

**Máster Universitario en Ingeniería de  
Telecomunicación**



**Trabajo Fin de Máster**

Desarrollo de un sistema remoto de adquisición de datos basado  
en LoRaWAN para aplicaciones IoT



ESCUELA POLITECNICA  
SUPERIOR

**Autor:** Sergio Lluva Plaza

**Tutor:** José Manuel Villadangos Carrizo



UNIVERSIDAD DE ALCALÁ  
Escuela Politécnica Superior

**Máster Universitario en Ingeniería de Telecomunicación**

Trabajo Fin de Máster  
Desarrollo de un sistema remoto de adquisición de datos basado en  
LoRaWAN para aplicaciones IoT

**Autor:** Sergio Lluva Plaza

**Tutor:** José Manuel Villadangos Carrizo

**TRIBUNAL:**

**Presidente:** José Manuel Rodríguez Ascariz

**Vocal 1º:** Bernardo Alarcos Alcázar

**Vocal 2º:** José Manuel Villadangos Carrizo

**FECHA:** 24 de septiembre de 2021



## Resumen

Este Trabajo de Fin de Máster tiene como objetivo el desarrollo e implementación de un sistema capaz de obtener datos de parámetros ambientales mediante un dispositivo diseñado ad-hoc basado en un microcontrolador Cortex-M3 de ST Microelectronics. El dispositivo transmite la información obtenida de los sensores a través de comunicación inalámbrica usando la tecnología LoRa y aplicando el protocolo de comunicaciones LoRaWAN que permite el envío de la información a la nube, siendo una aplicación orientada al Internet de las Cosas. La información se recibe en un servidor instalado en Amazon Web Services para ser almacenada en una base de datos, que posteriormente se recupera para ser representada gráficamente en una página web utilizando el framework Flask.

**Palabras clave:** LoRa, LoRaWAN, Internet of Things (IoT), microcontrolador, sensores



## **Abstract**

The objective of this Master Thesis is the development and implementation of a system capable of obtaining data of environmental parameters through an ad-hoc designed device based on a Cortex-M3 microcontroller from ST Microelectronics. The device transmits the information obtained from the sensors through wireless communication using LoRa technology and implementing the LoRaWAN communications protocol that allows sending the information to the cloud, being an application oriented to the Internet of Things. The information is received in a server installed in Amazon Web Services to be stored in a database, which is later retrieved to be graphically represented in a web page using the Flask framework.

**Key Words:** LoRa, LoRaWAN, Internet of Things (IoT), microcontroller, sensors





# Índice general

Resumen.....	1
Abstract .....	3
Índice general.....	5
Índice de figuras.....	9
Índice de tablas.....	13
Lista de acrónimos .....	15
Resumen extendido.....	17
1.Introducción .....	19
1.1. Motivación .....	19
1.2. Objetivos .....	19
1.3. Estructura de la memoria .....	20
2. Estado del arte.....	21
2.1. Comparación de redes LPWAN.....	21
2.1.2. DASH7.....	23
2.1.3. NB-IoT .....	24
2.1.4. LoRa.....	24
2.2. Nodos LoRaWAN.....	24
2.3. Gateways LoRaWAN .....	26
2.4. Servidores de red LoRaWAN .....	29
3. Tecnología LoRa.....	33
3.1. Introducción .....	33
3.2. Modulación .....	34
3.3. Frecuencias utilizadas .....	35
3.4. Factor de dispersión .....	36

3.5. Ancho de banda.....	36
3.6. Tasa de codificación.....	36
3.7. SNR.....	36
3.8. Aplicaciones.....	37
4. Protocolo LoRaWAN.....	39
4.1. Introducción .....	39
4.2. Clases .....	40
4.3. Estructura de las tramas LoRaWAN .....	42
4.4. Seguridad .....	42
4.5. Modos de activación .....	43
5. Diseño del hardware.....	45
5.1. Microcontrolador .....	45
5.2. Sensores del nodo.....	46
5.2.1. Sensor de radiación UV .....	47
5.2.2. Sensor BME680.....	48
5.2.3. Sensor de luz VEML7700.....	50
5.2.4. Sensor de calidad del aire.....	51
5.2.5. Receptor GPS.....	52
5.2.6. Sensor de lluvia.....	53
5.3. Sistema de alimentación.....	53
5.4. Comunicación LoRa .....	54
5.5. Integración de los componentes del nodo .....	54
5.6. Esquema y diseño de la PCB .....	55
6. Diseño del software.....	59
6.1. Software en el microcontrolador.....	59
6.2. Software de configuración del gateway .....	66
6.3. Software en AWS.....	68
6.3.1. ChirpStack.....	69
6.3.2. Base de datos InfluxDB .....	76
6.3.3. Framework Flask.....	77
7. Conclusiones y trabajo futuro .....	85
7.1. Conclusiones .....	85
7.2. Trabajo futuro .....	86
Bibliografía .....	87
Anexos .....	91

Anexo A. Presupuesto.....	91
A.1. Costes de equipo y software.....	91
A.2. Costes de material .....	91
A.3. Costes de personal.....	92
A.4. Presupuesto total .....	92
Anexo B: Pliego de condiciones .....	93
Anexo C. Código de programación del microcontrolador.....	94
Anexo C1. Código del fichero principal (main.c).....	94
Anexo C2. Código de configuración del bus I2C (i2c.c).....	101
Anexo C3. Código de configuración de la interfaz UART (uart.c) .....	102
Anexo C4. Código del sensor BME680 (BME680.c).....	103
Anexo C5. Código del sensor VEML6075 (VEML6075.c) .....	105
Anexo C6. Código del sensor CCS811 (CCS811.c).....	108
Anexo C7. Código del sensor VEML7700 (VEML7700.c) .....	111
Anexo C8. Código del receptor GPS (GPS.c) .....	112
Anexo D. Código HTML de la página web (index.html) .....	114
Anexo E. Código de la aplicación Flask (app.py).....	117
Anexo F. Captura de esquema para el diseño de la PCB .....	123



# Índice de figuras

Fig. 1. Esquema del sistema a implementar.....	18
Fig. 2. Mapa mundial de cobertura Sigfox.....	22
Fig. 3. Mapa de cobertura Sigfox en Europa.....	22
Fig. 4. Arquitectura de una red DASH7 [2].....	23
Fig. 5. Modos de funcionamiento NB-IoT [3].....	24
Fig. 6. Arduino Mega y módulo LoRa.....	25
Fig. 7. Módulo Heltec ESP32 LoRa.....	25
Fig. 8. Dragino LBT1.....	26
Fig. 9. Tarjeta de evaluación de ST junto con tarjeta de evaluación de LoRa.....	26
Fig. 10. Gateway Wirnet Station.....	27
Fig. 11. Gateway Multitech Multiconnect.....	27
Fig. 12. The Things Indoor Gateway.....	28
Fig. 13. Gateway Lorix One.....	28
Fig. 14. Tarjeta TTGO LoRa32 V1 868 MHz.....	29
Fig. 15. Rapberry Pi con concentrador RAK833 [16].....	29
Fig. 16. Arquitectura de red usando TTN.....	30
Fig. 17. Mapa de TTN con los gateways disponibles.....	30
Fig. 18. Arquitectura de ChirpStack.....	31
Fig. 19. Arquitectura de red con Lorient.....	31
Fig. 20. Comparación de LoRa con otras tecnologías[24].....	33
Fig. 21. Chirp ascendente [20].....	34
Fig. 22. Banda ancha vs banda estrecha [20].....	34
Fig. 23. Modulación LoRa [25].....	35
Fig. 24. Link Budget LoRa.....	37
Fig. 25. Estructura de capas del protocolo LoRaWAN[20].....	40
Fig. 26. Arquitectura de una red LoRaWAN[21].....	40
Fig. 27. Ventanas de recepción en clase A [22].....	41
Fig. 28. Ventanas de recepción en clase B [22].....	41
Fig. 29. Ventanas de recepción en clase C [22].....	41
Fig. 30. Estructura de mensajes LoRaWAN [23].....	42
Fig. 31. Elementos implicados en el proceso de unión [20].....	44

Fig. 32. Diagrama de bloques del sistema Hardware.....	45
Fig. 33. Diagrama de bloques del nodo LoRa.....	46
Fig. 34. Sensor VEML6075.....	47
Fig. 35. Respuesta espectral sensor VEML6075.....	48
Fig. 36. Sensor BME680.....	48
Fig. 37. Air Quality Index.....	49
Fig. 38. Sensor VEML7700.....	50
Fig. 39. Respuesta del sensor VEML7700 vs ojo humano.....	50
Fig. 40. Sensor CCS811.....	51
Fig. 41. Receptor GPS Quectel L80-M39.....	52
Fig. 42. Sensor de lluvia.....	53
Fig. 43. Diagrama de bloques comunicación LoRa.....	54
Fig. 44. Diagrama de conexionado del prototipo.....	55
Fig. 45. Prototipo del nodo LoRa.....	55
Fig. 46. Interconexión del sistema de alimentación y carga de la batería.....	56
Fig. 47. Diagrama de interconexión del sistema de radiofrecuencia LoRa.....	56
Fig. 48. Diagrama de la sección del microcontrolador.....	57
Fig. 49. Diagrama con los sensores VEML6075 y VEML7700.....	57
Fig. 50. Diagrama con los sensores BME680 y CCS811.....	57
Fig. 51. Aspecto PCB, cara superior (izquierda) y cara inferior (derecha).....	58
Fig. 52. Comunicación entre capas LoRaMAC [37].....	59
Fig. 53. Diagrama de bloques de la librería I-CUBE-LRWAN.....	60
Fig. 54. Estructura del proyecto en Keil uVision5.....	60
Fig. 55. Código para dar formato al valor de la temperatura.....	65
Fig. 56. Código para dar formato al valor de la latitud.....	65
Fig. 57. Código para dar formato al valor del sensor de lluvia.....	65
Fig. 58. Máquina de estados.....	66
Fig. 59. Estructura gateway [38].....	67
Fig. 60. Características del sistema operativo instalado en AWS.....	68
Fig. 61. Tipos de configuraciones para la máquina virtual.....	68
Fig. 62. Arquitectura de ChirpStack.....	69
Fig. 63. Diagrama del proceso de traducción de HTTP a gRPC [40].....	70
Fig. 64. Integraciones disponibles en ChirpStack.....	70
Fig. 65. Organización en ChirpStack.....	71
Fig. 66. Crear servidor de red en Chirpstack.....	71
Fig. 67. Crear perfil de gateway en ChirpStack.....	71
Fig. 68. Crear perfil de servicio en ChirpStack.....	72
Fig. 69. Crear gateway en ChirpStack.....	73
Fig. 70. Perfil de dispositivo en ChirpStack.....	74
Fig. 71. Crear una aplicación en ChirpStack.....	74
Fig. 72. Configuración de la integración con InfluxDB.....	75
Fig. 73. Crear dispositivo en ChirpStack.....	75
Fig. 74. Configuración de la clave de aplicación.....	76
Fig. 75. Función para decodificar los datos que llegan a ChirpStack.....	76
Fig. 76. Estructuras dentro de la base de datos “chirpstack”.....	77
Fig. 77. Estructura dl proyecto Flask.....	78
Fig. 78. Módulos de los paquetes Python utilizado.....	78

Fig. 79. Código de acceso a InfluxDB. ....	79
Fig. 80. Creación del gráfico del valor de la temperatura con HighCharts.....	79
Fig. 81. Extracción y renderizado del gráfico de la temperatura. ....	80
Fig. 82. Código HTML para insertar el gráfico de la temperatura. ....	80
Fig. 83. Código para crear el mapa. ....	81
Fig. 84. Extracción y renderizado del mapa.....	81
Fig. 85. Código HTML para insertar el mapa.....	81
Fig. 86. Encabezado de la página web. ....	82
Fig. 87. Mapa con la posición del nodo en la página web. ....	82
Fig. 88. Representación gráfica de los datos en la página web.....	83
Fig. 89. Pie de página de la página web.....	83





## Índice de tablas

Tabla I. Frecuencias de uplink en la banda de 868MHz.....	35
Tabla II. Costes de equipo y software.....	91
Tabla III. Costes de material.....	92
Tabla IV. Costes de personal.....	92
Tabla V. Presupuesto total.....	92



## Lista de acrónimos

IoT (Internet of Things)  
LoRa (Long Range Modulation)  
WAN (Wide Area Network)  
LPWAN (Low Power Wide Area Network)  
WPAN (Wireless Personal Area Network)  
WLAN (Wireless Local Area Network)  
WMAN (Wireless Metropolitan Area Network)  
WWAN (Wireless Wide Area Network)  
PCB (Printed Circuit Board)  
WiFi (Wireless Fidelity)  
DIY (Do It Yourself)  
IDE (Integrated Development Environment)  
API (Application Programming Interfaces)  
OSI (Open Systems Interconnection)  
LiPo (Lithium-ion Polymer battery)  
GPS (Global Positioning System)  
ISM (Industrial, Scientific and Medical)  
ABP (Activation By Personalization)  
OTAA (Over The Air Activation)  
AES (Advanced Encryption Standard)  
NwkSKey (Network Session Key)

AppSKey (Application Session Key)

UNB (Ultra Narrow Band)

I2C (Inter-Integrated Circuit)

SPI (Serial Peripheral Interface)

UART (Universal Asynchronous Receiver-Transmitter)

ADC (Analog to Digital Converter)

GPIO (General Purpose Input/Output)

GFSK (Gaussian Frequency Shift Keying)

BPSK (Binary Phase-Shift Keying)

BSP (Board Support Package)

HAL (Hardware Abstraction Layer)

AES (Advanced Encryption Standard)

## Resumen extendido

En la actualidad, se está viviendo un aumento considerable de dispositivos que se conectan a Internet para muy variados propósitos. Además, las predicciones en cuanto al número de dispositivos conectados a Internet en el año 2030 son de 500 billones [1], por tanto, es un mercado a tener en cuenta y que va a tener una gran repercusión en los siguientes años.

Debido al gran auge que está teniendo el Internet de las Cosas durante los últimos años, han surgido tecnologías de comunicaciones para hacer posible crear un ecosistema de dispositivos conectados a Internet que sean capaces de enviar y recibir datos de todo tipo de aplicaciones. De esta acción de conectar dispositivos a Internet surgió el concepto de Internet de las Cosas. El IoT se basa en la conexión de todo tipo de dispositivos a Internet e incluso de una comunicación entre dispositivos para facilitar la vida a las personas. Por ejemplo, un sistema que avise al usuario de que se ha dejado la luz del salón encendida al salir de casa, para proceder a apagarla desde un smartphone. O tener monitorizada el agua o la comida de una mascota para que al acabarse se avise a la persona encargada para reponer existencias. Es un campo en el que hay una infinidad de aplicaciones.

En este trabajo se realiza un estudio teórico de las tecnologías que se van a utilizar y de las posibles tecnologías que compiten en la actualidad. Una vez comprendido el protocolo y la tecnología de transmisión de datos se desarrolla una aplicación práctica que consiste en un sistema capaz de medir parámetros ambientales que pueden utilizarse para analizar la contaminación ambiental de la zona donde se instale el dispositivo.

Con este trabajo se pretende explicar el funcionamiento de la tecnología de capa física LoRa y del protocolo LoRaWAN. Una vez que se comprenden cómo funcionan las tecnologías a utilizar se implementará el desarrollo de un dispositivo basado en la adquisición de datos ambientales, que fácilmente puede ser extendido a otras áreas de interés.

Utilizando el protocolo LoRaWAN se crea la red de sensores que envía datos mediante LoRa, siendo sus características más destacables el gran alcance de transmisión y el bajo consumo de energía. Por tanto, con un solo gateway se puede dar cobertura a una amplia zona y además los dispositivos pueden ser alimentados durante largos periodos de tiempo con baterías. Por el contrario, la tasa de datos que permite es muy baja, siendo ideal para aplicaciones que envían datos puntualmente. Por ejemplo, un sensor instalado en contenedores de basura subterráneos que solo envía información cuando el contenedor supera un cierto porcentaje de su capacidad.

Una red LoRaWAN tiene formato tipo estrella y está compuesta por nodos, que son los dispositivos capaces de enviar y recibir información a un gateway que es el encargado de canalizar la información

al servidor de red para su posterior envío al servidor de aplicación donde se puede hacer uso de ellos. Toda transmisión iniciada por un nodo debe ser recibida por un gateway, es decir, no puede haber comunicación directa entre dos nodos. Usando la tecnología de capa física LoRa si puede realizarse esta comunicación directa entre dispositivos, que puede ser utilizada para el envío de datos punto a punto, aplicaciones de telemetría, etc.

Debido al gran auge de LoRaWAN en los últimos años, han surgido proyectos como The Things Network para ofrecer a los usuarios una infraestructura completa y abierta para el uso de esta tecnología. En concreto en The Things Network se integra un servidor de red en el que usuarios de todo el mundo pueden registrar sus gateways para crear una red pública y que todo el mundo tenga acceso. También se pueden crear redes privadas, siendo el principal inconveniente que se deben instalar los gateways que son costosos y gestionar un servidor de red privado como puede ser ChirpStack.

En la Fig. 1 se presenta en forma de diagrama de bloques la solución tecnológica que se ha desarrollado en este Trabajo Fin de Máster. En primer lugar, se desarrolla un prototipo basado en tarjetas de evaluación, en las que está incluido el microcontrolador de ST, el módulo de comunicaciones LoRa y los sensores de parámetros ambientales. Tras comprobar el correcto funcionamiento del prototipo se desarrolla el diseño del dispositivo de adquisición de datos a bajo nivel mediante el diseño del esquema y su implementación en una PCB basada en los componentes utilizados en el prototipo. Los datos de interés, se transmite mediante un sistema de radiofrecuencia, también incluido en la PCB, que utiliza la tecnología LoRa como capa física y hace uso de LoRaWAN como protocolo de acceso al medio. El nodo es autónomo ya que está alimentado con una batería que se recarga a través de un panel solar. Los datos enviados por el nodo los recibe un gateway que es el encargado de transmitirlos hacia Internet, donde se ha instalado una máquina virtual en Amazon Web Services para recibir los datos con el servidor de red ChirpStack. La información se almacena en una base de datos, en concreto, en InfluxDB para posteriormente ser recuperados por una aplicación web desarrollada bajo el framework Flask y así ser representados gráficamente en una página web.

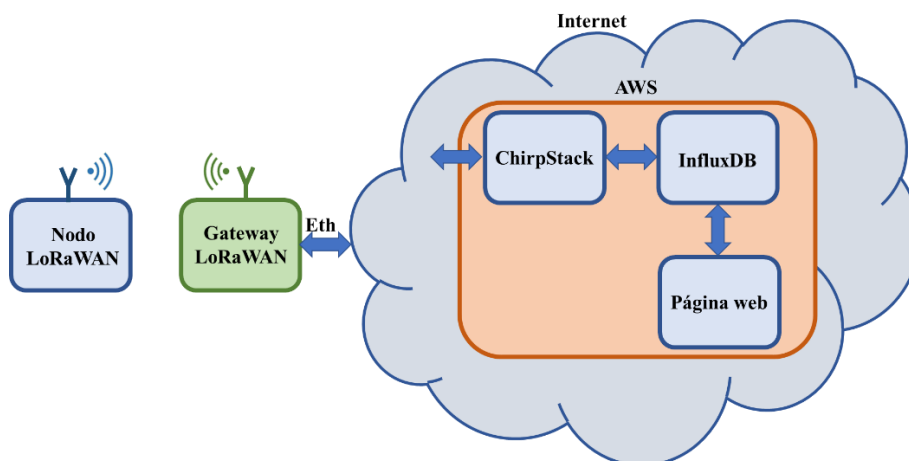


Fig. 1. Esquema del sistema a implementar.

