_TXSTART, EV_TXCANCELED, EV_RXSTART, EV_JOIN_TXCOMPLETE };
typedef enum _ev_t ev_t;

// this macro can be used to initalize a normal table of event strings
#define LMIC_EVENT_NAME_TABLE__INIT                                     \
    "<<zero>>",                                                         \
    "EV_SCAN_TIMEOUT", "EV_BEACON_FOUND",                               \
    "EV_BEACON_MISSED", "EV_BEACON_TRACKED", "EV_JOINING",              \
    "EV_JOINED", "EV_RFU1", "EV_JOIN_FAILED", "EV_REJOIN_FAILED",       \
    "EV_TXCOMPLETE", "EV_LOST_TSYNC", "EV_RESET",                       \
    "EV_RXCOMPLETE", "EV_LINK_DEAD", "EV_LINK_ALIVE", "EV_SCAN_FOUND",  \
    "EV_TXSTART", "EV_TXCANCELED", "EV_RXSTART", "EV_JOIN_TXCOMPLETE"

// if working on an AVR (or worried about it), you can use this multi-zero
// string and put this in a single const F() string.  Index through this
// counting up from 0, until you get to the entry you want or to an
// entry that begins with a \0.
#define LMIC_EVENT_NAME_MULTISZ__INIT                                   \
    "<<zero>>\0"                                                        \
    "EV_SCAN_TIMEOUT\0" "EV_BEACON_FOUND\0"                             \
    "EV_BEACON_MISSED\0" "EV_BEACON_TRACKED\0" "EV_JOINING\0"           \
    "EV_JOINED\0" "EV_RFU1\0" "EV_JOIN_FAILED\0" "EV_REJOIN_FAILED\0"   \
    "EV_TXCOMPLETE\0" "EV_LOST_TSYNC\0" "EV_RESET\0"                    \
    "EV_RXCOMPLETE\0" "EV_LINK_DEAD\0" "EV_LINK_ALIVE\0" "EV_SCAN_FOUND\0"  \
    "EV_TXSTART\0" "EV_TXCANCELED\0" "EV_RXSTART\0" "EV_JOIN_TXCOMPLETE\0"

enum {
    LMIC_ERROR_SUCCESS = 0,
    LMIC_ERROR_TX_BUSY = -1,
    LMIC_ERROR_TX_TOO_LARGE = -2,
    LMIC_ERROR_TX_NOT_FEASIBLE = -3,
    LMIC_ERROR_TX_FAILED = -4,
```

```
};

typedef int lmic_tx_error_t;

#define LMIC_ERROR_NAME__INIT                                           \
    "LMIC_ERROR_SUCCESS",                                               \
    "LMIC_ERROR_TX_BUSY",                                               \
    "LMIC_ERROR_TX_TOO_LARGE",                                          \
    "LMIC_ERROR_TX_NOT_FEASIBLE",                                       \
    "LMIC_ERROR_TX_FAILED"

#define LMIC_ERROR_NAME_MULTISZ__INIT                                   \
    "LMIC_ERROR_SUCCESS\0"                                              \
    "LMIC_ERROR_TX_BUSY\0"                                              \
    "LMIC_ERROR_TX_TOO_LARGE\0"                                         \
    "LMIC_ERROR_TX_NOT_FEASIBLE\0"                                      \
    "LMIC_ERROR_TX_FAILED"

enum {
    LMIC_BEACON_ERROR_INVALID   = -2,
    LMIC_BEACON_ERROR_WRONG_NETWORK = -1,
    LMIC_BEACON_ERROR_SUCCESS_PARTIAL = 0,
    LMIC_BEACON_ERROR_SUCCESS_FULL = 1,
};

typedef s1_t lmic_beacon_error_t;

static inline bit_t LMIC_BEACON_SUCCESSFUL(lmic_beacon_error_t e) {
    return e < 0;
}

// LMIC_CFG_max_clock_error_ppm
#if !defined(LMIC_CFG_max_clock_error_ppm)
# define LMIC_CFG_max_clock_error_ppm   2000    /* max clock error: 0.2% (2000 ppm) */
#endif


enum {
        // This value represents 100% error in LMIC.clockError
        MAX_CLOCK_ERROR = 65536,
        //! \brief maximum clock error that users can specify: 2000 ppm (0.2%).
        //! \details This is the limit for clock error, unless LMIC_ENABLE_arbitrary_clock_error is set.
        //! The default is 4,000 ppm, which is .004, or 0.4%; this is what you get on an
        //! STM32L0 running with the HSI oscillator after cal. If your clock error is bigger,
        //! usually you want to calibrate it so that millis() and micros() are reasonably
        //! accurate. Important: do not use clock error to compensate for late serving
        //! of the LMIC. If you see that LMIC.radio.rxlate_count is increasing, you need
        //! to adjust your application logic so the LMIC gets serviced promptly when a
        //! Class A downlink (or beacon) is pending.
        LMIC_kMaxClockError_ppm = 4000,
};