# 1.Introducción

## 1.1. Motivación

En la actualidad el Internet de las Cosas (IoT) se encuentra en pleno auge por lo que surgen diversas tecnologías, protocolos de comunicaciones (LoRaWAN, Sigfox, etc) y aplicaciones para satisfacer o facilitar la vida a las personas. Con este TFM se pretende realizar un estudio de la tecnología de capa física LoRa y del protocolo LoRaWAN para la transmisión de datos a largas distancias y con un bajo consumo de energía. Estas características son esenciales para sistemas que requieran ser alimentados con baterías para que su duración sea lo más elevada posible y se puedan emplazar en lugares remotos.

Se desarrollará un sistema hardware y una aplicación software, en concreto de sensado de parámetros ambientales, los cuales serán transmitidos desde un lugar remoto hacia un servidor de Internet para su posterior visualización por los usuarios. Con anterioridad a este tipo de tecnologías, los datos se transmitían mediante tecnologías de acceso a redes móviles como GPRS, 2G, 3G, etc. las cuales tienen el principal inconveniente de que consumen bastante energía y requieren de una tarjeta SIM y una tarifa de pago para la transmisión de los datos.

## 1.2. Objetivos

El objetivo principal de este Trabajo Fin de Máster es el desarrollo de un sistema de adquisición de parámetros ambientales. En concreto se parte del bajo nivel con el desarrollo de una PCB basada en un microcontrolador, sensores y módulo de radiofrecuencia para el envío/recepción de la información. Después se transmite la información de interés mediante LoRa utilizando el protocolo de comunicaciones LoRaWAN, que previamente han sido estudiados, para recibirla en una base de datos y así poder extraer los datos desde una aplicación para ser visualizados en una página web, que sería el alto nivel de la aplicación. Por tanto, los objetivos parciales del proyecto son:

- Estudio de la tecnología de capa física LoRa.
- Estudio del protocolo LoRaWAN.
- Comparación con otras tecnologías LPWAN como Sigfox, DASH, etc.
- Descripción de los dispositivos comerciales que hay en la actualidad, principales nodos y gateways.

- Elección de sensores ambientales, módulo de comunicación LoRa y subsistema de alimentación.
- Desarrollo del código necesario para realizar la lectura de los sensores y el envío de los a través de la red LoRaWAN.
- Diseño de una tarjeta PCB que incluya un microcontrolador STM32 y los componentes elegidos.
- Desarrollo de un entorno web para la visualización de los datos adquiridos.

### 1.3. Estructura de la memoria

La memoria de este Trabajo Fin de Máster está organizada en seis secciones:

- **Estado del arte:** Se hace una introducción a las tecnologías de comunicación disponibles para el Internet de las Cosas y se muestran los principales componentes de redes LoRaWAN que hay en la actualidad.
- **Tecnología LoRa:** Descripción de la tecnología de comunicaciones de radiofrecuencia LoRa, los parámetros asociados a la comunicación y las principales aplicaciones de esta tecnología de capa física.
- **Protocolo LoRaWAN:** Se describe las características del protocolo de comunicaciones LoRaWAN, en concreto, los tres tipos de clases de dispositivos, el formato de las tramas, la seguridad que implementa el protocolo y los tipos de activación de los dispositivos.
- **Diseño del hardware:** Descripción del diseño del Hardware. Se describen las principales características de los sensores utilizados y módulos necesarios para implementar un prototipo y verificar su correcto funcionamiento. Después se diseña una PCB basada en el prototipo para integrar todos los componentes en una única placa.
- **Diseño del software:** En esta sección se describe tanto la programación necesaria para que el sistema basado en microcontrolador envíe datos al servidor, como la parte de almacenamiento, procesado y visualización de datos que se lleva a cabo en la máquina virtual instalada en Amazon Web Services.
- **Conclusiones y trabajo futuro:** Se muestran las conclusiones obtenidas tras la realización del Trabajo Fin de Máster y se analizan posibles campos de actuación en los que se puede añadir nueva funcionalidad al trabajo desarrollado.



## 2. Estado del arte

Las tecnologías inalámbricas de comunicaciones que se pueden utilizar para la transmisión de información en el campo del Internet de las Cosas son muy variadas. Estas tecnologías se pueden clasificar en varios tipos según su alcance:

- WPAN (Wireless Personal Area Network): Utilizadas para intercambiar información entre dispositivos localizados en un área de reducido tamaño, alrededor de 10 metros. Dentro de estas redes destacan las tecnologías de Bluetooth, ZigBee o RFID (Radio Frequency Identification).
- WLAN (Wireless Local Area Network): Se implementan principalmente en edificios para evitar que los dispositivos estén cableados y tienen un alcance de unos 100 metros. La principal tecnología usada en estas redes es WiFi.
- WMAN (Wireless Metropolitan Area Network): Tecnología inalámbrica utilizada para conectar ubicaciones que se encuentran en distintos puntos de un área metropolitana. En estas redes destaca la tecnología WiMAX (Worldwide Interoperability for Microwave Access).
- WWAN (Wireless Wide Area Network): Estas redes se utilizan principalmente para dar cobertura a una amplia zona para proveer un servicio específico. Destacan las redes de telefonía móvil y las comunicaciones inalámbricas utilizando satélites geoestacionarios o de baja órbita.

Dentro de las redes WWAN se encuentran las redes LPWAN (redes de banda ancha y baja potencia) que son un tipo de redes inalámbricas utilizadas para transmisiones de datos a grandes distancias, con un consumo bajo de energía y una velocidad de datos muy baja.

Las redes LoRaWAN se enmarcan dentro de las redes LPWAN y están formadas principalmente por tres dispositivos: nodos, gateways y servidores de red. En esta sección también se hace una pequeña introducción a estos dispositivos mostrando los que se utilizan en la actualidad junto con sus características.

### 2.1. Comparación de redes LPWAN

Los dispositivos del Internet de las Cosas para enviar los datos a un servidor o para comunicarse entre ellos utilizan tecnologías de comunicación principalmente basadas en radiofrecuencia. En las

siguientes secciones, se realiza una revisión de las principales tecnologías que se utilizan en la actualidad.

### 2.1.1. Sigfox

Es una red de comunicaciones independiente pensada principalmente para el Internet de las Cosas. Esta tecnología está patentada bajo una empresa francesa con el mismo nombre (Sigfox) que también es un proveedor de red, e instala sus antenas y dispositivos necesarios en estaciones base de su propiedad o en emplazamientos de otras compañías para aprovecharse de todo el núcleo de la red y no tener que crear una infraestructura.

Sigfox opera en 65 países, en los que tiene una superficie de cobertura total de 5 millones de kilómetros cuadrados (datos consultados en octubre de 2019). En la siguiente imagen se puede observar en color azul el territorio que tiene cobertura y en color morado el territorio bajo despliegue.

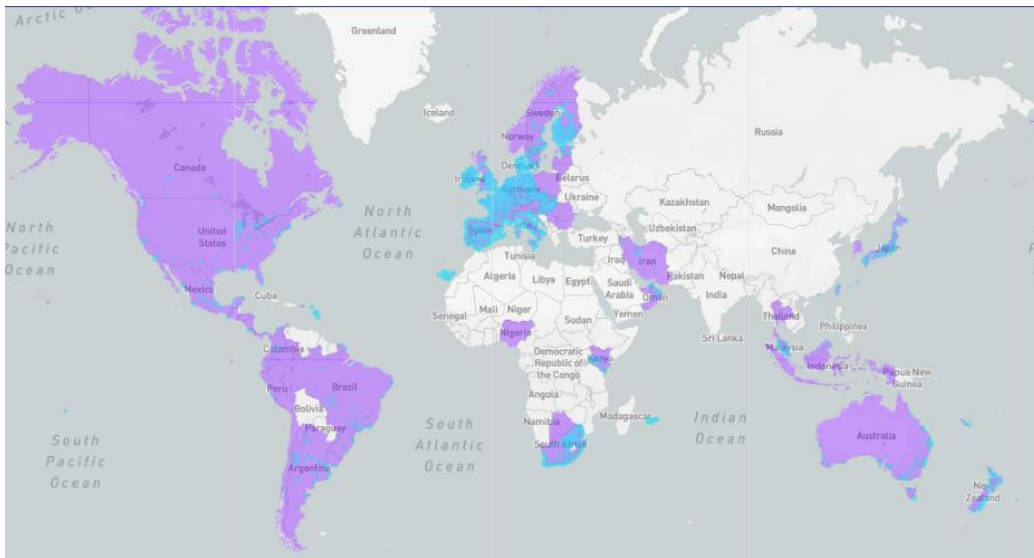


Fig. 2. Mapa mundial de cobertura Sigfox.

La mayor parte de zonas con cobertura están en Europa, siendo los países con mayor territorio con cobertura Sigfox, España, Francia, Alemania, Italia y Reino Unido.

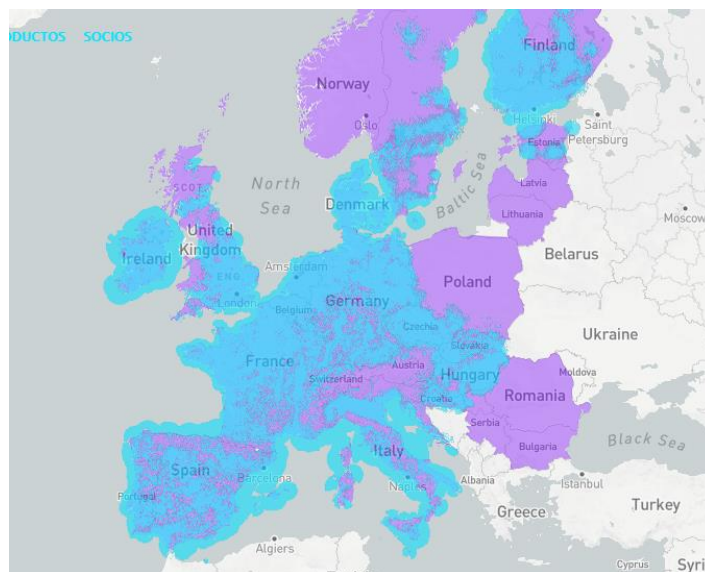


Fig. 3. Mapa de cobertura Sigfox en Europa.

Utiliza una modulación BPSK (Binary Phase-Shift Keying) en una banda ultra estrecha UNB (Ultra Narrow Band) de 100 Hz en las bandas ISM sin licencia por debajo de los GHz. Al utilizar UNB el ancho de banda se usa de forma eficiente y el nivel de ruido que se introduce en el espectro que utiliza es muy pequeño, consiguiendo un bajo consumo y una alta sensibilidad en el receptor, sin embargo, la velocidad que se puede obtener es de hasta 100 bps.

Inicialmente Sigfox solo tenía la posibilidad de enviar datos en el enlace ascendente, de los nodos hacia la estación base, pero posteriormente se implementó la capacidad de enviar datos en el enlace descendente, pero con restricciones. Solo se puede enviar un mensaje de downlink después de recibir un mensaje de uplink. La cantidad de mensajes que pueden enviar los dispositivos que forman esta red está limitada, en concreto se pueden enviar 140 mensajes de uplink al día de 12 bytes cada uno. El número de mensajes de downlink también está limitado a 4 mensajes por día de 8 bytes cada uno.

Esta tecnología no soporta mensajes de confirmación (ACK) por lo que para asegurarse que un mensaje llega correctamente a una estación base, se transmite cada mensaje varias veces, por defecto tres, sobre canales de frecuencia diferentes elegidos aleatoriamente.

### 2.1.2. DASH7

DASH7 es un protocolo de red inalámbrica de código abierto, que utiliza las bandas ISM disponibles en cada país para establecer comunicaciones. Implementa mecanismos de seguridad, se puede obtener un rango de cobertura de hasta 5km, una baja latencia de conexión y se pueden transmitir datos a una tasa de hasta 166.667kbit/s.

Este protocolo está gestionado por la organización DASH7 Alliance, que es una organización sin ánimo de lucro. El ancho de canal que utiliza puede ser de 25kHz o 200kHz y permite las tres topologías de conexión: árbol, estrella y nodo a nodo. Cada paquete que se transmite puede tener un tamaño de 256 bytes como máximo. En la capa física utiliza GFSK (Gaussian Frequency Shift Keying) como esquema de modulación. La arquitectura de una red basada en DASH7 se muestra en la Fig. 4, donde se pueden distinguir los nodos, que envían información, el gateway que hace de intermediario reenviando los datos hacia el servidor de red y el servidor de red, que recibe la información en la nube y provee los datos a una aplicación para que el usuario pueda visualizar los datos de interés.

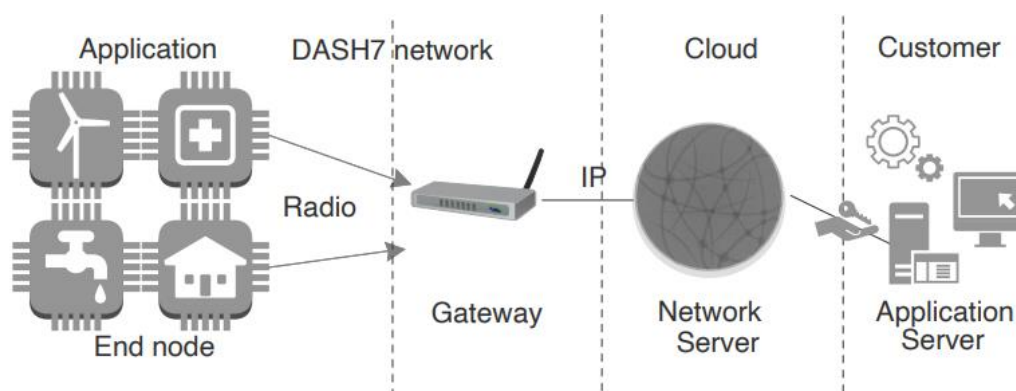


Fig. 4. Arquitectura de una red DASH7 [2].

### 2.1.3. NB-IoT

Está basada en la tecnología radio de banda estrecha. Está estandarizada por la 3GPP (3rd Generation Partnership Project). La primera especificación de esta tecnología fue publicada en junio de 2016 por la 3GPP. Utiliza las bandas que requieren de licencia y coexiste con las bandas de frecuencia que utiliza LTE y GSM (700MHz, 800MHz y 900MHz) ocupando un ancho de banda de 200kHz. Utiliza un protocolo que está basado en el protocolo de LTE, del que se reducen características para que pueda ser utilizado por dispositivos del Internet de las Cosas en los que la capacidad de procesamiento es reducida. En el enlace ascendente utiliza la modulación QPSK y FDMA, mientras que en el enlace descendente utiliza OFDMA. Los mensajes enviados pueden tener una carga útil máxima de 1600 bytes. En el enlace ascendente se pueden transmitir datos a una tasa de 20kbps y en el enlace descendente a 200kbps. NB-IoT dispone de tres modos de funcionamiento, véase Fig. 5; modo “Stand-alone”, donde se utiliza el espectro de las bandas de portadora de GSM. Modo “Guard-band” donde se utilizan las bandas de guarda que no utiliza LTE para la transmisión de datos. Modo “In-band”, donde se utiliza el espectro de las bandas de portadora de LTE.

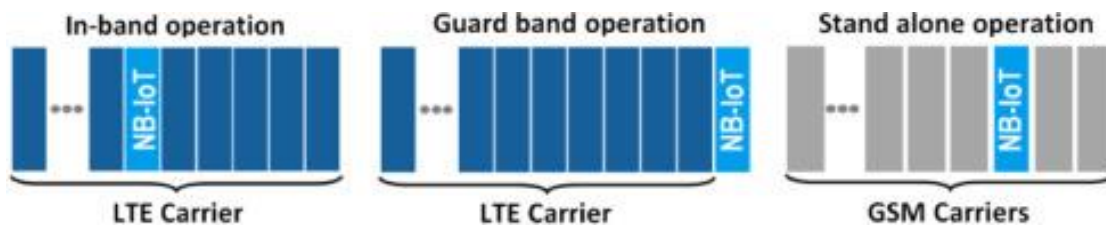


Fig. 5. Modos de funcionamiento NB-IoT [3].

### 2.1.4. LoRa

LoRa es una tecnología de capa física para establecer comunicaciones con dispositivos IoT. Utiliza las bandas ISM disponibles en cada región. Ofrece una comunicación bidireccional basada en una modulación de espectro ancho (CSS) con la que se puede recibir la información por debajo del nivel de ruido. LoRa hace uso de un parámetro llamado factor de dispersión para ajustar el alcance de la transmisión y la tasa de datos. La tasa de transmisión de datos puede estar comprendida entre 300bps y 50kbps, pudiendo transmitir como máximo 243 bytes de carga útil.

A partir de la tecnología LoRa surge un protocolo llamado LoRaWAN para crear redes de sensores. Este protocolo permite gestionar comunicaciones aportando seguridad en el intercambio de la información. Dentro de una red LoRaWAN se pueden distinguir cuatro entidades: nodos, gateways, servidores de red y servidores de aplicación.

Todas las características de LoRa y LoRaWAN se explican con más detalle en secciones siguientes.

## 2.2. Nodos LoRaWAN

Los nodos son los dispositivos que envían o reciben información a la red LoRaWAN. Los datos enviados por los nodos deben pasar por un gateway para que este reenvíe los datos al servidor de red. Puede existir el caso en el que la información de un nodo sea recibida por varios gateways, situación en la que será el servidor de red el que elija los datos de qué gateway obtiene la información en base a parámetros de calidad de servicio.

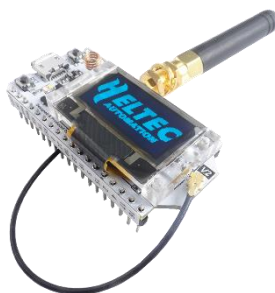
En la actualidad existen varias empresas que desarrollan dispositivos que utilizan LoRaWAN para el envío de información, todos ellos están basados en los chips de Semtech SX127x [4], destacando los siguientes dispositivos:

- Dispositivos basados en el microcontrolador ESP32: Debido a la popularidad del dispositivo ESP32 surgen varios sistemas que hacen uso de este microcontrolador para gestionar las comunicaciones con el módulo LoRa. Estos dispositivos, principalmente, hacen uso de la librería LMIC [5] que fue desarrollada por IBM y se ha adaptado a los dispositivos compatibles con Arduino. Se puede conectar una placa Arduino a un módulo con un módulo SX127x a través del bus SPI y así poder desarrollar una aplicación LoRaWAN como se muestra en la Fig. 6.



*Fig. 6. Arduino Mega y módulo LoRa.*

Otra opción es usar una placa de desarrollo que incluye todos los componentes necesarios. Un ejemplo son las tarjetas Heltec como la de la Fig. 7 que están basadas en el microcontrolador ESP32 y permiten hacer uso tanto de la librería LMIC como de una librería desarrollada por Semtech (LoRaMAC-node) [6] que ha sido adaptada al microcontrolador ESP32.



*Fig. 7. Módulo Heltec ESP32 LoRa.*

- La empresa Dragino [7] también dispone de un amplio abanico de soluciones basadas en LoRaWAN. Ponen a disposición del cliente una serie de dispositivos con una función predefinida y además proporcionan el código fuente para poder modificar la funcionalidad de los dispositivos. Sus soluciones están basadas en microcontroladores de bajo consumo de ST Microelectronics, así como en microcontroladores compatibles con el IDE de Arduino para facilitar su programación. Por ejemplo, el dispositivo denominado LBT1 [8], véase Fig. 8, es un escáner de dispositivos Bluetooth para estimar la posición en entornos interiores, también dispone de un acelerómetro para el seguimiento de la actividad física. La información de interés la transmite a la nube vía LoRaWAN.