// callbacks for client alerts.
// types and functions are always defined, to reduce #ifs in example code and libraries.
typedef void LMIC_ABI_STD lmic_rxmessage_cb_t(void *pUserData, uint8_t port, const uint8_t *pMessage, size_t nMessage);
typedef void LMIC_ABI_STD lmic_txmessage_cb_t(void *pUserData, int fSuccess);
typedef void LMIC_ABI_STD lmic_event_cb_t(void *pUserData, ev_t e);

// network time request callback function
// defined unconditionally, because APIs and types can't change based on config.
// This is called when a time-request succeeds or when we get a downlink
// without time request, "completing" the pending time request.
typedef void LMIC_ABI_STD lmic_request_network_time_cb_t(void *pUserData, int flagSuccess);

// how the network represents time.
typedef u4_t lmic_gpstime_t;

// rather than deal with 1/256 second tick, we adjust ostime back
// (as it's high res) to match tNetwork.
typedef struct lmic_time_reference_s lmic_time_reference_t;

```

```
370 struct lmic_time_reference_s {
371     // our best idea of when we sent the uplink (end of packet).
372     ostime_t tLocal;
373     // the network's best idea of when we sent the uplink.
374     lmic_gpstime_t tNetwork;
375 };
376
377 enum lmic_request_time_state_e {
378     lmic_RequestTimeState_idle = 0,     // we're not doing anything
379     lmic_RequestTimeState_tx,           // we want to tx a time request on next uplink
380     lmic_RequestTimeState_rx,           // we have tx'ed, next downlink completes.
381     lmic_RequestTimeState_success       // we sucessfully got time.
382 };
383
384 typedef u1_t lmic_request_time_state_t;
385
386 enum lmic_engine_update_state_e {
387     lmic_EngineUpdateState_idle = 0,    // engineUpdate is idle.
388     lmic_EngineUpdateState_busy = 1,    // engineUpdate is busy, but has not been reentered.
389     lmic_EngineUpdateState_again = 2,   // engineUpdate is busy, and has to be evaluated again.
390 };
391
392 typedef u1_t lmic_engine_update_state_t;
393
394 /*
395
396 Structure:  lmic_client_data_t
397
398 Function:
399         Holds LMIC client data that must live through LMIC_reset().
400
401 Description:
402         There are a variety of client registration linkage items that
403         must live through LMIC_reset(), because LMIC_reset() is called
404         at frame rollover time.  We group them together into a structure
405         to make copies easy.
406
407 */
```

```
408
409 //! abstract type for collection of client data that survives LMIC_reset().
410 typedef struct lmic_client_data_s lmic_client_data_t;
411
412 //! contents of lmic_client_data_t
413 struct lmic_client_data_s {
414
415     /* pointer-width things come first */
416 #if LMIC_ENABLE_DeviceTimeReq
417     lmic_request_network_time_cb_t *pNetworkTimeCb; //! call-back routine for network time
418     void        *pNetworkTimeUserData;              //! call-back data for network time.
419 #endif
420
421 #if LMIC_ENABLE_user_events
422     lmic_event_cb_t     *eventCb;           //! user-supplied callback function for events.
423     void                *eventUserData;     //! data for eventCb
424     lmic_rxmessage_cb_t *rxMessageCb;       //! user-supplied message-received callback
425     void                *rxMessageUserData; //! data for rxMessageCb
426     lmic_txmessage_cb_t *txMessageCb;       //! transmit-complete message handler; reset on each tx complete.
427     void                *txMessageUserData; //! data for txMessageCb.
428 #endif // LMIC_ENABLE_user_events
429
430     /* next we have things that are (u)int32_t */
431     /* none at the moment */
432
433     /* next we have things that are (u)int16_t */
434
435     u2_t        clockError;                 //! Inaccuracy in the clock. CLOCK_ERROR_MAX represents +/-100% error
436
437     /* finally, things that are (u)int8_t */
438     /* none at the moment */
439 };
440
```

```
441 /*
442
443 Structure:  lmic_radio_data_t
444
445 Function:
446     Holds LMIC radio driver.
447
448 Description:
449     Eventually this will be used for all portable things for the radio driver,
450     but for now it's where we can start to add things.
451
452 */
453
454 typedef struct lmic_radio_data_s lmic_radio_data_t;
455
456 struct lmic_radio_data_s {
457     // total os ticks of accumulated delay error. Can overflow!
458     ostime_t    rxlate_ticks;
459     // number of rx late launches.
460     unsigned    rxlate_count;
461     // total os ticks of accumulated tx delay error. Can overflow!
462     ostime_t    txlate_ticks;
463     // number of tx late launches.
464     unsigned    txlate_count;
465 };
466
467 /*
468
469 Structure:  lmic_t
470
471 Function:
472         Provides the instance data for the LMIC.
473
474 */
475
```