Fig. 8. Dragino LBT1.

- El fabricante ARM junto con Semtech también desarrollan tarjetas de evaluación basadas en LoRaWAN que se pueden acoplar a las tarjetas de evaluación de ST Microelectronics como se puede observar en la Fig. 9. La principal ventaja de utilizar esta combinación es que ST proporciona una librería [9] para su programación lo que facilita su puesta en funcionamiento.

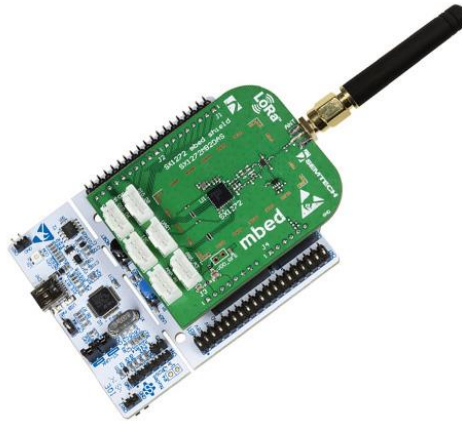


Fig. 9. Tarjeta de evaluación de ST junto con tarjeta de evaluación de LoRa.

## 2.3. Gateways LoRaWAN

Un gateway LoRaWAN es un dispositivo que hace de enlace entre los nodos y el servidor de red. Tienen la capacidad de recibir mensajes del servidor y enviarlos a los nodos y viceversa. Con un solo gateway se puede dar cobertura a una gran zona de cobertura, por ejemplo, en zonas rurales con línea de vista se pueden alcanzar distancias de 15km. En concreto, los principales gateways disponibles en la actualidad son:

- Wirnet Station [10]: es un gateway para instalar en exteriores de la compañía Kerlin. Fue el primer gateway comercial disponible internacionalmente puesto a la venta en 2014. Sus principales características son: Dispone de 49 demoduladores sobre 9 canales de comunicación. Lleva incluido un receptor GPS de alta sensibilidad. Diseñado con filtros SAW para evitar interferencias con bandas de frecuencia adyacentes. Conectividad con Internet mediante Ethernet y comunicaciones móviles (GPRS, EDGE, HSDPA y UMTS).





*Fig. 10. Gateway Wirnet Station.*

- Multitech Multiconnect [11]: Es un gateway muy versátil ya que permite intercambiar ranuras extraíbles de forma que se pueden implementar en él distintos tipos de comunicaciones. Dispone de un receptor GPS para obtener las marcas de tiempo y geolocalización con LoRaWAN. También se puede elegir la forma de envío de datos hacia la nube: WiFi, Ethernet o comunicaciones móviles como 3G o 4G.



*Fig. 11. Gateway Multitech Multiconnect.*

- The Things Indoor Gateway [12]: Es un gateway desarrollado por The Things Industries para entornos interiores. Es de bajo coste, se alimenta mediante un conector USB-C y puede consumir hasta 900mA. Su configuración se realiza a través de una interfaz gráfica de forma sencilla. Se conecta a Internet mediante WiFi.



Fig. 12. The Things Indoor Gateway.

- Lorix One [13]: Es un gateway desarrollado por la compañía Wifx. Está diseñado para ser instalado en entornos exteriores ya que tiene varias versiones con distintos certificados de protección: IP65 o IP43. Se alimenta a través de un PoE (Power over Ethernet) de 24V y consume 3W. Lleva instalado un sistema operativo basado en Yocto y se puede acceder a el para su configuración mediante SSH o a través de un USB. Lleva preinstalados varios packet forwarder de forma que facilita la puesta en marcha ya que solo se debe elegir el servidor con el que se va a comunicar. Es compatible con varias antenas, en concreto con dos exteriores de fibra de vidrio de 3dBi o 5dBi, y con una interior de 2dBi.

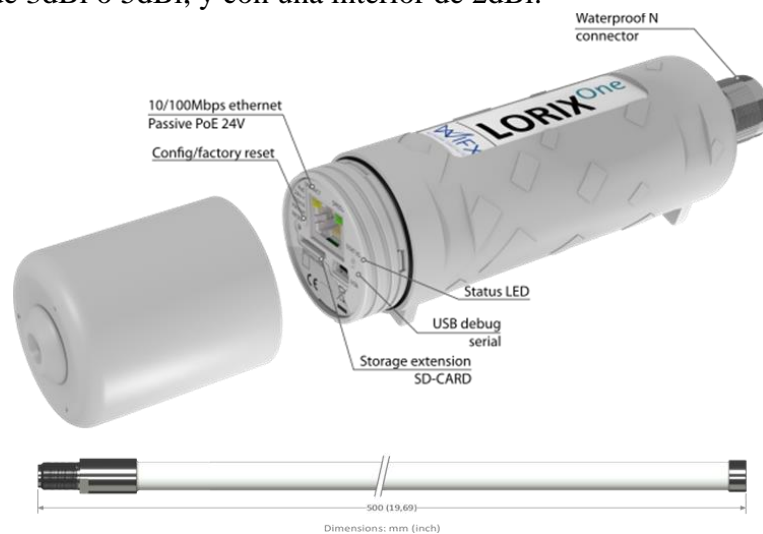


Fig. 13. Gateway Lorix One.

Además de los dispositivos comerciales descritos anteriormente, también surgen iniciativas en el mundo Maker para hacer proyectos DIY (Do It Yourself) disponibles para toda la comunidad, las principales plataformas de evaluación donde se desarrollan estos proyectos están basadas en Raspberry, en Arduino o en ESP32. A continuación, se describen dos de los proyectos de puesta en marcha de un gateway más usados en la comunidad DIY.

- TTGO LoRa32 V1 868 MHz: es una placa desarrollo de reducido tamaño y bajo precio que incluye un microcontrolador ESP32 de doble núcleo, el chip de Semtech SX1276 y un conector UFL para una antena externa de uso exclusivo para LoRa. Además, tiene disponible comunicación WiFi y Bluetooth. Es muy sencilla de programar ya que se puede realizar a través del IDE de Arduino, donde la mayoría de las librerías de Arduino son compatibles con esta placa o requieren de pequeñas modificaciones.





Fig. 14. Tarjeta TTGO LoRa32 V1 868 MHz.

Para esta placa cabe destacar un proyecto de código abierto que está publicado en Github [14]. Con este proyecto se puede crear un gateway de un canal de forma sencilla y económica. Aunque en las especificaciones de LoRaWAN los gateways deben ser de 8 canales, con este proyecto se pueden realizar pruebas a un coste muy reducido.

- Raspberry Pi con concentrador RAK833: La empresa RAKWireless tiene disponible un repositorio en Github [15] con todas las instrucciones necesarias para montar un gateway con una Raspberry Pi y todos los concentradores que vende. El concentrador más popular es el RAK833 debido a sus buenas prestaciones y precio ajustado: Dispone de interfaz PCI Express Mini para acceder a los pines del bus SPI desde donde se controla el módulo. Basado en el chip SX1301. Lleva incorporados 8 demoduladores para el enlace ascendente y 1 demodulador para el enlace descendente. Potencia de transmisión máxima de 20 dBm. Alta sensibilidad de recepción de -136 dBm



Fig. 15. Raspberry Pi con concentrador RAK833 [16].

## 2.4. Servidores de red LoRaWAN

En esta sección se describen de forma general los principales servidores de red que hay en la actualidad, mostrando sus principales características.

- TTN (The Things Network) [17]: es un proyecto o una iniciativa para dar cobertura global LoRaWAN. Se puede utilizar de forma gratuita haciendo uso de los dispositivos que otros usuarios registran en la red. También permite aplicar soluciones comerciales ya que dispone de una capa de pago. Las integraciones que implementa son con los principales servidores que

hay en la actualidad, como AWS, Azure o Google; también se puede acceder a los datos vía MQTT o HTTP. La arquitectura de red usando TTN se muestra en la Fig. 16. El principal inconveniente del uso de la capa gratuita de TTN es que no se tiene ningún tipo de control por lo que si por cualquier motivo queda fuera de servicio hay que esperar a que los administradores soluciones los problemas, quedando los nodos fuera de servicio.

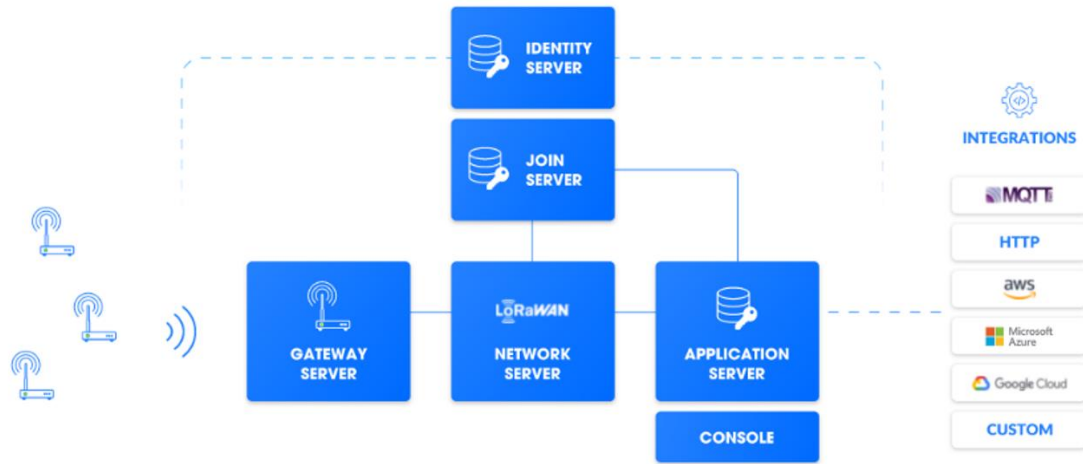


Fig. 16. Arquitectura de red usando TTN.

TTN tiene implementado un mapa interactivo, véase Fig. 17, donde se muestran los gateways disponibles en cualquier lugar del mundo. Estos gateways han sido registrados por usuarios de forma altruista para dar cobertura LoRaWAN a su zona y de esta forma otros usuarios puedan hacer uso de TTN sin necesidad de instalar un gateway. En la actualidad TTN está presente en 151 países, tiene 153000 miembros y hay registrados en la plataforma un total de 21500 gateways.



Fig. 17. Mapa de TTN con los gateways disponibles.

- ChirpStack [18]: Es una solución completa para implementar la pila LoRaWAN. Es un proyecto abierto desarrollado por el ingeniero Orne Brocaar. Todo el proyecto tiene una licencia de tipo MIT por lo que se puede usar con fines comerciales. Los componentes que forman la pila son independientes y el desarrollador que la utilice puede instalar los que sean de utilidad, por lo que permite la integración en arquitecturas existentes sin tener que realizar modificaciones. Incluye una interfaz web de configuración y administración de los

dispositivos que forman la red, así como una API para favorecer la integración con otros servicios. Todos los componentes que incluye ChirpStack se presentan de forma esquemática en la Fig. 18.

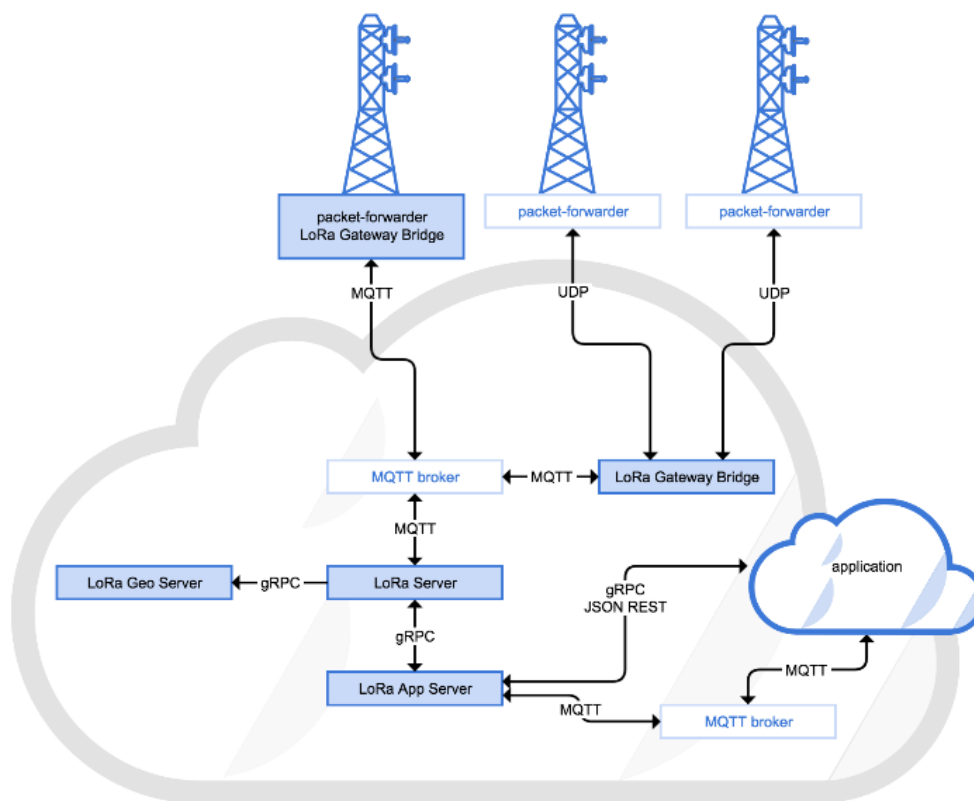


Fig. 18. Arquitectura de ChirpStack.

- Lorient Network Server [19]: Lorient es una empresa de Suiza fundada en 2015 que se dedica a implementar soluciones para el Internet de las Cosas. Ofrece un plan gratuito para implementar la capa LoRaWAN y otros planes de pago que se pueden adaptar a cualquier empresa para implementar soluciones comerciales. Las principales características del servidor de red Lorient son: Seguridad, escalabilidad, opciones de integración e independencia. La principal ventaja del uso de Lorient es la sencillez de configuración de los dispositivos y la transparencia en la gestión del servidor.

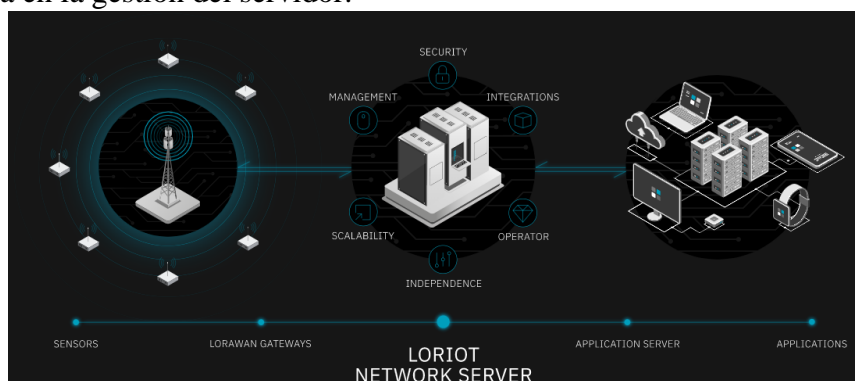


Fig. 19. Arquitectura de red con Lorient.



## 3. Tecnología LoRa

### 3.1. Introducción

LoRa es el acrónimo de Long Range y consiste en una modulación de radiofrecuencia que permite el envío de información a grandes distancias debido a su robustez. La tecnología LoRa fue desarrollada por la empresa Cycleo en el año 2009 en Francia, posteriormente fue comprada por la empresa Semtech de Estados Unidos, la cual tiene la patente, por lo que puede ser exclusivamente implementada por esta empresa. Dentro del modelo OSI (Open Systems Interconnection) se enmarca en la capa 1 ya que es la tecnología física con la que se puede establecer las comunicaciones. Esta tecnología inalámbrica se basa en establecer comunicaciones de largo alcance utilizando poca energía y enviando reducida cantidad de datos, por lo que es adecuada para dispositivos del Internet de las Cosas que estén alimentados con baterías, ya que estas pueden durar incluso años. En espacios urbanos tiene un alcance entorno a 3km, y en entornos rurales con línea de vista puede superar los 15km de alcance. Para conseguir todo ello, esta tecnología tiene unas características particulares que se describen en los apartados siguientes. En la Fig. 20 se puede observar una comparación entre varias tecnologías en cuanto a su alcance de transmisión y su ancho de banda.

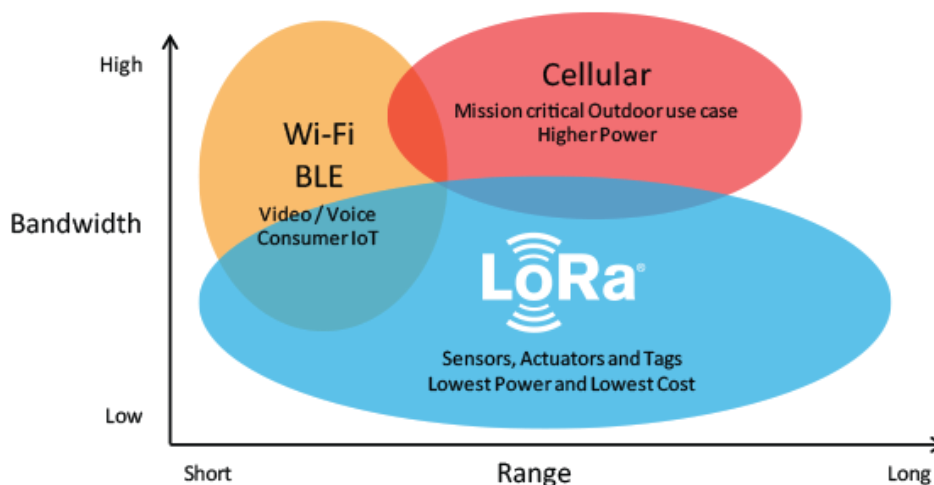


Fig. 20. Comparación de LoRa con otras tecnologías[24].

### 3.2. Modulación

LoRa está basada en una modulación de espectro ensanchado CSS (Chirp Spread Spectrum). Esta modulación consiste en aumentar o disminuir la frecuencia de una señal desde una frecuencia inicial hasta una frecuencia final para crear los símbolos de transmisión, llamados chirps. En la se muestra el aspecto de un chirp ascendente, ya que va aumentando la frecuencia de la señal progresivamente.

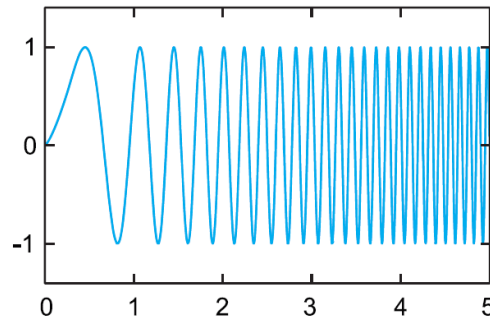


Fig. 21. Chirp ascendente [20].

La modulación implementada por LoRa es de banda ancha, véase Fig. 22, por lo que en el receptor el nivel de ruido será mayor que si se utilizase por ejemplo una modulación de banda estrecha como Ultra Narrow Band.