```
476 struct lmic_t {
477     // client setup data, survives LMIC_reset().
478     lmic_client_data_t  client;
479
480     // the OS job object. pointer alignment.
481     osjob_t     osjob;
482
483 #if !defined(DISABLE_BEACONS)
484     bcninfo_t   bcninfo;      // Last received beacon info
485 #endif
486
487 #if !defined(DISABLE_PING)
488     rxsched_t   ping;         // pingable setup
489 #endif
490
491     // the radio driver portable context
492     lmic_radio_data_t   radio;
493
494     /* (u)int32_t things */
495
496     // Radio settings TX/RX (also accessed by HAL)
497     ostime_t    txend;
498     ostime_t    rxtime;
499
500     // LBT info
501     ostime_t    lbt_ticks;      // ticks to listen
502
503     u4_t        freq;
504
505     ostime_t    globalDutyAvail; // time device can send again
506
507     u4_t        netid;        // current network id (~0 - none)
508     devaddr_t   devaddr;
509     u4_t        seqnoDn;      // device level down stream seqno
510     u4_t        seqnoUp;
511     u4_t        dn2Freq;
512
```

```
513 #if !defined(DISABLE_BEACONS)
514     ostime_t    bcnRxtime;
515 #endif
516
517 #if LMIC_ENABLE_DeviceTimeReq
518     // put here for alignment, to reduce RAM use.
519     ostime_t    localDeviceTime;    // the LMIC.txend value for last DeviceTimeAns
520     lmic_gpstime_t netDeviceTime;   // the netDeviceTime for lastDeviceTimeAns
521                                     // zero ==> not valid.
522 #endif // LMIC_ENABLE_DeviceTimeReq
523
524     // Channel scheduling -- very much private
525 #if CFG_LMIC_EU_like
526     band_t      bands[MAX_BANDS];
527     u4_t        channelFreq[MAX_CHANNELS];
528 #if !defined(DISABLE_MCMD_DlChannelReq)
529     u4_t        channelDlFreq[MAX_CHANNELS];
530 #endif
531     // bit map of enabled datarates for each channel
532     u2_t        channelDrMap[MAX_CHANNELS];
533     u2_t        channelMap;
534 #elif CFG_LMIC_US_like
535     u4_t        xchFreq[MAX_XCHANNELS];    // extra channel frequencies (if device is behind a repeater)
536     u2_t        xchDrMap[MAX_XCHANNELS];   // extra channel datarate ranges  ---XXX: ditto
537     u2_t        channelMap[(72+MAX_XCHANNELS+15)/16];  // enabled bits
538     u2_t        activeChannels125khz;
539     u2_t        activeChannels500khz;
540 #endif
541
542     /* (u)int16_t things */
543     rps_t       rps;            // radio parameter selections: SF, BW, CodingRate, NoCrc, implicit hdr
544     u2_t        opmode;         // engineUpdate() operating mode flags
545     u2_t        devNonce;       // last generated nonce
546
547     s2_t        adrAckReq;      // counter for link integrity tracking (LINK_CHECK_OFF=off)
548
```

```
#if !defined(DISABLE_BEACONS)
    s2_t        drift;          // last measured drift
    s2_t        lastDriftDiff;
    s2_t        maxDriftDiff;
    rxsyms_t    bcnRxsyms;      //
#endif

    /* (u)int8_t things */
    lmic_engine_update_state_t engineUpdateState;   // state of the engineUpdate() evaluator.
    s1_t        rssi;
    s1_t        snr;            // LMIC.snr is SNR times 4
    rxsyms_t    rxsyms;         // symbols for receive timeout.
    u1_t        dndr;
    s1_t        txpow;          // transmit dBm (administrative)
    s1_t        radio_txpow;    // the radio driver's copy of txpow, in dB limited by adrTxPow, and
                // also adjusted for EIRP/antenna gain considerations.
                // This is just the radio's idea of power. So if you are
                // controlling EIRP, and you have 3 dB antenna gain, this
                // needs to reduced by 3 dB.
    s1_t        lbt_dbmax;      // max permissible dB on our channel (eg -80)

    u1_t        txChnl;          // channel for next TX
    u1_t        globalDutyRate;  // max rate: 1/2^k

    u1_t        upRepeat;     // configured up repeat
    s1_t        adrTxPow;     // ADR adjusted TX power
    u1_t        datarate;     // current data rate
    u1_t        errcr;        // error coding rate (used for TX only)
    u1_t        rejoinCnt;    // adjustment for rejoin datarate

    u1_t        upRepeatCount;  // current up-repeat
    bit_t       initBandplanAfterReset; // cleared by LMIC_reset(), set by first join. See issue #244

    u1_t        pendTxPort;
    u1_t        pendTxConf;   // confirmed data
    u1_t        pendTxLen;    // count of bytes in pendTxData.
    u1_t        pendTxData[MAX_LEN_PAYLOAD];

    u1_t        pendMacLen;         // number of bytes of pending Mac response data
    bit_t       pendMacPiggyback;   // received on port 0 or piggyback?
    // response data if piggybacked
    u1_t        pendMacData[LWAN_FCtrl_FOptsLen_MAX];

    u1_t        nwkKey[16];   // network session key
    u1_t        artKey[16];   // application router session key

    u1_t        dnConf;       // dn frame confirm pending: LORA::FCT_ACK or 0
    u1_t        lastDnConf;   // downlink with seqnoDn-1 requested confirmation
    u1_t        adrChanged;

    u1_t        rxDelay;      // Rx delay after TX

    u1_t        margin;
    s1_t        devAnsMargin; // SNR value between -32 and 31 (inclusive) for the last successfully received DevStatusReq command
    u1_t        adrEnabled;
    u1_t        moreData;     // NWK has more data pending
#if LMIC_ENABLE_TxParamSetupReq
    u1_t        txParam;        // the saved TX param byte.
#endif
#if LMIC_ENABLE_DeviceTimeReq
    lmic_request_time_state_t txDeviceTimeReqState;  // current state, initially idle.
    u1_t        netDeviceTimeFrac;     // updated on any DeviceTimeAns.
#endif

    // rx1DrOffset is the offset from uplink to downlink datarate
    u1_t        rx1DrOffset;  // captured from join. zero by default.

    // 2nd RX window (after up stream)
    u1_t        dn2Dr;
#if !defined(DISABLE_MCMD_RXParamSetupReq)
    u1_t        dn2Ans;       // 0=no answer pend, 0x80+ACKs
#endif
#if !defined(DISABLE_MCMD_DlChannelReq)
    u1_t        macDlChannelAns;        // 0 ==> no answer pending, 0x80+ACK bits
#endif
```

```
#if !defined(DISABLE_MCMD_RXTimingSetupReq)
    bit_t       macRxTimingSetupAns;    // 0 ==> no answer pend, non-zero inserts response.
#endif

    // Class B state
#if !defined(DISABLE_BEACONS)
    u1_t        missedBcns;   // unable to track last N beacons
    u1_t        bcninfoTries; // how often to try (scan mode only)
#endif
    // Public part of MAC state
    u1_t        txCnt;
    u1_t        txrxFlags;  // transaction flags (TX-RX combo)
    u1_t        dataBeg;    // 0 or start of data (dataBeg-1 is port)
    u1_t        dataLen;    // 0 no data or zero length data, >0 byte count of data
    u1_t        frame[MAX_LEN_FRAME];

#if !defined(DISABLE_BEACONS)
    u1_t        bcnChnl;
#endif

    u1_t        noRXIQinversion;
    u1_t        saveIrqFlags;   // last LoRa IRQ flags
};

//! \var struct lmic_t LMIC
//! The state of LMIC MAC layer is encapsulated in this variable.
DECLARE_LMIC; //!< \internal

//! Construct a bit map of allowed datarates from drlo to drhi (both included).
#define DR_RANGE_MAP(drlo,drhi) (((u2_t)0xFFFF<<(drlo)) & ((u2_t)0xFFFF>>(15-(drhi))))
bit_t LMIC_setupBand (u1_t bandidx, s1_t txpow, u2_t txcap);
bit_t LMIC_setupChannel (u1_t channel, u4_t freq, u2_t drmap, s1_t band);
bit_t LMIC_disableChannel (u1_t channel);
bit_t LMIC_enableSubBand(u1_t band);
bit_t LMIC_enableChannel(u1_t channel);
bit_t LMIC_disableSubBand(u1_t band);
bit_t LMIC_selectSubBand(u1_t band);

void  LMIC_setDrTxpow   (dr_t dr, s1_t txpow);  // set default/start DR/txpow
void  LMIC_setAdrMode   (bit_t enabled);        // set ADR mode (if mobile turn off)

#if !defined(DISABLE_JOIN)
bit_t LMIC_startJoining (void);
void  LMIC_tryRejoin    (void);
void  LMIC_unjoin       (void);
void  LMIC_unjoinAndRejoin (void);
#endif

void  LMIC_shutdown     (void);
void  LMIC_init         (void);
void  LMIC_reset        (void);
void  LMIC_clrTxData    (void);
void  LMIC_setTxData    (void);
void  LMIC_setTxData_strict(void);
lmic_tx_error_t LMIC_setTxData2(u1_t port, xref2u1_t data, u1_t dlen, u1_t confirmed);
lmic_tx_error_t LMIC_setTxData2_strict(u1_t port, xref2u1_t data, u1_t dlen, u1_t confirmed);
lmic_tx_error_t LMIC_sendWithCallback(u1_t port, xref2u1_t data, u1_t dlen, u1_t confirmed, lmic_txmessage_cb_t *pCb, void *pUserData);
lmic_tx_error_t LMIC_sendWithCallback_strict(u1_t port, xref2u1_t data, u1_t dlen, u1_t confirmed, lmic_txmessage_cb_t *pCb, void *pUserData);
void  LMIC_sendAlive    (void);