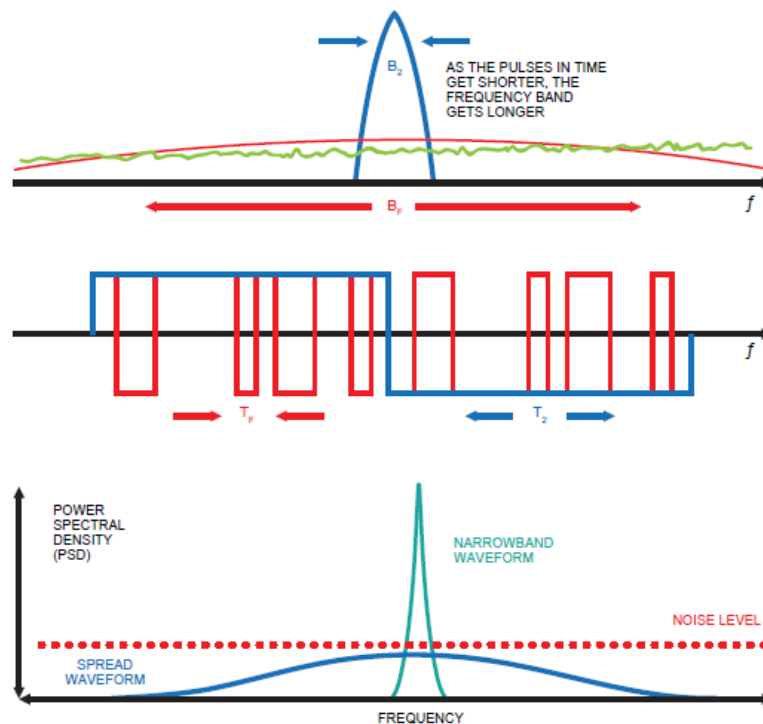


Fig. 22. Banda ancha vs banda estrecha [20].

En la Fig. 23 se muestra un ejemplo de información modulada con LoRA. Se puede observar cómo los símbolos están formados por chirps que inician en una determinada frecuencia y acaban en otra.

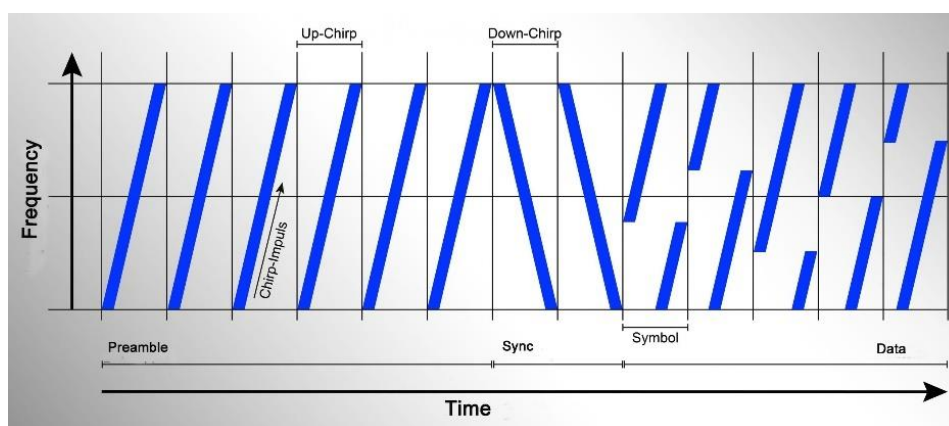


Fig. 23. Modulación LoRa [25].

### 3.3. Frecuencias utilizadas

LoRa usa el espectro de comunicación por debajo de 1 GHz, según el país donde se establezca la comunicación se puede utilizar la banda de 433MHz en Asia, 868MHz en Europa o 915MHz en América del Norte. Estas bandas de frecuencia se enmarcan dentro de las bandas ISM (Industriales, Científicas y Médicas) las cuales están reservadas internacionalmente para estos propósitos. Son bandas que se pueden utilizar sin licencia por lo que están reguladas para evitar que colapsen.

El espectro radioeléctrico es un recurso limitado por lo que existen organismos que redactan leyes para la correcta asignación a cada tecnología. En concreto, como LoRa utiliza las bandas ISM, en Europa se tiene que cumplir que la potencia de transmisión debe ser como máximo de 14dBm. También está limitado el tiempo de utilización del canal de transmisión que debe ser del 1% como máximo. Este parámetro es llamado Tiempo en el aire (ToA, Time on Air) y depende del factor de dispersión, del ancho de banda utilizado, de la tasa de codificación y de las longitudes de la carga útil y de las cabeceras. Hay calculadoras [26] que permiten obtener el tiempo en el aire del mensaje LoRa además de calcular el ciclo de trabajo, es decir, cada cuanto tiempo se puede enviar un nuevo mensaje.

En Europa se utiliza de forma general la banda de 868MHz y los canales asociados a esta banda se describen en la Tabla I para los mensajes de uplink.

Canal	Frecuencia	Factor de dispersión y Ancho de banda
1	868.1MHz	SF7-SF12 con BW125
2	868.3MHz	SF7-SF12 con BW125 y SF7 con BW250
3	868.5MHz	SF7-SF12 con BW125
4	867.1MHz	SF7-SF12 con BW125
5	867.3MHz	SF7-SF12 con BW125
6	867.5MHz	SF7-SF12 con BW125
7	867.7MHz	SF7-SF12 con BW125
8	867.9MHz	SF7-SF12 con BW125
9	868.8MHz	FSK

Tabla I. Frecuencias de uplink en la banda de 868MHz.

En el envío de mensajes de downlink, para la ventana de recepción primera se utilizan los mismos canales que para el enlace de subida; mientras que para la ventana de recepción segunda se utiliza la frecuencia de 869.525MHz con un factor de dispersión de 9 y un ancho de banda de 125kHz.



Debido a la alta sensibilidad en recepción y a que la frecuencia de transmisión no es muy elevada se puede lograr una buena penetración en edificios, obteniendo rangos de alcance en zonas urbanas de pocos kilómetros.

### 3.4. Factor de dispersión

El factor de dispersión es el número de bits por símbolo que se utilizan en la transmisión. Se puede configurar con un valor de 6, 7, 8, 9, 10, 11 o 12. Este valor influye en el espectro que utiliza la modulación.

A mayor valor de factor de dispersión mayor es el tiempo de símbolo lo que implica que el tiempo en aire de la información es mayor también, por lo que aumenta la sensibilidad del receptor, aumentando el alcance de la transmisión.

### 3.5. Ancho de banda

Se pueden elegir tres opciones de ancho de banda de transmisión, 125 kHz, 250 kHz y 500 kHz. Existe una relación entre ancho de banda y velocidad de transmisión, a mayor ancho de banda elegido, mayor será la velocidad de transmisión. Pero también existe una relación entre ancho de banda y alcance, a mayor ancho de banda utilizado, menor es el alcance que se puede conseguir en la transmisión de información.

### 3.6. Tasa de codificación

La tasa de codificación es un parámetro que indica el número de bits útiles de información que van a ser codificados respecto al total de bits codificados que se envía en una transmisión. Hay disponibles cuatro tipos de tasa de codificación, que son, 4/5, 4/6, 4/7 y 4/8. Esta notación indica que por cada 4 bits de información útil se van a codificar 5, 6, 7 u 8 bits totales respectivamente.

Una tasa de codificación menor implica que el paquete enviado va a estar más tiempo en el aire y va a tardar más la transmisión de ese paquete. Al estar más tiempo en el aire cada símbolo, el receptor puede demodular la información que recibe con menor potencia, aumentando la sensibilidad del receptor, lo que aumenta el link Budget del enlace. También, una tasa de codificación de 4/8 consume mayor energía que una tasa de codificación de 4/5, lo que es negativo cuando se utilizan baterías.

### 3.7. SNR

El receptor puede recibir información con una transmisión LoRa 20dB por debajo del nivel de ruido, lo que hace que sea una modulación robusta frente al ruido. Esto implica que el receptor debe tener una sensibilidad de -137dBm como máximo. Esa recepción con 20dB por debajo del nivel de ruido se puede producir cuando la tasa de transmisión de datos es baja, ya que al aumentar la tasa de transmisión aumenta el ancho de banda por lo que el nivel de ruido aumenta en el receptor haciendo que la relación señal-ruido sea peor. Los elementos que influyen en la sensibilidad del receptor se resumen en la Fig. 24.



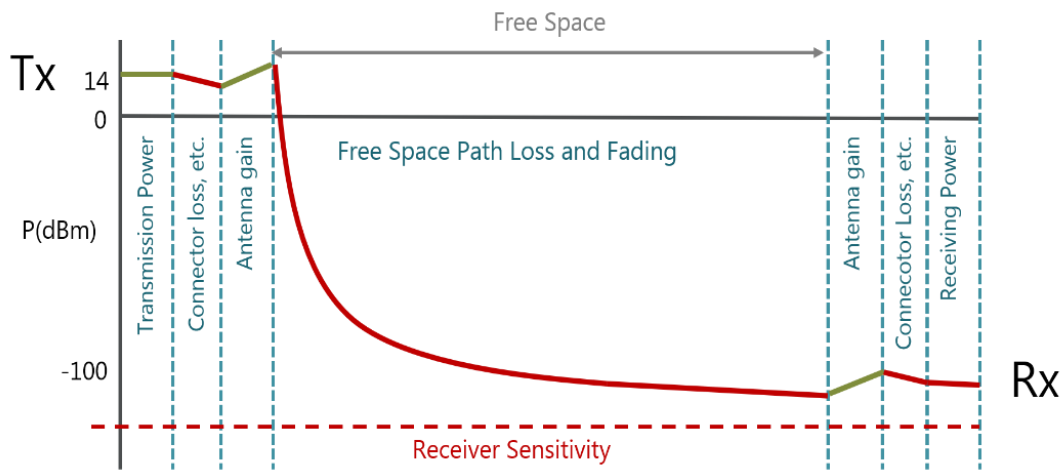


Fig. 24. Link Budget LoRa.

### 3.8. Aplicaciones

Los dispositivos que utilizan LoRa están revolucionando el mercado del Internet de las Cosas gracias a que son capaces de transmitir información a grandes distancias con un consumo de energía reducido. Debido a ello, surgen numerosas aplicaciones que pueden utilizar LoRa como tecnología de comunicaciones, destacando:

- Agricultura inteligente: Se puede tener un control de todos los parámetros ambientales de los cultivos de forma que se maximiza la capacidad de producción. También se puede estimar de forma más eficiente el momento idóneo de recogida de los frutos.
- Ciudades inteligentes: Mediante dispositivos LoRa se pueden tener controlados casi todos los servicios que ofrece una ciudad para optimizar su utilización por parte de las personas. Se puede controlar el momento idóneo de encendido de luces, monitorizar las plazas de aparcamientos públicos o monitorizar la recogida de residuos para optimizar la ruta.
- Salud inteligente: Debido al bajo consumo se puede hacer una monitorización continua de pacientes, de sistemas o de activos durante las 24 horas del día para garantizar que se cumplen todas las especificaciones de forma adecuada.
- Logística: Con la tecnología LoRa se puede hacer un seguimiento de toda la cadena de suministro, ya sea con parámetros medibles como la temperatura o con la posición de los activos. Todo ello se puede aplicar a productos que se encuentran en almacenes o a productos que están en ruta hacia un destino.
- Control industrial: Con sensores LoRa se puede tener monitorizada toda la maquinaria de una industria. Con los datos obtenidos se pueden analizar para hacer una optimización de los procesos productivos o para realizar un mantenimiento predictivo de la maquinaria, lo que implicaría grandes beneficios para las empresas.
- Contadores inteligentes: Las empresas pueden obtener la información de los contadores sin tener que desplazarse a los domicilios lo que implica un ahorro de costes y de personal. Se podría aplicar a todo tipo de contadores, de electricidad, de agua, etc.



## 4. Protocolo LoRaWAN

### 4.1. Introducción

LoRaWAN es un protocolo de comunicaciones para crear redes de LPWAN que utiliza la tecnología de capa física LoRa como medio de envío de datos. El envío de los datos es bidireccional, es decir, los nodos pueden enviar datos a la red y también pueden recibir datos de la red.

Permite la creación de redes en estrella de forma que todo tipo de comunicación debe pasar por un Gateway, no siendo posible la comunicación entre dos nodos finales directamente. Esto es debido a que se utiliza el protocolo LoRaWAN, pero utilizando la tecnología LoRa, si se pueden comunicar dos nodos entre sí.

El intercambio de información entre gateway e Internet se realiza a través del protocolo IP ya sea mediante una conexión Ethernet o por comunicaciones móviles como 3G o 4G. Por ejemplo, el gateway Lorix One que se va a utilizar en este proyecto dispone de comunicación Ethernet ya que no lleva incorporado un módem de comunicaciones móviles.

LoRaWAN permite comunicaciones bidireccionales, es decir, un nodo puede enviar datos al servidor y también el servidor puede enviar datos a un nodo. El primer caso se conoce como mensajes de uplink y el segundo caso como mensajes de downlink. El protocolo no implementa un mecanismo de acceso al medio, básicamente desde que se da la orden de transmisión se espera un pequeño intervalo de tiempo aleatorio para iniciar el proceso, por lo que puede haber colisiones que impidan recibir la información en el receptor.

LoRaWAN es un protocolo de capa MAC (Medium Access Control) que define todas las características y permite gestionar los dispositivos para implementar de forma correcta este tipo de comunicaciones. En el modelo OSI (Open Systems Interconnection) se enmarca en la capa 2 como se muestra en el diagrama de la Fig. 25. Este protocolo es abierto y está definido por la organización LoRa Alliance, mientras que la capa física es propietaria bajo la compañía Semtech.

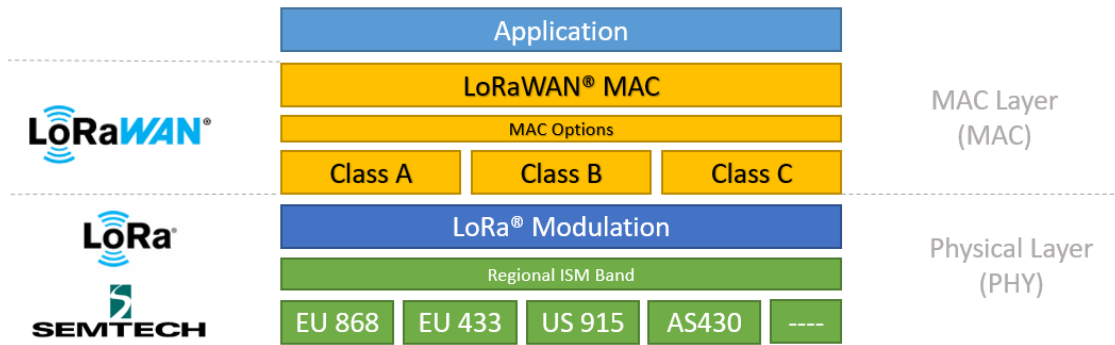


Fig. 25. Estructura de capas del protocolo LoRaWAN[20].

La arquitectura de una red basada en LoRaWAN es de topología en estrella donde el flujo de comunicaciones sigue un camino ascendente si el nodo envía datos o descendente si el nodo recibe datos. La arquitectura está formada, principalmente, por tres tipos de elementos. Los nodos son los encargados de enviar o recibir información. Esta información se transmite por radiofrecuencia, vía LoRa, y es recibida por los gateways que están disponibles en su zona de cobertura, los cuales se encargan de reenviar la información a la nube o a un servidor local mediante TCP/IP. Por último, los datos llegan a un servidor de red, que tiene la función de administración de la red para redirigir cada dato a la aplicación que le corresponde.

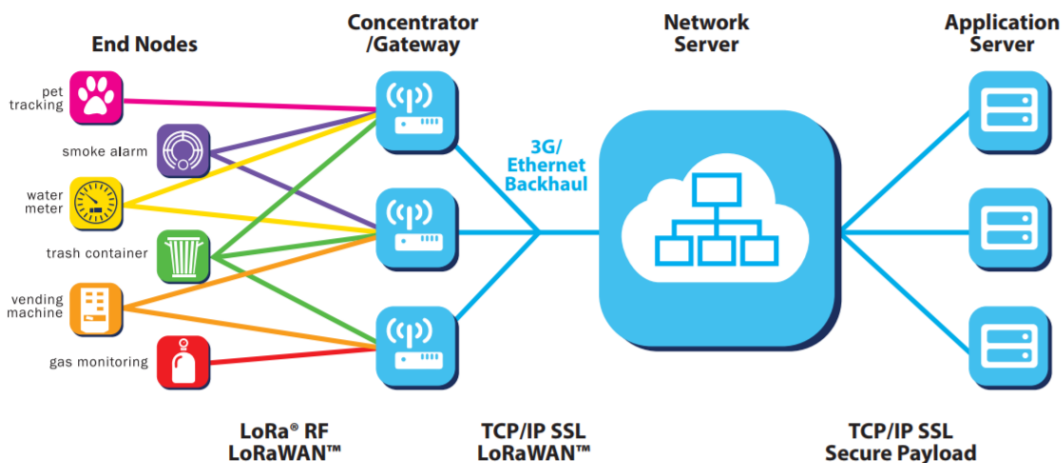


Fig. 26. Arquitectura de una red LoRaWAN[21].

## 4.2. Clases

Los dispositivos de una red LoRaWAN soportan comunicaciones bidireccionales, pero no pueden emitir y recibir información al mismo tiempo por lo que debe existir un mecanismo que defina los tiempos de emisión y recepción, de esta forma, pueden configurarse según tres tipos de clases diferentes. La apertura de ventanas de recepción es un parámetro clave en este protocolo ya que tiene que ser preciso para que los nodos puedan recibir información de la red. Estas ventanas deben abrirse en el momento adecuado, admitiendo una desviación, para que la comunicación sea exitosa.

Clase A: en esta clase se abren dos ventanas de recepción de datos en instantes de tiempo específicos solo después de que el nodo envíe datos por iniciativa propia, como se muestra en la Fig. 27. Esta

clase es ideal para nodos que requieran de una duración de batería alta ya que los nodos envían datos y la red solo se puede comunicar con ellos después de ese envío de datos.

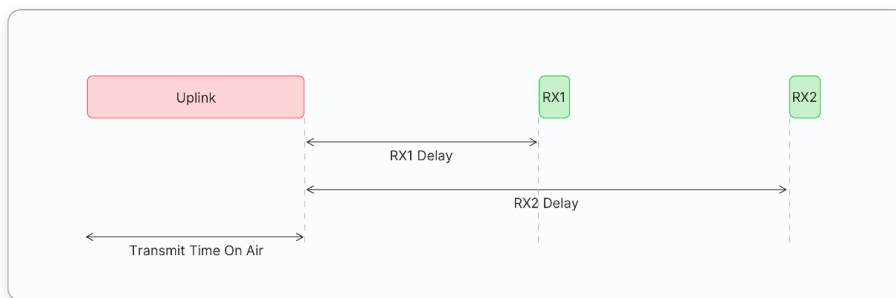


Fig. 27. Ventanas de recepción en clase A [22].

Clase B: con esta clase el nodo final puede recibir mensajes durante intervalos de tiempo llamados “Ping Slots” o después del envío de un mensaje como ocurre en los dispositivos configurados con la clase A. Se utilizan balizas que se emiten de forma periódica por el gateway para sincronizar el envío de datos y así abrir las ventanas de recepción. Estos dispositivos consumen mayor energía que los dispositivos de clase A ya que tienen mayor tiempo abierta la ventana de recepción de datos.

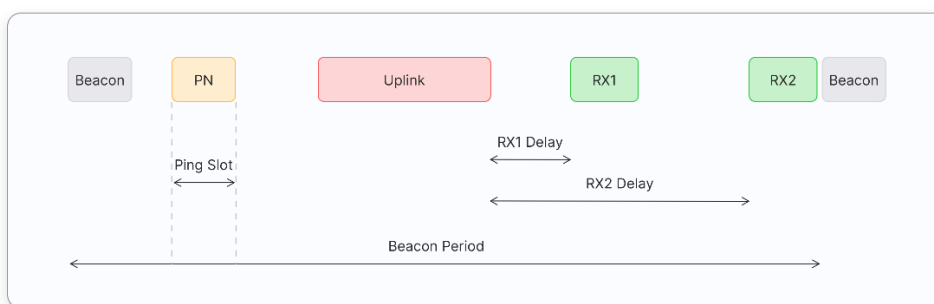


Fig. 28. Ventanas de recepción en clase B [22].