#if !defined(DISABLE_BEACONS)
bit_t LMIC_enableTracking  (u1_t tryBcnInfo);
void  LMIC_disableTracking (void);
#endif

#if !defined(DISABLE_PING)
void  LMIC_stopPingable  (void);
void  LMIC_setPingable   (u1_t intvExp);
#endif

void LMIC_setSession (u4_t netid, devaddr_t devaddr, xref2u1_t nwkKey, xref2u1_t artKey);
void LMIC_setLinkCheckMode (bit_t enabled);
void LMIC_setClockError(u2_t error);
```

```
698 u4_t LMIC_getSeqnoUp    (void);
699 u4_t LMIC_setSeqnoUp    (u4_t);
700 void LMIC_getSessionKeys (u4_t *netid, devaddr_t *devaddr, xref2u1_t nwkKey, xref2u1_t artKey);
701
702 void LMIC_requestNetworkTime(lmic_request_network_time_cb_t *pCallbackfn, void *pUserData);
703 int LMIC_getNetworkTimeReference(lmic_time_reference_t *pReference);
704
705 int LMIC_registerRxMessageCb(lmic_rxmessage_cb_t *pRxMessageCb, void *pUserData);
706 int LMIC_registerEventCb(lmic_event_cb_t *pEventCb, void *pUserData);
707
708 // APIs for client half of compliance.
709 typedef u1_t lmic_compliance_rx_action_t;
710
711 enum lmic_compliance_rx_action_e {
712     LMIC_COMPLIANCE_RX_ACTION_PROCESS = 0,  // process this message normally
713     LMIC_COMPLIANCE_RX_ACTION_START,        // enter compliance mode, discard this message
714     LMIC_COMPLIANCE_RX_ACTION_IGNORE,       // continue in compliance mode, discard this message
715     LMIC_COMPLIANCE_RX_ACTION_END           // exit compliance mode, discard this message
716 };
717
718 lmic_compliance_rx_action_t LMIC_complianceRxMessage(u1_t port, const u1_t *pMessage, size_t nMessage);
719
720 // Declare onEvent() function, to make sure any definition will have the
721 // C conventions, even when in a C++ file.
722 #if LMIC_ENABLE_onEvent
723 DECL_ON_LMIC_EVENT;
724 #endif /* LMIC_ENABLE_onEvent */
725
726 // Special APIs - for development or testing
727 // !!!See implementation for caveats!!!
728
729 #ifdef __cplusplus
730 } // extern "C"
731 #endif
732
733 // names for backward compatibility
734 #include "lmic_compat.h"
735
736 #endif // _lmic_h_
737
```

- Librería lmic_bandplan_eu868.h del LMiC:

```
#ifndef _lmic_eu868_h_
# define _lmic_eu868_h_

#ifndef _lmic_eu_like_h_
# include "lmic_eu_like.h"
#endif

// return maximum frame length (including PHY header) for this data rate (eu868); 0 --> not valid dr.
uint8_t LMICeu868_maxFrameLen(uint8_t dr);
// return maximum frame length (including PHY header) for this data rate; 0 --> not valid dr.
#define LMICbandplan_maxFrameLen(dr) LMICeu868_maxFrameLen(dr)

int8_t LMICeu868_pow2dBm(uint8_t mcmd_ladr_p1);
#define pow2dBm(mcmd_ladr_p1)   LMICeu868_pow2dBm(mcmd_ladr_p1)

// Times for half symbol per DR
// Per DR table to minimize rounding errors
ostime_t LMICeu868_dr2hsym(uint8_t dr);
#define dr2hsym(dr) LMICeu868_dr2hsym(dr)


// TODO(tmm@mcci.com) this looks bogus compared to current 1.02 regional
// spec. https://github.com/mcci-catena/arduino-lmic/issues/18
static inline int
LMICeu868_isValidBeacon1(const uint8_t *d) {
    return d[OFF_BCN_CRC1] != (u1_t)os_crc16(d, OFF_BCN_CRC1);
}

#undef LMICbandplan_isValidBeacon1
#define LMICbandplan_isValidBeacon1(pFrame) LMICeu868_isValidBeacon1(pFrame)

// override default for LMICbandplan_isFSK()
#undef LMICbandplan_isFSK
#define LMICbandplan_isFSK()    (/* RX datarate */LMIC.dndr == EU868_DR_FSK)

#define LMICbandplan_getInitialDrJoin() (EU868_DR_SF7)

void LMICeu868_setBcnRxParams(void);
#define LMICbandplan_setBcnRxParams()   LMICeu868_setBcnRxParams()

u4_t LMICeu868_convFreq(xref2cu1_t ptr);
#define LMICbandplan_convFreq(ptr)      LMICeu868_convFreq(ptr)

void LMICeu868_initJoinLoop(void);
#define LMICbandplan_initJoinLoop()     LMICeu868_initJoinLoop()

ostime_t LMICeu868_nextTx(ostime_t now);
#define LMICbandplan_nextTx(now)        LMICeu868_nextTx(now)

ostime_t LMICeu868_nextJoinState(void);
#define LMICbandplan_nextJoinState()    LMICeu868_nextJoinState()

void LMICeu868_initDefaultChannels(bit_t join);
#define LMICbandplan_initDefaultChannels(join)  LMICeu868_initDefaultChannels(join)

#undef LMICbandplan_nextJoinTime
ostime_t LMICeu868_nextJoinTime(ostime_t now);
#define LMICbandplan_nextJoinTime(now)     LMICeu868_nextJoinTime(now)

void LMICeu868_setRx1Params(void);
#define LMICbandplan_setRx1Params()     LMICeu868_setRx1Params()

#endif // _lmic_eu868_h_
```

Librería lorabase_eu868.h del LMiC:

```
#ifndef _lorabase_eu868_h_
#define _lorabase_eu868_h_

#ifndef _LMIC_CONFIG_PRECONDITIONS_H_
# include "lmic_config_preconditions.h"
#endif

/*******************************************************************************\
|
| Basic definitions for EU868 (always in scope)
|
\*******************************************************************************/

//
// Default frequency plan for EU 868MHz ISM band
// data rates
// this is a little confusing: the integer values of these constants are the
// DataRates from the LoRaWAN Regional Parameter spec. The names are just
// convenient indications, so we can use them in the rare case that we need to
// choose a DataRate by SF and configuration, not by DR code.

enum _dr_eu868_t {
        EU868_DR_SF12 = 0,
        EU868_DR_SF11,
        EU868_DR_SF10,
        EU868_DR_SF9,
        EU868_DR_SF8,
        EU868_DR_SF7,
        EU868_DR_SF7B,
        EU868_DR_FSK,
        EU868_DR_NONE
};

// Bands:
//  g1 :   1%  14dBm
//  g2 : 0.1%  14dBm
//  g3 :  10%  27dBm
//                  freq            band     dataRates
enum {
        EU868_F1 = 868100000,      // g1   SF7-12
        EU868_F2 = 868300000,      // g1   SF7-12 FSK SF7/250
        EU868_F3 = 868500000,      // g1   SF7-12
        EU868_F4 = 868850000,      // g2   SF7-12
        EU868_F5 = 869050000,      // g2   SF7-12
        EU868_F6 = 869525000,      // g3   SF7-12
        EU868_J4 = 864100000,      // g2   SF7-12  used during join
        EU868_J5 = 864300000,      // g2   SF7-12   ditto
        EU868_J6 = 864500000,      // g2   SF7-12   ditto
};
enum {
        EU868_FREQ_MIN = 863000000,
        EU868_FREQ_MAX = 870000000
};
enum {
        EU868_TX_EIRP_MAX_DBM = 16      // 16 dBm EIRP. So subtract 3 dBm for a 3 dBi antenna.
};

enum { EU868_LMIC_REGION_EIRP = 1 };         // region uses EIRP

enum { DR_PAGE_EU868 = 0x10 * (LMIC_REGION_eu868 - 1) };

#endif /* _lorabase_eu868_h_ */
```

- Librería principal de HAL:

```
/*
 * Copyright (c) 2014-2016 IBM Corporation.
 * Copyright (c) 2016, 2018-2019 MCCI Corporation.
 * All rights reserved.
 *
 *  Redistribution and use in source and binary forms, with or without
 *  modification, are permitted provided that the following conditions are met:
 *  * Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *  * Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *  * Neither the name of the <organization> nor the
 *    names of its contributors may be used to endorse or promote products
 *    derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL <COPYRIGHT HOLDER> BE LIABLE FOR ANY
 * DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#ifndef _lmic_hal_h_
#define _lmic_hal_h_

#ifndef _oslmic_types_h_
# include "oslmic_types.h"
#endif

#ifndef _lmic_env_h_
# include "lmic_env.h"
#endif

#ifdef __cplusplus
extern "C"{
#endif

```

```
// The type of an optional user-defined failure handler routine
typedef void LMIC_ABI_STD hal_failure_handler_t(const char* const file, const uint16_t line);

/*
 * initialize hardware (IO, SPI, TIMER, IRQ).
 * This API is deprecated as it uses the const global lmic_pins,
 * which the platform can't control or change.
 */
void hal_init (void);

/*
 * Initialize hardware, passing in platform-specific context
 * The pointer is to a HalPinmap_t.
 */
void hal_init_ex (const void *pContext);

/*
 * drive radio RX/TX pins (0=rx, 1=tx). Actual polarity
 * is determined by the value of HalPinmap_t::rxtx_rx_active.
 */
void hal_pin_rxtx (u1_t val);

/*
 * control radio RST pin (0=low, 1=high, 2=floating)
 */
void hal_pin_rst (u1_t val);

/*
 * Perform SPI write transaction with radio chip
 *    - write the command byte 'cmd'
 *    - write 'len' bytes out of 'buf'
 */
void hal_spi_write(u1_t cmd, const u1_t* buf, size_t len);