Clase C: en esta clase se mantiene la ventana de recepción de datos siempre abierta, pero se sigue el mismo patrón de apertura de ventanas que en la clase A. La ventana de recepción solo se mantiene cerrada cuando se está realizando una transmisión de información por parte del nodo. Por tanto, esta es la clase que más energía consume.

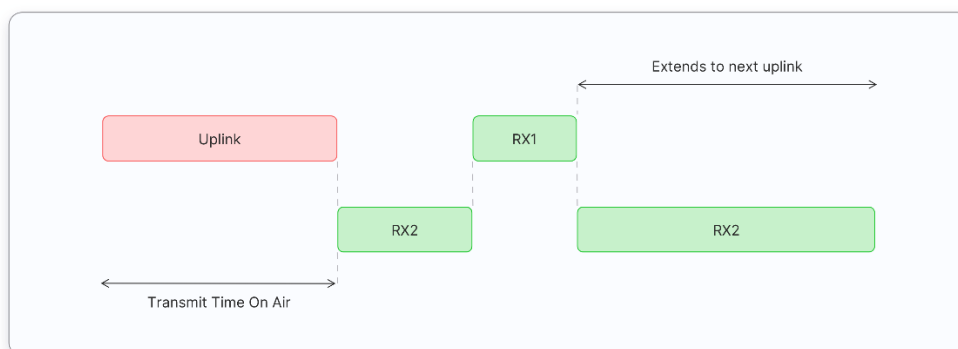


Fig. 29. Ventanas de recepción en clase C [22].

### 4.3. Estructura de las tramas LoRaWAN

Los dos tipos de mensajes que pueden enviarse en una red LoRaWAN, mensajes de uplink y downlink, se diferencian en algunos campos de las tramas que son enviadas. En la Fig. 30 se muestra de manera esquemática la estructura de cada trama, junto con el tamaño de cada campo en bits. En la capa física también se incluye un campo de preámbulo y un campo de CRC solo en los mensajes de uplink.

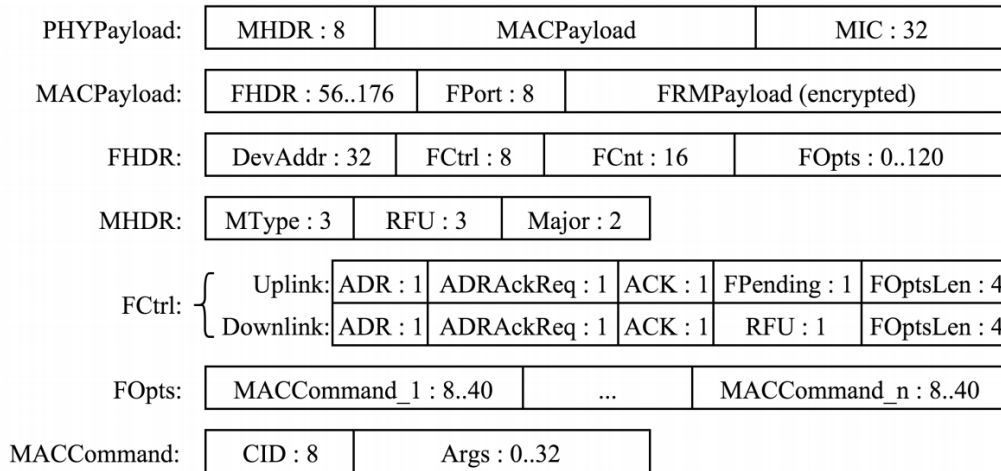


Fig. 30. Estructura de mensajes LoRaWAN [23].

En base al esquema de la Fig. 30 se describen de forma general las funciones de cada campo:

- **PHYPayload:** Dentro de la carga de la capa física se encuentran tres campos. El campo MHDR es la cabecera e indica el tipo de mensaje que se está enviando dentro de la carga útil de la capa MAC (MACPayload). El campo MIC (Message Integrity Check) es un indicador de integridad para asegurarse de que el mensaje no se ha modificado.
- **MACPayload:** FHDR es la cabecera de la trama MAC. FPort indica el puerto por el que se transmite el mensaje. FRMPayload incluye la información útil a transmitir desde el nodo, el cual está cifrado con las claves de sesión.
- **FHDR:** Es la cabecera de la trama MAC, incluye el campo DevAddr, que es la dirección única del nodo. El campo FCtrl tiene funcionalidad de control y FCnt es un contador. FRMPayload contiene comandos MAC.
- **MHDR:** Es la cabecera de la trama física. MType indica el tipo de mensaje que se va a enviar. Los campos RFU y Major se utilizan para el proceso de unión a la red LoRaWAN.

### 4.4. Seguridad

Las redes LoRaWAN han sido diseñadas para ser seguras, siendo una de las partes más complicadas de entender de este sistema. Existe seguridad en diferentes capas que se basan en encriptación AES (Advanced Encryption Standard) con diferentes claves de 128 bits cada una.

LoRaWAN utiliza LoRa para el envío de los datos, al utilizar radiofrecuencia para ello, los mensajes son difundidos al espacio y cualquier persona puede interceptarlos y realizar un ataque modificando el contenido de los mensajes. Por esto hay que aplicar mecanismos de seguridad, que en LoRaWAN se basan principalmente en claves y en un contador de tramas. Con las claves se encriptan los mensajes y los protegen frente a posibles manipulaciones. Con el contador de tramas se evita que los

mensajes sean interceptados y reenviados para realizar ataques de repetición. Para ello se implementan dos contadores de tramas para el enlace ascendente FCntUp y para el enlace descendente FCntDown, que al inicio de la activación del nodo se reinician a valor 0 y cada vez que se envía un mensaje por parte del nodo o hacia el nodo se incrementará en una unidad su contador asociado. Así, solo se aceptarán los mensajes que porten un contador de trama superior al anterior. Si ocurre al contrario el mensaje se descartaría.

## 4.5. Modos de activación

Existen dos tipos de activación de nodos en una red LoRaWAN, de esta forma se provee de seguridad al sistema para que los nodos sean autenticados por el servidor de red para el envío de datos. Para realizar el proceso de activación de un nodo se puede hacer de dos formas distintas, con OTAA (Over The Air Activation) o con ABP (Activation By Personalization).

La principal diferencia entre los dos métodos de activación es que mediante ABP las claves necesarias para la autenticación se almacenan de forma estática en el nodo, mientras que usando OTAA las claves de autenticación se renuevan cada sesión mediante un intercambio de mensajes entre el nodo y el servidor de red, por lo que es una activación dinámica. ABP es una activación menos segura que OTAA, ya que al tener las claves estáticas puede comprometer la aplicación ante una vulnerabilidad del sistema.

Para el desarrollo del proceso de activación hay una serie de claves que se deben utilizar:

- DevAddr: es la dirección del dispositivo formada por 32 bits. Los bits <31:25> almacenan el identificador de red NwkID y los bits <24:0> almacenan la dirección de red NwkAddr.
- AppEUI: es el identificador de aplicación donde el nodo debe enviar los datos, que serán redirigidos por el Servidor de red.
- NwksKey: es la clave de sesión de red, utilizada para dotar de seguridad a las comunicaciones entre nodo y servidor de red mediante verificación MIC, que es un procedimiento de control para que, si un mensaje es interceptado y modificado, pueda ser identificado. Esta clave es única para cada nodo y para cada sesión activa del nodo. Si el nodo se activa mediante OTAA, esta clave se regenera en cada activación, mientras que si se activa mediante ABP la clave se mantiene hasta que se cambie manualmente.
- AppSKey: es la clave de sesión de aplicación que se usa para cifrar y descifrar la carga útil de los mensajes, ya que la carga útil está cifrada entre el nodo y el servidor de aplicación para proteger los mensajes de posibles ataques. Al igual que ocurre con la clave NwksKey, esta clave también es única para cada nodo y por cada activación.
- AppKey: es la clave de aplicación utilizada solo en el método de activación OTAA.
- DevEUI: es el identificador del dispositivo.

Una vez que se ha realizado el proceso de activación, el nodo dispone de las claves DevAddr, AppEUI, NwksKey y AppSKey. Estas son las claves que se obtienen tras el proceso de activación vía OTAA, mientras que, si se utiliza ABP, son las claves que se deben almacenar de forma estática en el nodo.

Para obtener las claves de seguridad entra en acción el servidor de unión, siguiendo una serie de pasos:

- El nodo envía un mensaje de solicitud de unión (Join-request) al servidor de unión. El mensaje incluye los identificadores AppEUI y DevEUI; y un campo llamado DevNonce, que es un número aleatorio para evitar duplicados de solicitudes pasadas.
- Si el servidor de unión autentica correctamente al nodo, le contesta con un mensaje de aceptación (Join-accept). Este mensaje contiene los siguientes campos: AppNonce, es un valor aleatorio proporcionado por el servidor de red que utiliza el nodo para obtener las claves NwkSKey y AppSKey. NetID es el identificador de la red. DevAddr es la dirección del nodo. RxDelay es el tiempo que debe tardar en abrir la ventana de recepción el nodo. CFList es una lista opcional con los canales de frecuencia. DLSettings contiene información relacionada con las ventanas de recepción del nodo.

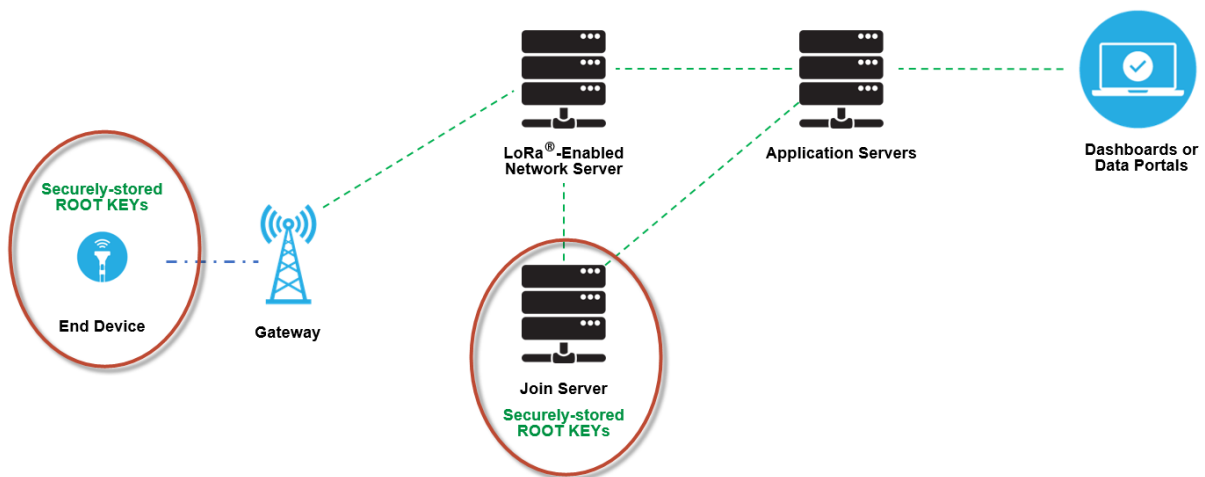


Fig. 31. Elementos implicados en el proceso de unión [20].



## 5. Diseño del hardware

La parte hardware del proyecto se corresponde principalmente con el diseño de una tarjeta PCB que tendrá la funcionalidad de nodo dentro de la red LoRaWAN a implementar. El nodo está basado en un microcontrolador de ST con arquitectura Cortex M3 de bajo consumo que es el encargado de gestionar toda la comunicación LoRaWAN además de recoger los datos de todos los sensores. Estos datos se transmiten a un servidor mediante una comunicación inalámbrica basada en la tecnología LoRa utilizando el chip SX1276. Como el sistema debe ser autónomo los módulos se alimentan con una batería que se recarga a través de un panel solar. El diagrama de bloques del sistema Hardware se muestra en la Fig. 32.

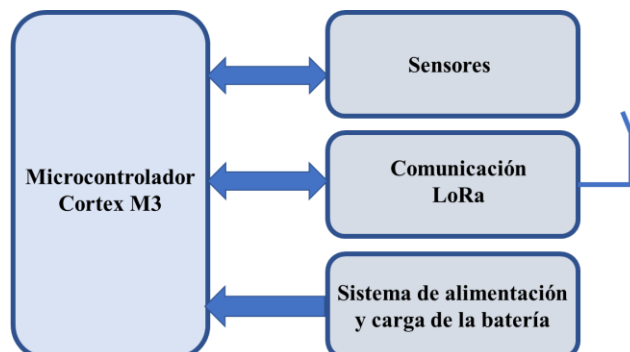


Fig. 32. Diagrama de bloques del sistema Hardware.

### 5.1. Microcontrolador

El microcontrolador elegido es del fabricante STMicroelectronics que ofrece una amplia gama de dispositivos, de los cuales se ha optado por la serie de bajo consumo L1, en concreto STM32L152RE. Lleva incorporada una memoria Flash de 512 Kbytes y 80 Kbytes de memoria RAM, las cuales son suficientes para la aplicación a desarrollar. El encapsulado es de 64 pines del tipo LQFP (Low-profile Quad Flat Package) de montaje superficial. Puede operar a una frecuencia de hasta 32 MHz. También dispone de los periféricos adecuados para desarrollar la aplicación, como interfaces I2C, UART, SPI, RTC, ADC, DAC, Timers de propósito general, etc.

En la Fig. 33 se muestra el diagrama de bloques del nodo a bajo nivel. Se utiliza el periférico USB para conectarse con un ordenador y poder transmitir mensajes de depuración. Con una interfaz UART se envían comandos al módulo GPS y se reciben las tramas NMEA para extraer la posición del nodo.

El sensor de lluvia es equivalente a un interruptor por lo que se conecta directamente a un pin de entrada del microcontrolador donde se comprueba si está a nivel alto o bajo. Haciendo uso del bus SPI se controla el chip SX1276 para establecer la comunicación LoRaWAN, además son necesarios una serie de pines de propósito general para controlar los eventos del chip y también para controlar el switch de radiofrecuencia PE4259. Al bus I2C se conectan los sensores restantes, el BME680, VEML6075, VEML7700 y CCS811, los cuales disponen de direcciones de esclavo diferentes para poder establecer la comunicación con cada uno de ellos correctamente desde el microcontrolador.

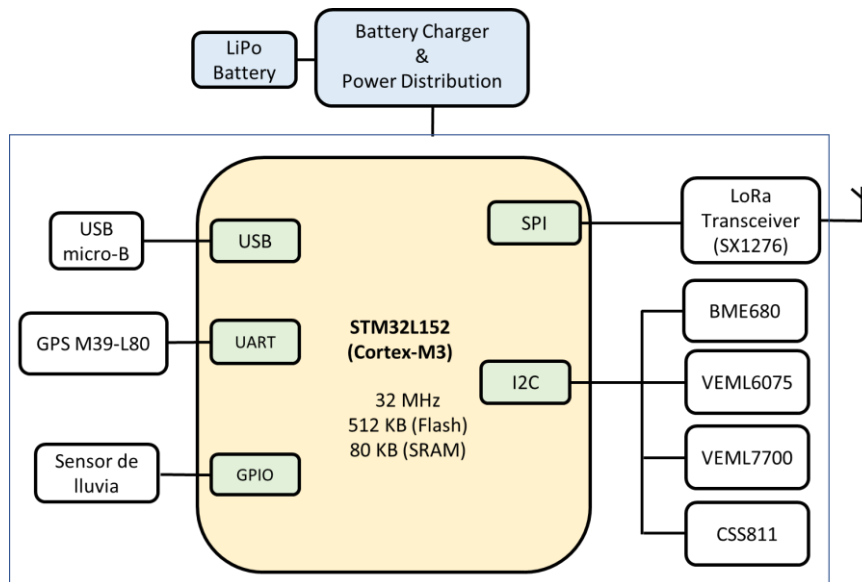


Fig. 33. Diagrama de bloques del nodo LoRa.

## 5.2. Sensores del nodo

El sistema a desarrollar en este trabajo consiste en la monitorización de parámetros ambientales mediante un dispositivo portátil que sea autónomo mediante la carga de su batería con un panel solar.

Se han elegido una serie de sensores para medir diferentes parámetros ambientales. Principalmente se han elegido sensores digitales, que están compuestos por un transductor que transforma una variación de un parámetro físico en una señal eléctrica (voltaje o corriente), posteriormente se aplicaran una serie de transformaciones de la señal analógica, como filtrado, amplificación, etc. para que un microcontrolador cuantifique la señal y la transforme a un valor digital mediante un ADC que será el que transmita por el bus digital que soporte I2C, UART, SPI típicamente. En este proyecto se ha elegido la interfaz I2C en los sensores que la tienen disponible debido a que no se requiere una velocidad elevada de lectura de datos, si así fuese, la interfaz SPI sería una opción más adecuada ya que permite velocidades superiores a I2C.

También hay que tener en cuenta el consumo de los sensores a utilizar. Los fabricantes suelen especificar datos de consumo para una frecuencia de lectura de datos de los sensores y cuando el sensor está en reposo. En este proyecto interesa principalmente el dato del consumo de los sensores cuando están dormidos debido a que la mayor parte del tiempo estarán configurados de esta forma, realizándose medidas puntuales con una frecuencia configurable por el usuario, típicamente del orden de minutos.

Para la elección de los sensores se han tenido en cuenta los parámetros técnicos que presenta cualquier sensor comercial:

- Precisión: dispersión de la medida realizada ante una misma magnitud física.
- Resolución: variación mínima de la magnitud física que puede ser medida por el sensor.
- Exactitud: diferencia entre el valor medido y el valor real de la magnitud física medida.
- Rango: valor máximo y mínimo de la magnitud física que va a ser medida.

### 5.2.1. Sensor de radiación UV

El sensor VEML6075 de Vishay es capaz de medir la radiación ultravioleta, pero a diferencia de otros sensores puede distinguir entre radiación ultravioleta en la banda A (UVA) y la radiación ultravioleta en la banda B (UVB) e incluso es capaz de proporcionar el índice ultravioleta (UV) real.

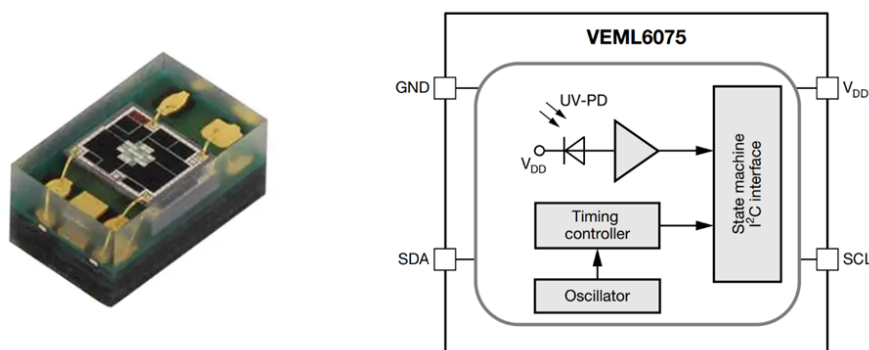


Fig. 34. Sensor VEML6075.

Las principales características de este sensor son:

- Bus de conexión mediante I2C.
- Voltaje de alimentación entre 1.7V y 3.6V.
- ADC de 16 bits de resolución.
- Corriente de shutdown de 800 nA.
- La dirección de esclavo para el bus I2C en formato 7 bits es 0x10.
- Buen rechazo de la luz visible e infrarroja.

Dispone de 13 registros, cada uno de 16 bits, de forma que se puede leer y/o escribir en ellos accediendo a la parte baja del registro (LSB) o a la parte alta (MSB). Aunque la mayoría de ellos están reservados, por lo que se procede a detallar los que van a ser usados:

- Registro UV\_CONF (0x00): tiene disponibles los 8 bits de menor peso para propósitos de configuración del sensor. Con el bit 0 (SD) se configura el estado del sensor, si se escribe un 0 está activado y si se escribe un 1 está en modo shut down. Con el bit 1 (UV\_AF) se deshabilita el modo forzado si se escribe un 0, o se habilita el modo forzado si se escribe un 1. Con el bit 2 (UV\_TRIG) se configura el modo de disparo de una sola medida si se escribe un 1. El bit 3 (HD) se usa para configurar el rango dinámico del sensor, si se escribe un 0 utilizará un rango dinámico normal, mientras que si se escribe un 1 se utiliza un rango dinámico alto. Los bits 4 al 6 (UV\_IT) sirven para configurar el tiempo de integración, siendo los valores disponibles de 50 ms, 100 ms, 200 ms, 400 ms y 800 ms.
- Registro UVA\_Data (0x07): contiene la lectura asociada a la radiación UVA.
- Registro UVB\_Data (0x09): contiene la lectura asociada a la radiación UVB.
- Registro UVCOMP1\_Data (0x0A): contiene la lectura asociada a la radiación UVcomp1.
- Registro UVCOMP2\_Data (0x0B): contiene la lectura asociada a la radiación UVcomp2.

- Registro ID (0x0C): almacena el identificador del dispositivo con valor 0x0026.

La lectura de la radiación ultravioleta se realiza a través de cuatro canales cada uno de 16 bits, obteniendo los datos de UVA, UVB, UVcomp1 y UVcomp2.

En la Fig. 35 se muestra la respuesta espectral normalizada, pudiendo observar que la respuesta relativa a la radiación UVA tiene el máximo pico de sensibilidad en 365 nm, mientras que la respuesta referente a la radiación UVB tiene el máximo pico de sensibilidad en 330 nm.

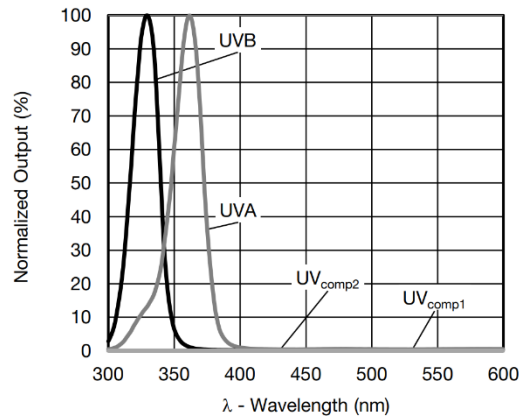


Fig. 35. Respuesta espectral sensor VEML6075.

## 5.2.2. Sensor BME680

Este sensor está fabricado por la empresa Bosch, siendo su principal característica que puede medir cuatro parámetros ambientales en un chip de reducido tamaño. Es capaz de medir la temperatura, humedad, presión barométrica y gases VOC (Volatile Organic Compounds).



Fig. 36. Sensor BME680.

Las principales características de este sensor son:

- La interfaz digital para comunicarse con el sensor para configuración o lectura de datos puede ser I2C o SPI, siendo I2C la elegida en este proyecto. La dirección de esclavo para el bus I2C en formato 7 bits es 0x77.
- La tensión de alimentación puede estar comprendida entre 1.7V y 3.7V.
- El consumo en modo dormido es de 0.15 $\mu$ A.
- Sensor de humedad: Puede medir la humedad relativa en un rango del 10% al 90%. Tiene un consumo máximo de 2.8 $\mu$ A. La precisión es del  $\pm$ 3% y su resolución típica es de 0.008%.

- Sensor de temperatura: Dispone de una resolución de salida de 0.01°C. El consumo de corriente es muy reducido, siendo de 1μA. La precisión del sensor en el rango de 0°C a 65°C es de ±1°C.
- Sensor de presión barométrica: Tiene un rango de medida de 300 a 1100 hPa. Puede ser configurado para leer datos a una tasa de muestreo de hasta 182 Hz. Además, debido a su alta precisión en la medida de la presión barométrica, de ±0.12 hPa, se puede utilizar como altímetro con una resolución de 1 metro. Consume un máximo de 4.2μA. La resolución de la presión cuando se configura con el mayor valor de sobremuestreo es de 0.18 hPa.
- Sensor de gases VOC: Es el sensor que más energía consume, ya que necesita una fase de calentamiento que puede llegar a consumir hasta 17mA, después en modo de bajo consumo solo consume 0.9mA. Tiene una resolución típica de 0.08% en la medida de resistencia. No tiene la capacidad de diferenciar entre gases presentes en el ambiente, pero proporciona un valor de resistencia relacionado con el contenido en el ambiente de compuestos orgánicos volátiles (VOC).

IAQ Index	Air Quality
0 – 50	good <sup>10</sup>
51 – 100	average
101 – 150	little bad
151 – 200	bad
201 – 300	worse <sup>2</sup>
301 – 500	very bad

Fig. 37. Air Quality Index.

La configuración y lectura de los datos de este sensor se puede realizar mediante unas librerías que proporciona el fabricante disponible en Github [29].

Este sensor dispone de varios registros de configuración y lectura de datos de los cuales se proporciona una descripción de los que se van a usar en este proyecto:

- reset (0xE0): escribir en valor 0xB6 produce un reset al sensor vía software con el mismo efecto que un reset vía hardware.
- id (0xD0): en este registro se almacena el valor 0x61, que es el identificador del sensor.
- ctrl\_meas (0x74): con los bits 1 y 0 se controla el modo de operación del sensor, con 00 se configura en modo dormido y con 01 en modo forzado.
- ctrl\_hum (0x72): controla la tasa de sobremuestreo del sensor de humedad.
- ctrl\_meas (0x74): controla la tasa de sobremuestreo del sensor de temperatura y del sensor de presión atmosférica.
- config (0x75): controla el coeficiente del filtro IIR que se puede aplicar a los datos de salida del sensor de temperatura y de presión.
- res\_wait\_0-9 (0x5A – 0x63): conjunto de registros para programar el valor de la resistencia de calentamiento del sensor de gas.
- gas\_wait\_0-9 (0x64 – 0x6D): conjunto de registros para indicar al sensor de gas el tiempo entre el inicio del calentamiento del sensor y el inicio de la conversión.
- ctrl\_gas\_0 (0x70): registro para parar la inyección de corriente al calentador del sensor de gas.
- ctrl\_gas\_1 (0x71): registro para parar iniciar la conversión del sensor y para indicar el perfil de calentamiento del sensor.

- `press_msb (0x1F)`, `press_lsb (0x20)` y `press_xlsb (0x21)`: contiene la medida en crudo de la presión atmosférica y se usa el byte de menor peso si se aumenta la resolución mediante el factor de sobremuestreo.
- `temp_msb (0x22)`, `temp_lsb (0x23)` y `temp_xlsb (0x24)`: contiene la medida en crudo de la temperatura y se usa el byte de menor peso si se aumenta la resolución mediante el factor de sobremuestreo.
- `hum_msb (0x25)` y `hum_lsb (0x26)`: contiene la medida en crudo de la humedad relativa.
- `gas_r_msb (0x2A)` y `gas_r_lsb (0x2B)`: contiene la medida de resistencia relacionada con los gases presentes en el medio.

### 5.2.3. Sensor de luz VEML7700

Este sensor fabricado por Vishay permite obtener la lectura de la iluminación ambiental en medida de lux, a diferencia de otros sensores que solo proporcionan un valor que es proporcional a esta medida, sin unidades. El lux es una unidad derivada de la candela que forma parte del Sistema Internacional de Unidades para la medida de la luminosidad.

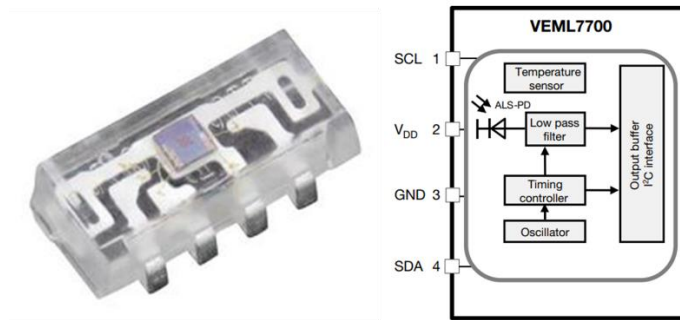


Fig. 38. Sensor VEML7700.

Las principales características de este sensor son:

- Interfaz de comunicación solo mediante I2C. La dirección de esclavo para el bus I2C en formato 7 bits es 0x10.
- Respuesta del sensor parecida a la que produce el ojo humano como se muestra en la Fig. 39.

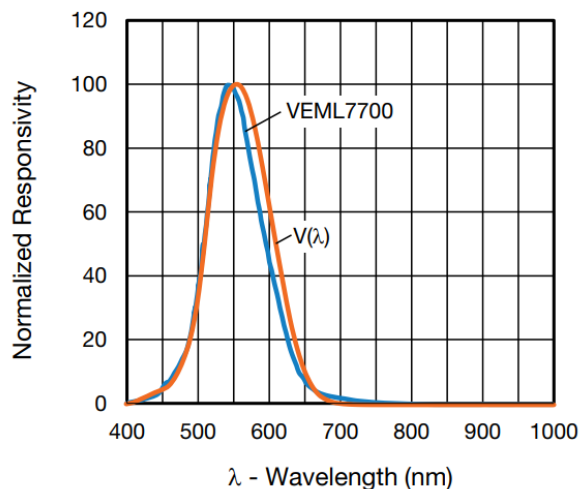


Fig. 39. Respuesta del sensor VEML7700 vs ojo humano.

- Compensación interna de temperatura.
- ADC de 16 bits para obtener medidas de 0 a 120 lux con una resolución de 0.0036 lux por cuenta.
- Voltaje de alimentación entre 2.5V y 3.6V.
- Corriente de shutdown de tan solo 0.5  $\mu$ A.

Este sensor dispone de siete registros de 16 bits cada uno, los cuales se describen a continuación:

- ALS\_CONF\_0 (0x00): En este registro se puede configura la ganancia y el tiempo de integración del sensor. Además, permite habilitar o deshabilitar las interrupciones. También se puede encender o apagar el sensor.
- ALS\_WH (0x01): Permite configurar el valor de umbral de ventana por encima para que se dispare la interrupción si está activada.
- ALS\_WL (0x02): Permite configurar el valor de umbral de ventana por debajo para que se dispare la interrupción si está activada.
- Power saving (0x03): Con los bits 2 y 1 se puede establecer uno de los cuatro modos de ahorro de energía disponibles. Con el bit 0 se habilita o deshabilita el modo de ahorro de energía.
- ALS (0x04): Contiene la lectura del sensor asociada a la luz ambiente.
- WHITE (0x05): Contiene la lectura del sensor asociada a la luz blanca.
- ALS\_INT (0x06): Solo tiene dos bits accesibles. El bit 15 se activa cuando el valor leído por el sensor sobrepasa por debajo un umbral predefinido. El bit 14 se activa cuando el valor leído por el sensor sobrepasa por encima un umbral predefinido.

## 5.2.4. Sensor de calidad del aire

El sensor CCS811 está fabricado por la compañía AMS y se utiliza para la medida de la calidad del aire. Es capaz de medir la concentración de monóxido de carbono y compuestos volátiles (VOC). A través de estas mediciones, el sensor realiza una estimación del dióxido de carbono presente en el aire.

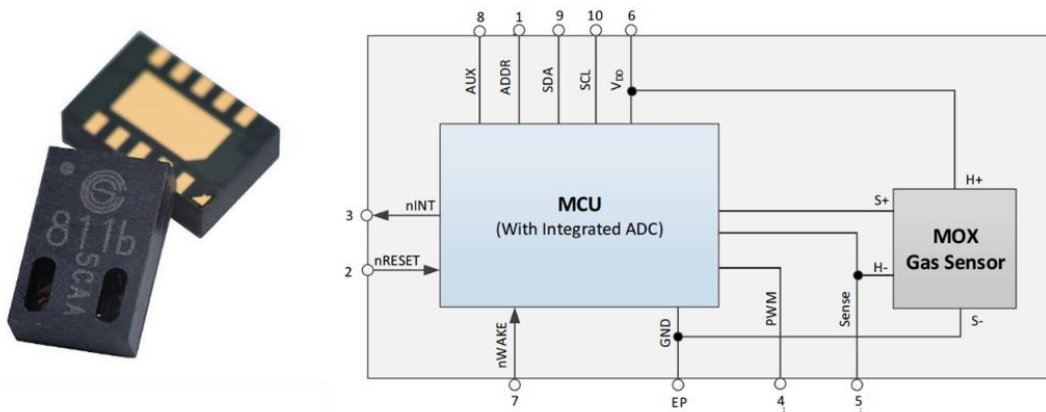


Fig. 40. Sensor CCS811.

Las principales características de este sensor son:

- La tensión de alimentación debe estar en el rango de 1.8V a 3.3V.
- Solo integra comunicación I2C con el exterior. La dirección de esclavo del bus I2C asociada a este sensor es 0x5A.



- El rango de medida de dióxido de carbono equivalente (eCO<sub>2</sub>) está comprendido entre 400ppm y 8192ppm.
- El rango de medida de compuestos volátiles totales (TVOC) está comprendido entre 0ppb y 1187ppb.
- Para realizar la medida es necesario calentar el sensor, función que se realiza internamente.
- Dispone de cinco modos de funcionamiento: modo de bajo consumo, modo de medida constante cada segundo, modo de medida cada 10 segundos, modo de medida de bajo consumo cada 60 segundos y modo de medida constante cada 250ms.

Este sensor dispone de 14 registros para propósitos de configuración o lectura de datos. En este proyecto solo se utilizan algunos de ellos, los cuales se detallan a continuación:

- Status (0x00): Con el bit DATA\_READY se indica si hay un dato disponible para leer. Este bit se borra automáticamente cuando se lee el registro de datos con la lectura del sensor.
- Measure Mode (0x01): En este registro se configura uno de los cinco modos de funcionamiento que tiene el sensor. También se configura la interrupción asociada a un pin cuando hay un dato disponible.
- Algorithm Results Register (0x02): Contiene la estimación de la medida de CO<sub>2</sub> equivalente y la medida del TVOC.
- Software Reset (0xFF): registro para realizar un reset del sensor.

### 5.2.5. Receptor GPS

Este dispositivo de la compañía Quectel es un receptor GPS compacto que lleva incluida una antena cerámica tipo parche en la parte superior como se puede ver en la Fig. 41. Sus reducidas dimensiones totales con antena integrada son de 16 mm x 16 mm x 6.45 mm hacen que sea ideal para utilizarlo en diseños en los que el tamaño de la PCB sea una restricción.

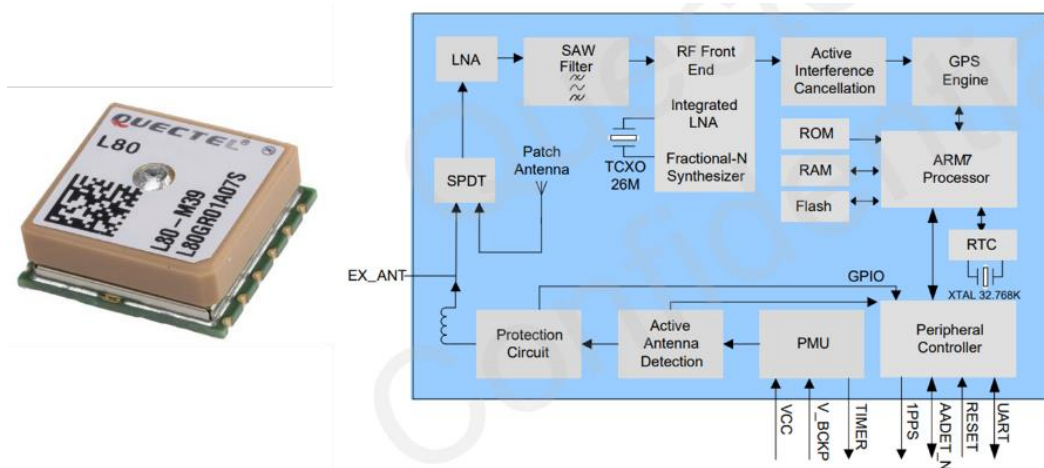


Fig. 41. Receptor GPS Quectel L80-M39.

En este proyecto se ha optado por incluir un receptor GPS para obtener la posición con precisión dónde se encuentra el sistema de monitorización. En este caso el sistema se encontrará fijo en un lugar remoto por lo que solo será necesario el envío de la posición la primera vez que se coloque en el emplazamiento elegido. En casos en los que el sistema tuviese cambios de posición, la inclusión de un receptor GPS sería un aspecto clave para saber en todo momento dónde se están realizando las



medidas. Por ejemplo, este sistema se podría ubicar en los autobuses urbanos para monitorizar la calidad ambiental de una ciudad.

Principales características de este módulo:

- Antena integrada.
- Bajo consumo.
- Alta sensibilidad.
- Dispone de 66 canales de adquisición.
- Tramas en formato NMEA.
- Utiliza la banda L1 en 1575.42 MHz.
- Interfaz de comunicación mediante UART con velocidades de transmisión desde 4800 bps a 115200 bps, por defecto está configurado a 9600 bps.
- Voltaje de alimentación de 3V a 4.3V.

La lectura de los datos que proporciona el receptor GPS se realiza por el puerto serie, para ello primero se envía un comando AT y se espera a que el módulo responda con una serie de datos.

### 5.2.6. Sensor de lluvia

El funcionamiento de este sensor es muy básico, cuando caen gotas de lluvia sobre su superficie se produce contacto entre las dos bandas de aluminio que están descubiertas por lo que se produce un cortocircuito a la salida de los dos pines que tiene. En este proyecto este sensor se trata como si fuese un interruptor. A diferencia de otros sensores que proporcionan la cantidad de agua que ha precipitado este sensor solo puede de dar información de si está lloviendo o no.



Fig. 42. Sensor de lluvia.

## 5.3. Sistema de alimentación

El sistema al completo debe ser autónomo, por lo que se alimenta mediante una batería LiPo (Lithium Polymer) de una celda, junto con un circuito de carga y protección, y para recargar la batería se utiliza un panel solar.

La batería LiPo es de una sola celda, por tanto, proporciona un voltaje nominal de 3.7 V y llega a 4.2 V cuando está cargada al completo. La diferencia entre el voltaje que proporciona la batería y el voltaje de alimentación es pequeña, teniendo que escoger un regulador de 3.3 V de bajo Dropout.

También se ha implementado un circuito de medida del voltaje de la batería que está basado en un divisor resistivo. Como la batería puede tener un voltaje máximo de 4.2 V y el ADC del microcontrolador solo admite voltajes entre 0 y 3.3 V se debe adaptar el voltaje de la batería para no

dañar el microcontrolador y ajustar las tensiones al SPAN del ADC para obtener la máxima resolución en la conversión de analógico a digital.

$$V_{IN\_ADC} = V_{BAT} \cdot \frac{R1}{R1 + R2}$$

La batería lleva incorporado un circuito de protección de sobrecarga y de descarga. Con este circuito la batería estará siempre dentro de los márgenes de seguridad para prolongar la vida de la batería o para evitar daños a la misma como puede ser una explosión.