/*
 * Perform SPI read transaction with radio chip
 *    - write the command byte 'cmd'
 *    - read 'len' bytes into 'buf'
 */
void hal_spi_read(u1_t cmd, u1_t* buf, size_t len);

```

```
85 /*
86  * disable all CPU interrupts.
87  *   - might be invoked nested
88  *   - will be followed by matching call to hal_enableIRQs()
89  */
90 void hal_disableIRQs (void);
91
92 /*
93  * enable CPU interrupts.
94  */
95 void hal_enableIRQs (void);
96
97 /*
98  * return CPU interrupt nesting count
99  */
100 uint8_t hal_getIrqLevel (void);
101
102 /*
103  * put system and CPU in low-power mode, sleep until interrupt.
104  */
105 void hal_sleep (void);
106
107 /*
108  * return 32-bit system time in ticks.
109  */
110 u4_t hal_ticks (void);
111
112 /*
113  * busy-wait until specified timestamp (in ticks) is reached. If on-time, return 0,
114  * otherwise return the number of ticks we were late.
115  */
116 u4_t hal_waitUntil (u4_t time);
117
118 /*
119  * check and rewind timer for target time.
120  *   - return 1 if target time is close
121  *   - otherwise rewind timer for target time or full period and return 0
122  */
123 u1_t hal_checkTimer (u4_t targettime);
124
```

```
/*
 * perform fatal failure action.
 *   - called by assertions
 *   - action could be HALT or reboot
 */
void hal_failed (const char *file, u2_t line);

/*
 * set a custom hal failure handler routine. The default behaviour, defined in
 * hal_failed(), is to halt by looping infintely.
 */
void hal_set_failure_handler(const hal_failure_handler_t* const);

/*
 * get the calibration value for radio_rssi
 */
s1_t hal_getRssiCal (void);

/*
 * control the radio state
 *   - if val == 0, turn tcxo off and otherwise prepare for sleep
 *   - if val == 1, turn tcxo on and otherwise prep for activity
 *   - return the number of ticks that we need to wait
 */
ostime_t hal_setModuleActive (bit_t val);

/* find out if we're using Tcxo */
bit_t hal_queryUsingTcxo(void);

/* represent the various radio TX power policy */
enum    {
    LMICHAL_radio_tx_power_policy_rfo   = 0,
    LMICHAL_radio_tx_power_policy_paboost   = 1,
    LMICHAL_radio_tx_power_policy_20dBm = 2,
};

/*
 * query the configuration as to the Tx Power Policy
 * to be used on this board, given our desires and
 * requested power.
 */
uint8_t hal_getTxPowerPolicy(
    u1_t inputPolicy,
    s1_t requestedPower,
    u4_t freq
    );

#ifdef __cplusplus
} // extern "C"
#endif

#endif // _lmic_hal_h_
```

- Código .cpp de mapeo del Adafruit Feather M0 LoRa utilizado por HAL:

```
/*

Module:  getconfig_featherm0lora.cpp

Function:
        Arduino-LMIC C++ HAL pinmap for Adafruit Feather M0 LoRa

Copyright & License:
        See accompanying LICENSE file.

Author:
        Terry Moore, MCCI       November 2018

*/

#include <arduino_lmic_hal_boards.h>
#include <Arduino.h>

#include "../lmic/oslmic.h"

namespace Arduino_LMIC {

class HalConfiguration_FeatherM0LoRa_t : public HalConfiguration_t
    {
public:
    enum DIGITAL_PINS : uint8_t
        {
        PIN_SX1276_NSS = 8,
        PIN_SX1276_NRESET = 4,
        PIN_SX1276_DIO0 = 3,
        PIN_SX1276_DIO1 = 6,
        PIN_SX1276_DIO2 = HalPinmap_t::UNUSED_PIN,
        PIN_SX1276_ANT_SWITCH_RX = HalPinmap_t::UNUSED_PIN,
        PIN_SX1276_ANT_SWITCH_TX_BOOST = HalPinmap_t::UNUSED_PIN,
        PIN_SX1276_ANT_SWITCH_TX_RFO = HalPinmap_t::UNUSED_PIN,
        PIN_VDD_BOOST_ENABLE = HalPinmap_t::UNUSED_PIN,
        };


    virtual void begin(void) override
        {
        digitalWrite(PIN_SX1276_NSS, 1);
        pinMode(PIN_SX1276_NSS, OUTPUT);
        }

    // virtual void end(void) override

    // virtual ostime_t setModuleActive(bool state) override

    };

static HalConfiguration_FeatherM0LoRa_t myConfig;

static const HalPinmap_t myPinmap =
        {
        .nss = HalConfiguration_FeatherM0LoRa_t::PIN_SX1276_NSS,      // chip select is D7
        .rxtx = HalConfiguration_FeatherM0LoRa_t::PIN_SX1276_ANT_SWITCH_RX, // RXTX is D29
        .rst = HalConfiguration_FeatherM0LoRa_t::PIN_SX1276_NRESET,   // NRESET is D8

        .dio = {HalConfiguration_FeatherM0LoRa_t::PIN_SX1276_DIO0,    // DIO0 (IRQ) is D25
                HalConfiguration_FeatherM0LoRa_t::PIN_SX1276_DIO1,    // DIO1 is D26
                HalConfiguration_FeatherM0LoRa_t::PIN_SX1276_DIO2,    // DIO2 is D27
               },
        .rxtx_rx_active = 0,
        .rssi_cal = 10,
        .spi_freq = 8000000,     /* 8MHz */
        .pConfig = &myConfig
        };

const HalPinmap_t *GetPinmap_FeatherM0LoRa(void)
    {
    return &myPinmap;
    }

}; // namespace Arduino_LMIC

```

- Credenciales para registro de dispositivo my_credentials.h:

```
//credenciales para dispositivo WIFI
#define WSSID "REPLACE WITH YOUR WIFI SSID"
#define WPASS ""

//credenciales para OTAA
#define myAPPEUI    {0x99, 0x83, 0x01, 0xD0, 0x7E, 0xD5, 0xB3, 0x70}
#define myDEVEUI    {0x87, 0x47, 0x7C, 0xE9, 0xD1, 0x3D, 0xFF, 0x00}
#define myAPPKEY    {0x15, 0xDC, 0xEB, 0x00, 0x50, 0x42, 0x03, 0x77, 0x0B, 0x5E, 0xB4, 0xE9, 0x7D, 0x01, 0xCF, 0xC7}

//credenciales para otros procesos LoRaWAN
#define NETSKEYDRAG "REPLACE WITH YOUR NETWORK SESSION KEY"
#define APPSKEYDRAG "REPLACE WITH YOUR APPLICATION SESSION KEY"
#define DEVADDRDRAG "REPLACE WITH YOUR DEVICE ADRESS"

#define DEVICE_ID "REPLACE WITH YOUR DEVICE ID"
#define ACCESS_KEY "REPLACE WITH YOUR ACCESS KEY"
#define ACCESS_SECRET "REPLACE WITH YOUR ACCES SECRET"

```

- Librería ArduinoLowPower.h usada para *sleep mode* del Adafruit:

```
#ifndef _ARDUINO_LOW_POWER_H_
#define _ARDUINO_LOW_POWER_H_

#include <Arduino.h>

#ifdef ARDUINO_ARCH_AVR
#error The library is not compatible with AVR boards
#endif

#ifdef ARDUINO_ARCH_SAMD
#include "RTCZero.h"
#endif

#if defined(ARDUINO_SAMD_TIAN) || defined(ARDUINO_NRF52_PRIMO)
// add here any board with companion chip which can be woken up
#define BOARD_HAS_COMPANION_CHIP
#endif

#define RTC_ALARM_WAKEUP    0xFF

//typedef void (*voidFuncPtr)( void ) ;
typedef void (*onOffFuncPtr)( bool ) ;

typedef enum{
    OTHER_WAKEUP = 0,
    GPIO_WAKEUP = 1,
    NFC_WAKEUP = 2,
    ANALOG_COMPARATOR_WAKEUP = 3
} wakeup_reason;


class ArduinoLowPowerClass {
    public:
        void idle(void);
        void idle(uint32_t millis);
        void idle(int millis) {
            idle((uint32_t)millis);
        }

```

```
40          void sleep(void);
41          void sleep(uint32_t millis);
42          void sleep(int millis) {
43              sleep((uint32_t)millis);
44          }
45
46          void deepSleep(void);
47          void deepSleep(uint32_t millis);
48          void deepSleep(int millis) {
49              deepSleep((uint32_t)millis);
50          }
51
52          void attachInterruptWakeup(uint32_t pin, voidFuncPtr callback, uint32_t mode);
53
54          #ifdef BOARD_HAS_COMPANION_CHIP
55          void companionLowPowerCallback(onOffFuncPtr callback) {
56              companionSleepCB = callback;
57          }
58          void companionSleep() {
59              companionSleepCB(true);
60          }
61          void companionWakeup() {
62              companionSleepCB(false);
63          }
64          #endif
65
66          #ifdef ARDUINO_ARCH_NRF52
67          void enableWakeupFrom(wakeup_reason peripheral, uint32_t pin = 0xFF, uint32_t event = 0xFF, uint32_t option = 0xFF);
68          wakeup_reason wakeupReason();
69          #endif
70
71      private:
72          void setAlarmIn(uint32_t millis);
73          #ifdef ARDUINO_ARCH_SAMD
74          RTCZero rtc;
75          #endif
76          #ifdef BOARD_HAS_COMPANION_CHIP
77          void (*companionSleepCB)(bool);
78          #endif
79  };
80
81  extern ArduinoLowPowerClass LowPower;
82
83  #endif
```