## 5.4. Comunicación LoRa

Para proporcionar comunicaciones externas, el dispositivo integra la tecnología LoRa. Para ello se utiliza el chip SX1276 de Semtech controlado con el bus SPI desde el microcontrolador, una serie de componentes pasivos para implementar filtros y adaptación de impedancias y un conmutador de radiofrecuencia. El diseño se basa en la placa SX1276MB1MAS proporcionada por Semtech. El conmutador de radiofrecuencia PE4259 [36] está controlado por una salida digital del microcontrolador para permitir la entrada de señal al SX1276 o la salida del mismo.

Se utiliza una antena helicoidal externa adaptada a la frecuencia de 868 MHz, que se conecta a la PCB con un conector UFL. Esto permite colocarla en el lugar deseado ubicación deseada en la caja. El sistema completo de radiofrecuencia debe adaptarse a una impedancia de 50  $\Omega$ .

El diagrama de bloques de la sección de radiofrecuencia que permite establecer comunicaciones basadas en LoRa se muestra en la Fig. 43.

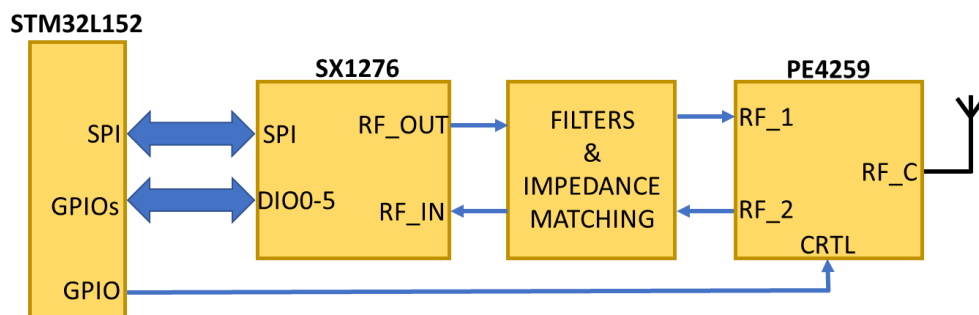


Fig. 43. Diagrama de bloques comunicación LoRa.

## 5.5. Integración de los componentes del nodo

Una vez descritos todos los dispositivos y módulos que forman el nodo se procede a integrarlos en un mismo diseño, en primer lugar, mediante un prototipo basado en tarjetas de evaluación. En la Fig. 45 se puede observar que cada sensor tiene una placa independiente y se conectan con cables a los pines de la tarjeta de evaluación NUCLEO-L152RE, donde se encuentra el microcontrolador principalmente. La comunicación LoRa se lleva a cabo utilizando la tarjeta SX1276MB1MAS que se conecta a la tarjeta principal a través de una tira de pines estándar. Esta tarjeta está basada en el chip SX1276, por lo que permite el envío de información en la banda de 433MHz o 868MHz, pero al estar en Europa la banda a utilizar es la de 868MHz. El diagrama de conexión entre los módulos que forman el prototipo se muestra en la Fig. 44, detallando los pines de la tarjeta NUCLEO-L152RE en los que se conecta cada dispositivo.

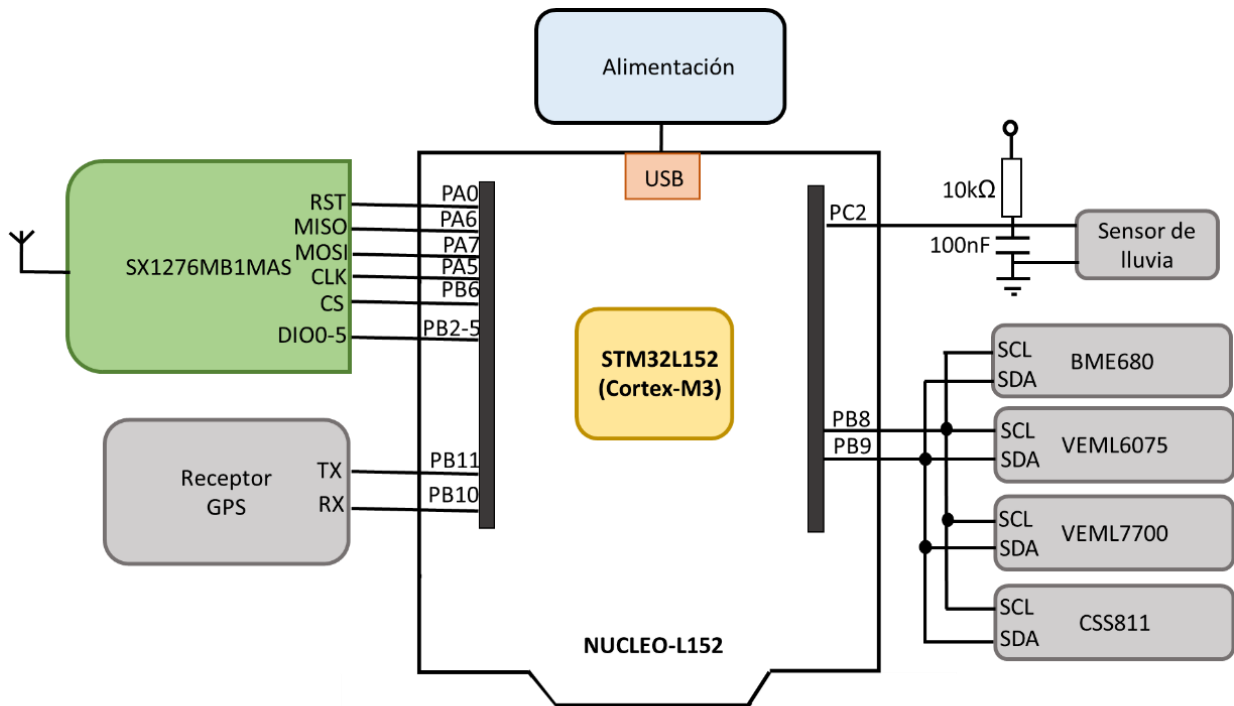


Fig. 44. Diagrama de conexionado del prototipo.

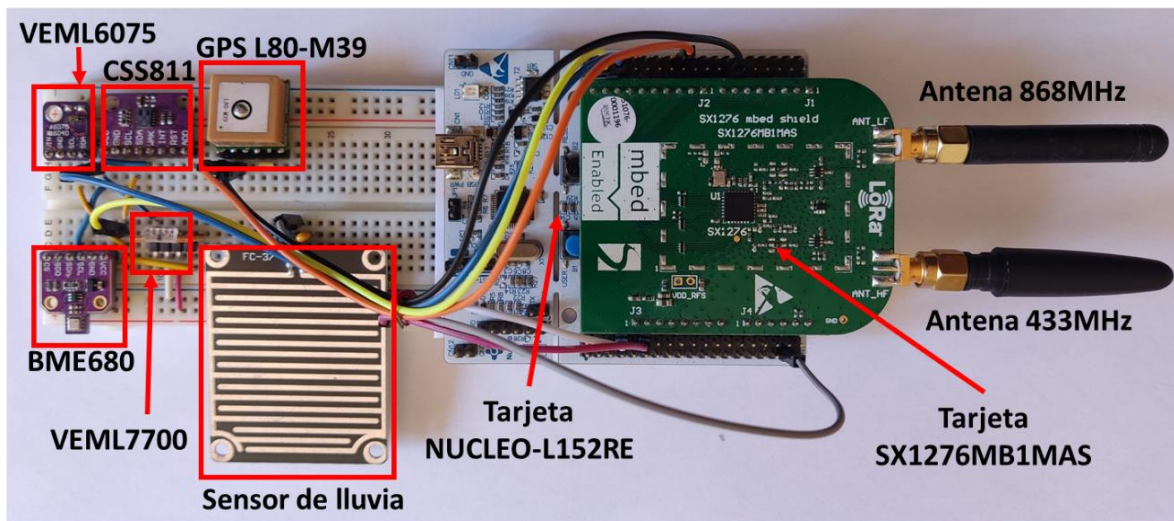


Fig. 45. Prototipo del nodo LoRa.

## 5.6. Esquema y diseño de la PCB

Para integrar todos los componentes en un solo módulo se ha diseñado una PCB utilizando el software Altium Designer. De esta forma se presenta un producto final acabado. Todo ello se ha realizado tras verificar el funcionamiento de todos los bloques que forman parte del sistema usando módulos de desarrollo e interconectándolos para formar un prototipo. La PCB se ha diseñado con dos capas y se ha utilizado como plano de masa la capa de abajo y el de alimentación de 3V3 en la capa superior. El esquema del diseño de la PCB se puede dividir en varios bloques, los cuales presentan algunas singularidades en su diseño y se comentan a continuación:

- Sistema de alimentación y carga de la batería:** El diseño se puede alimentar de dos formas, en primer lugar, por un conector micro-USB a 5V o mediante un panel solar de máximo 6.5V. El cargador de la batería dispone de leds indicadores del estado de la batería. En último lugar se utiliza el regulador de tensión para proporcionar los 3V3 a los demás elementos del sistema.

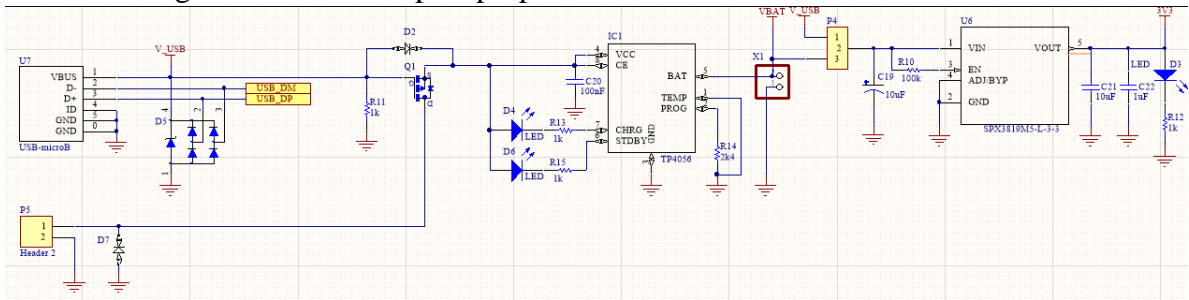


Fig. 46. Interconexión del sistema de alimentación y carga de la batería.

- Módulo de radiofrecuencia LoRa:** Es la parte del diseño que más dificultad presenta ya que es de radiofrecuencia. Hay que tener todo el sistema adaptado a la impedancia de 50 ohmios. Para ello, los componentes se colocan en la cara superior de la PCB y en la cara inferior se coloca el plano de masa, pudiendo tratar a las conexiones entre componentes como líneas microstrip.

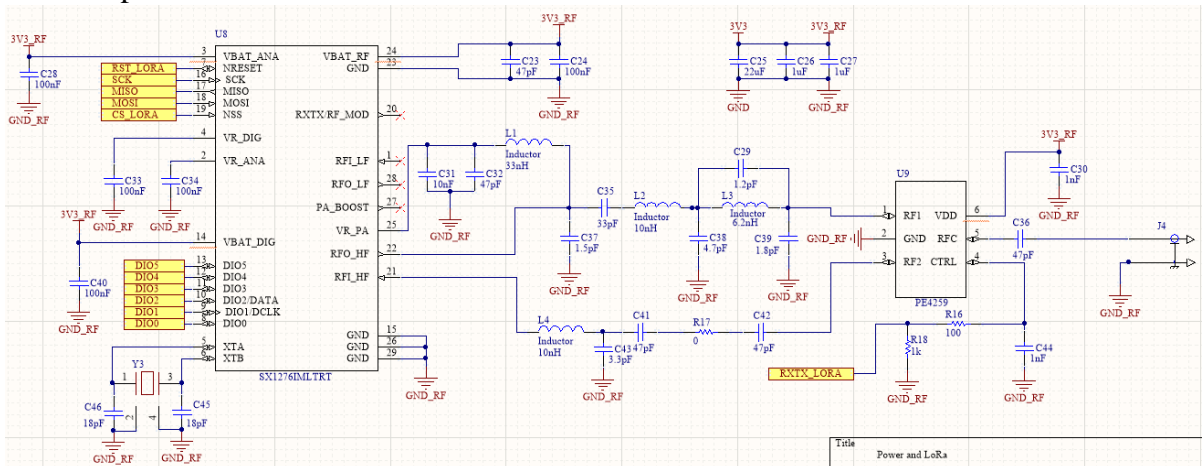


Fig. 47. Diagrama de interconexión del sistema de radiofrecuencia LoRa.

- Sistema de control y procesamiento:** Esta sección está formada por el microcontrolador de ST junto con los condensadores de desacoplo de la alimentación y los cristales para el oscilador principal y para el reloj de tiempo real. En este caso no se ha incluido pila de back-up para el RTC ya que el sistema dispone de un panel solar que permite recargar la batería principal.

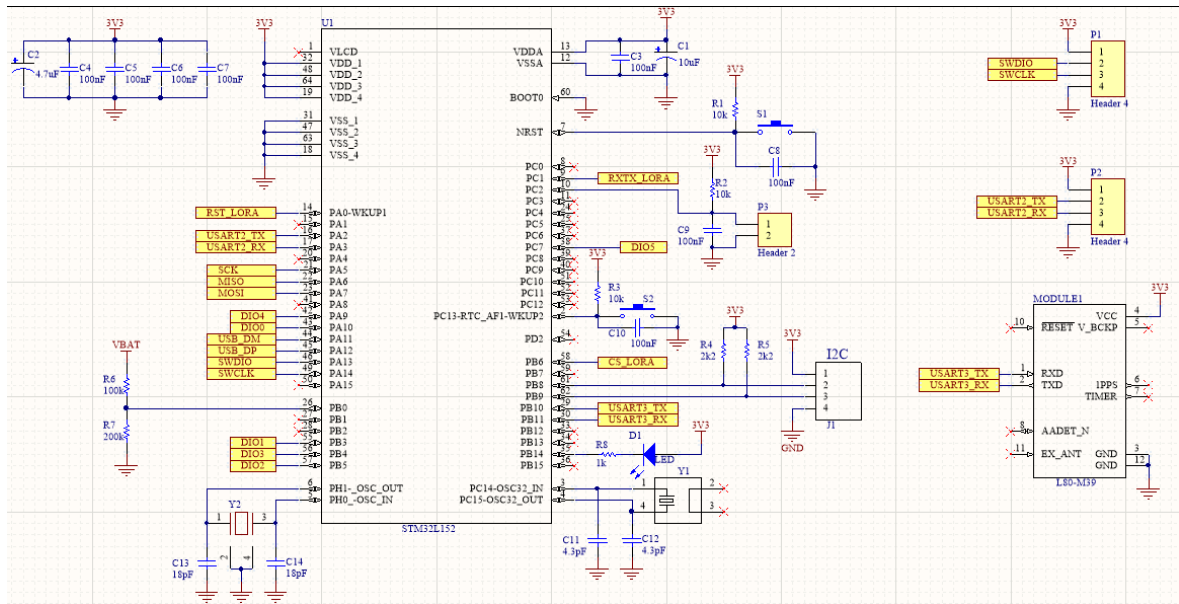


Fig. 48. Diagrama de la sección del microcontrolador.

- Sensores de medida de parámetros ambientales:** En PCBs separadas se han colocado los sensores de luminosidad y radiación UV para poder colocarlos en una posición dentro de la caja en la que tengan visión directa con la radiación solar; y en otra PCB se han colocado los sensores de partículas y el sensor de temperatura, humedad y presión atmosférica para colocarlos en un lugar de la caja donde haya ventilación de forma que se puedan realizar mediciones correctas de todos los parámetros.

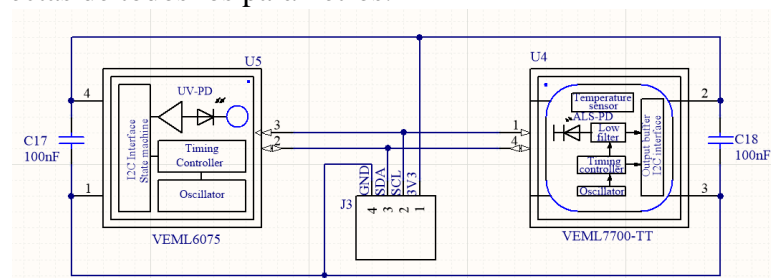


Fig. 49. Diagrama con los sensores VEML6075 y VEML7700.

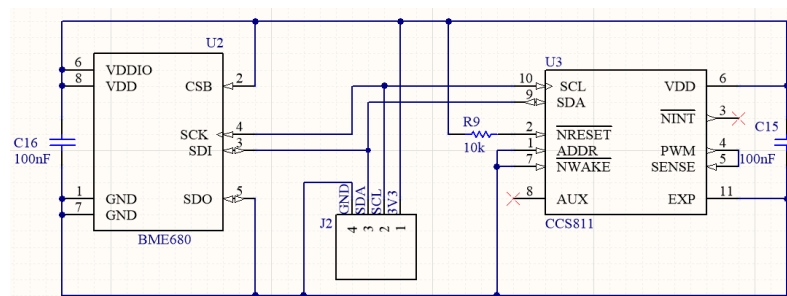


Fig. 50. Diagrama con los sensores BME680 y CCS811.

Finalmente, a partir de la captura de esquemas se integran todos los bloques descritos anteriormente en una PCB con el aspecto que se muestra en la Fig. 51. La PCB está diseñada en dos caras. En la cara superior se colocan las huellas de los componentes y se rellenan las zonas vacías con un plano de 3.3V. La cara inferior queda libre de componentes y se utiliza como plano de masa.



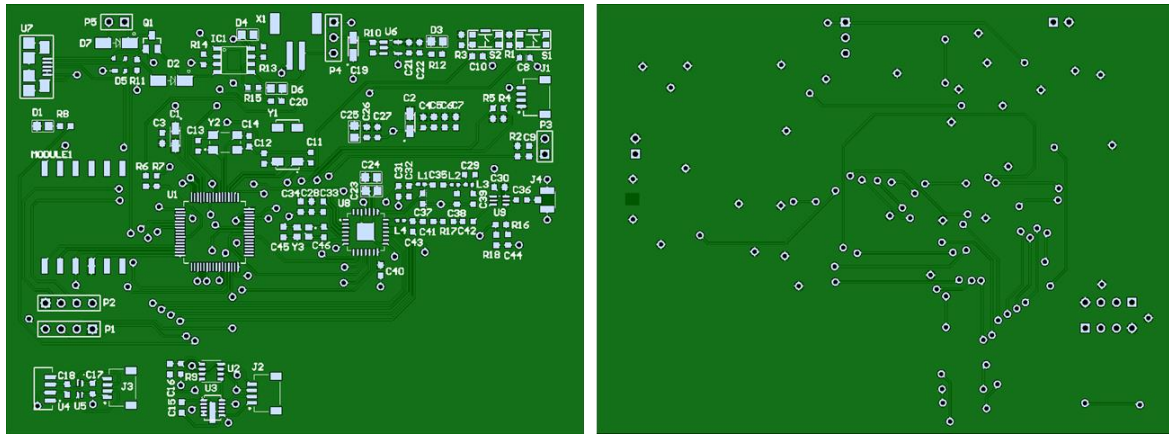


Fig. 51. Aspecto PCB, cara superior (izquierda) y cara inferior (derecha).

## 6. Diseño del software

La sección software del proyecto se puede dividir en tres partes bien diferenciadas. En primer lugar, se encuentra el software a implementar en el microcontrolador STM32L152RE para gestionar la lectura de todos los sensores y, posteriormente, enviar la información mediante una comunicación LoRaWAN. En segundo lugar, se encuentra el software instalado en el gateway Lorix One que permite la comunicación entre el nodo y el servidor de red. Por último, está todo el software instalado en la máquina virtual de Amazon Web Services, que incluye el servidor de red ChirpStack para gestionar la comunicación LoRaWAN, la base de datos InfluxDB que se encarga de almacenar los datos recibidos y el framework Flask, desde donde se despliega una página web para la visualización de los datos.

### 6.1. Software en el microcontrolador

El microcontrolador STM32L152RE es el elemento principal del nodo. Es el encargado de gestionar toda la comunicación basada en LoRaWAN y se comunica con los sensores para configurarlos y obtener las medidas de los parámetros ambientales. Para implementar toda la pila de protocolo basada en LoRaWAN se ha utilizado y modificado la librería que proporciona ST Microelectronics llamada I-CUBE-LRWAN [9]. En esta librería se implementa la capa de acceso al medio mediante el uso de dos primitivas basadas en los pares petición-confirmación e indicación-respuesta. La comunicación entre la capa LoRaMAC y capas superiores se realiza mediante las primitivas anteriores como se detalla en la Fig. 52. Por tanto, la capa de aplicación realiza solicitudes, con la primitiva “Request”, a la capa de acceso al medio LoRaMAC que esta confirma, con la primitiva “Confirm”. Mientras que la capa MAC utiliza la primitiva “Indication” para notificar un evento a la capa de aplicación y esta capa responde con la primitiva “Response”.

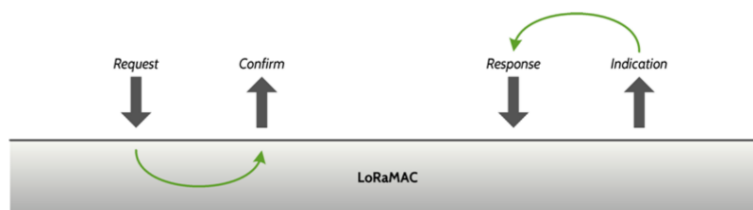


Fig. 52. Comunicación entre capas LoRaMAC [37].

Esta capa ofrece tres tipos de servicio:

- MCPS (MAC Common Part Sublayer): servicio utilizado para transmisión y recepción de datos.
- MLME (MAC Layer Management Entity): servicio para gestionar la red LoRaWAN
- MIB (MAC Information Base): servicio encargado de guardar información en tiempo de ejecución y de mantener la configuración de la capa MAC.

La librería utilizada dispone de varias capas como se muestra en el diagrama de bloque de la Fig. 53. La capa de abstracción hardware se comunica con todas las utilidades y drivers de la parte LoRa para implementar una aplicación final.

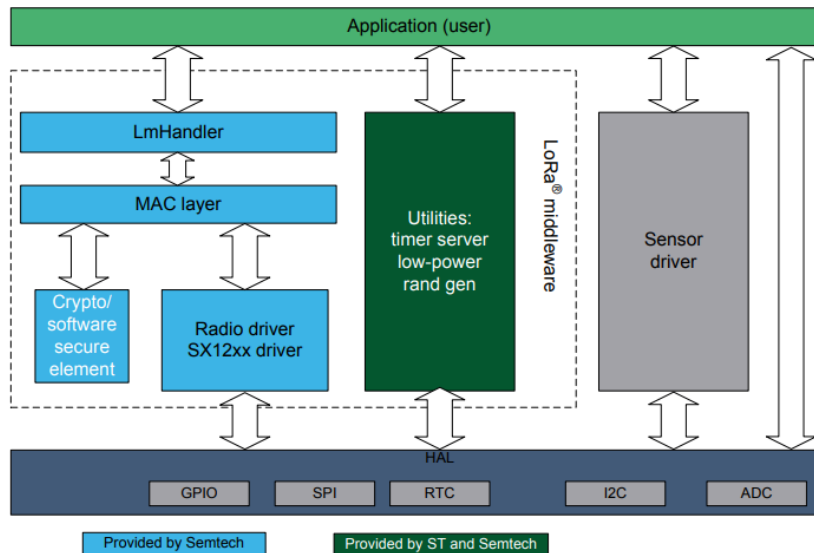


Fig. 53. Diagrama de bloques de la librería I-CUBE-LRWAN.

El entorno de desarrollo para programar y depurar el microcontrolador de ST es Keil uVision 5. La estructura del proyecto se muestra en la Fig. 54.

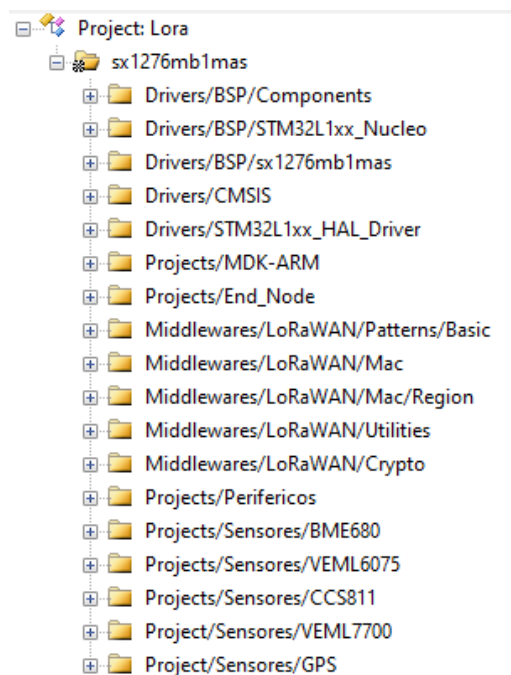


Fig. 54. Estructura del proyecto en Keil uVision5.



A continuación, se describe la funcionalidad de cada carpeta de la estructura de forma general:

- ***Driver/BSP/Components***: Incluye el driver para el dispositivo sx1276. Con este driver se implementan todas las funciones necesarias de configuración y operativa del chip que permite establecer la comunicación LoRa. Este chip se accede desde el microcontrolador principal mediante el bus SPI.
- ***Drivers/BSP/STM32L1xx\_Nucleo***: Implementa las funciones necesarias para controlar el hardware asociado a la tarjeta de desarrollo NUCLEO STM32L152RE. Por tanto, incluye funciones predefinidas para encender los leds de la placa o procesar la información de los interruptores entre otras funcionalidades.
- ***Drivers/BSP/sx1276mb1mas***: Incluye las funciones para controlar el hardware característico de la tarjeta de desarrollo SX1276MB1MAS que no incluye el driver del chip sx1276. Por ejemplo, permite configurar los pines y manejar el pin asociado al switch de radiofrecuencia que controla la salida/entrada de datos a/desde la antena.
- ***Drivers/CMSIS***: Define las funciones de inicialización del microcontrolador, referidas principalmente a la configuración de los cristales externos para obtener las frecuencias de operación del microcontrolador, tanto del reloj principal como del reloj en tiempo real.
- ***Drivers/STM32L1xx\_HAL\_Driver***: Contiene todos los ficheros con la implementación de la capa de abstracción hardware para acceder a todo los periféricos y funcionalidades del microcontrolador.
- ***Projects/MDK-ARM***: Incluye el fichero de inicio, donde se llama a la función de inicialización del sistema tras un reinicio, se definen los valores de configuración de la pila, se definen los vectores de excepción y los vectores de interrupción asociados al procesador Cortex-M3.
- ***Projects/End\_Node***: Incluye el programa principal, la configuración hardware del bus SPI, del RTC y de los GPIO y la gestión de las interrupciones.
- ***Middlewares/LoRaWAN/Patterns/Basic***: Se encuentran las funciones de la API para controlar la máquina de estados de la capa LoRa, que está basada en los servicios MCPS, MLME y MIB.
- ***Middlewares/LoRaWAN/Mac***: Contiene todas las funciones para implementar el protocolo de acceso al medio basado en LoRaWAN.
- ***Middlewares/LoRaWAN/Mac/Region***: Contiene los ficheros donde se configura los parámetros asociados a la región en la que se va a utilizar el nodo. En concreto se configura la banda de frecuencia (en este caso 868MHz), el ancho de banda, el máximo ciclo de trabajo y la potencia de transmisión para cumplir con la normativa que regula el uso de las bandas de frecuencia ISM.
- ***Middlewares/LoRaWAN/Utilities***: Implementa utilidades como la gestión de los modos de consumo de energía, gestión de las marcas de tiempo del sistema y funciones de depuración.
- ***Middlewares/LoRaWAN/Crypto***: Contiene las funciones necesarias para el encriptado y desencriptado de la información mediante AES que permiten obtener la información de las transmisiones utilizando las claves de LoRaWAN.
- ***Projects/Periféricos***: Incluyen las funciones de configuración de los periféricos necesarios para leer los datos de los sensores externos. En concreto, se utiliza el bus I2C para acceder a los datos de los sensores BME680, VEML6075, CCS811 y VEML7700; mientras que para configurar y leer los datos del receptor GPS se utiliza una UART.

- **Projects/Sensores/BME680**: Incluye el driver para configurar y leer los datos del sensor BME680.
- **Projects/Sensores/VEML6075**: Incluye el driver para configurar y leer los datos del sensor VEML6075.
- **Projects/Sensores/CCS811**: Incluye el driver para configurar y leer los datos del sensor CCS811.
- **Project/Sensores/VEML7700**: Incluye el driver para configurar y leer los datos del sensor VEML7700.
- **Project/Sensores/GPS**: Incluye el driver para configurar y leer los datos del receptor GPS. También contiene las funciones para extraer los datos de latitud y longitud de las tramas que proporciona el receptor GPS.

Una vez descrita la estructura del proyecto, se procede a describir las funciones más importantes de cada bloque.

#### Funciones de bajo nivel LoraWAN:

- *LmHandlerErrorStatus\_t LmHandlerInit (LmHandlerCallbacks\_t \*handlerCallbacks)*: Inicialización de la máquina de estados.
- *void LmHandlerJoin (ActivationType\_t mode)*: Función para realizar la petición de unión a la red según el tipo de activación que se configure, OTAA o ABP.
- *LmHandlerErrorStatus\_t LmHandlerSend (LmHandlerAppData\_t \*appData, LmHandlerMsgTypes\_t isTxConfirmed) TimerTime\_t \*nextTxIn, bool allowDelayedTx*: Función para que el nodo envíe una trama de datos y para configurar los timers dependiendo si se necesita confirmación del mensaje enviado o no.
- *int32\_t LmHandlerSetTxDataRate( int8\_t txDataRate)*: Configura el valor de la tasa de transmisión de datos.
- *int32\_t LmHandlerSetDutyCycleEnable( bool dutyCycleEnable)*: Función para configurar el ciclo de trabajo de las transmisiones de datos.
- *int32\_t LmHandlerSetTxPower( int8\_t txPower)*: Configura la potencia de transmisión del nodo.

#### Funciones de alto nivel LoraWAN:

- *void LORA\_Init (LoRaMainCallback\_t \*callbacks, LoRaParam\_t\* LoRaParam)*: Inicializa todas las primitivas de la capa LoRaMAC.
- *void LORA\_Join( void)*: Configura los identificadores y claves del nodo para solicitar unirse a la red mediante una petición de unión ya sea a través de OTAA o ABP.
- *static void LoraStartTx(TxEventType\_t EventType)*: Configura el evento con el que se inicia una transmisión. Este puede ser mediante una interrupción de un pin o a través de un timer para enviar datos de manera periódica.
- *static void Send(void \*context)*: Comprueba si el nodo está unido a la red y lee los datos de los sensores.
- *static void OnTxTimerEvent(void \*context)*: Cuando el timer llega al final de su cuenta, se vuelve a resetear y se notifica a la capa LoRaMAC para comenzar la transmisión de los datos.

**Parámetros a configurar del protocolo LoRaWAN:**

- *APP\_TX\_DUTYCYCLE*: Valor en milisegundos que transcurre entre un envío de datos y el siguiente.
- *LORAWAN\_APP\_PORT*: Puerto LoRaWAN por el que se transmiten los datos. En este proyecto se utiliza el puerto 2.
- *LORAWAN\_DEFAULT\_CLASS*: Variable con la que se configura el tipo de clase LoRaWAN. Configurada para que se utilice la clase A.
- *LORAWAN\_DEFAULT\_CONFIRM\_MSG\_STATE*: Indica si los mensajes enviados necesitan confirmación de llegada por parte del servidor de red o no la necesitan.
- *LORAWAN\_APP\_DATA\_BUFF\_SIZE*: Fija el tamaño del buffer de información que se envía.
- *OVER\_THE\_AIR\_ACTIVATION*: Configura el método de activación del nodo, eligiendo entre OTAA o ABP.

**Funciones intercambio de información sobre el bus I2C:**

- *void MX\_I2C1\_Init(void)*: Con esta función se configura el bus I2C número uno del microcontrolador a una frecuencia de 100kHz, un ciclo de trabajo del 50% y el modo de direccionamiento mediante direcciones de 7 bits.
- *void HAL\_I2C\_MspInit(I2C\_HandleTypeDef\* i2cHandle)*: Función para inicializar el reloj asociado al bus I2C y configurar los dos pines que se van a utilizar (SDA y SCL).
- *void HAL\_I2C\_MspDeInit(I2C\_HandleTypeDef\* i2cHandle)*: Función para desactivar el reloj asociado al bus I2C y desinicializar los pines SDA y SCL.

**Funciones intercambio de información sobre la UART:**

- *void UARTx\_Init(USART\_TypeDef\* USARTx, uint32\_t USART\_DIR, uint32\_t baudrate)*: Inicializa la UART que se indica, para ello configura su reloj, configura los pines de transmisión y recepción y configura la velocidad de transmisión en baudios.
- *void UARTx\_SetSpeed(USART\_TypeDef\* USARTx, uint32\_t baudrate)*: Configura la velocidad del puerto en bits por segundo.
- *void UARTx\_InitIRQ(USART\_TypeDef\* USARTx, uint32\_t USART\_IRQ, uint8\_t priority)*: Inicializa la interrupción asociada a la UART y la habilita en el NVIC.
- *void UARTx\_ConfigureDMA(USART\_TypeDef\* USARTx, uint8\_t DMA\_DIR, uint32\_t DMA\_BUF, uint8\_t \*pBuf, uint32\_t length)*: Asocia un canal del DMA al buffer de recepción de la UART.
- *void UARTx\_SetDMA(USART\_TypeDef\* USARTx, uint8\_t DMA\_DIR, FunctionalState NewState)*: Habilita o deshabilita el uso del DMA.
- *void UART\_SendBuf(USART\_TypeDef\* USARTx, char \*pBuf, uint16\_t length)*: Transmite un buffer con una longitud especificada en bytes.

**Funciones de comunicación con el sensor BME680:**

- *void bme680\_delay\_ms(uint32\_t period)*: Implementa un retardo activo del valor que se le pasa como argumento en milisegundos.
- *uint8\_t bme680\_i2c\_read(uint8\_t dev\_id, uint8\_t reg\_addr, uint8\_t \*reg\_data, uint16\_t len)*: Lee del registro “reg\_addr” del dispositivo con dirección de esclavo “dev\_id” los datos de longitud “len” y los guarda en “reg\_data”.

- *int8\_t bme680\_i2c\_write(uint8\_t dev\_id, uint8\_t reg\_addr, uint8\_t \*reg\_data, uint16\_t len)*: Escribe en el registro “reg\_addr” del dispositivo con dirección de esclavo “dev\_id” los datos guardados en “reg\_data” de longitud “len”.
- *int8\_t bme680\_begin(struct bme680\_dev \*dev)*: Función para inicializar los objetos del driver y enlazarlos a las funciones definidas anteriormente.

#### **Funciones de comunicación con el sensor CCS811:**

- *int8\_t ccs811\_init(struct ccs811\_dev \*dev)*: Inicializa el sensor comprobando que se ha leído correctamente su identificador único y posteriormente se hace un reset por software.
- *int8\_t ccs811\_set\_EnvironmentalData(uint32\_t relativeHumidity, uint32\_t temperature, struct ccs811\_dev \*dev)*: Si se disponen de los datos de temperatura y humedad relativa se le pueden enviar al sensor para mejorar la toma de medidas.
- *int8\_t ccs811\_get\_data(struct ccs811\_dev \*dev, struct ccs811\_data \*data)*: Lee los datos de los valores de CO2 y TVOC del sensor y los almacena en una estructura.
- *bool ccs811\_check\_error(struct ccs811\_dev \*dev)*: Comprueba si en el sensor se ha producido algún tipo de error mediante la lectura del byte de estado.
- *static bool isDataReady(struct ccs811\_dev \*dev)*: Comprueba si el sensor tiene un dato nuevo disponible mediante la lectura del byte de estado.

#### **Funciones de comunicación con el sensor VEML6075:**

- *static HAL\_StatusTypeDef VEML6075\_ReadRegister(I2C\_HandleTypeDef \*hi2c, uint8\_t adr, uint8\_t reg, uint16\_t \*regval)*: Función para leer los datos del registro “reg” y almacenarlos en “regval”.
- *static HAL\_StatusTypeDef VEML6075\_WriteRegister(I2C\_HandleTypeDef \*hi2c, uint8\_t adr, uint8\_t reg, uint16\_t regval)*: Función para escribir los datos guardados en “regval” en el registro “reg”.
- *HAL\_StatusTypeDef VEML6075\_WhoAmI(I2C\_HandleTypeDef \*hi2c, uint16\_t \*idval)*: Lee el identificador del sensor.
- *HAL\_StatusTypeDef VEML6075\_UVVAL(I2C\_HandleTypeDef \*hi2c, VEML6075\_PARAM\_t param, uint16\_t \*val)*: Lee los datos del sensor según se indique con el parámetro de entrada “param”. Se pueden leer los datos de la componente UVA o UVB de la radiación ultravioleta.
- *HAL\_StatusTypeDef VEML6075\_ShutDown(I2C\_HandleTypeDef \*hi2c, bool sd)*: Según el valor de “sd” el sensor entra en modo de bajo consumo o sale de este estado.
- *HAL\_StatusTypeDef VEML6075\_getUVA(I2C\_HandleTypeDef \*hi2c, float \*uva)*: Se obtiene el valor de la radiación ultravioleta del canal A después de aplicar compensaciones.
- *HAL\_StatusTypeDef VEML6075\_getUVB(I2C\_HandleTypeDef \*hi2c, float \*uvb)*: Se obtiene el valor de la radiación ultravioleta del canal B después de aplicar compensaciones.
- *HAL\_StatusTypeDef VEML6075\_calculateUVIndex(I2C\_HandleTypeDef \*hi2c, float \*uvindex)*: Función para calcular el índice ultravioleta en base a la radiación UVA y UVB.
- *HAL\_StatusTypeDef VEML6075\_getreadouts(I2C\_HandleTypeDef \*hi2c, VEML6075\_readout\_t \*reading)*: Permite almacenar en la estructura “reading” el valor de todas las medidas (UVA, UVB, IR y DARK).

### Funciones de comunicación con el sensor VEML7700:

- `void setALSConf(uint16_t conf)`: Configura la ganancia y el tiempo de integración.
- `void setPowerSaving(uint16_t ps)`: Habilita o deshabilita el modo de ahorro de energía.
- `void readRegs(uint8_t addr, uint8_t * data, int len)`: Función para leer los datos de cualquier registro del sensor.
- `void writeRegs(uint8_t * data, int len)`: Función para escribir datos en cualquier registro del sensor.

### Funciones de comunicación con el receptor GPS:

- `double grados_decimal(float grados)`: Función de conversión de grados a formato decimal utilizada para convertir los datos de longitud y latitud que proporciona el receptor GPS.
- `void Config_GPS(void)`: Inicializa el receptor GPS y lee los datos de longitud y latitud.
- `void GPS_Callback(void)`: Función para recibir a través de la interfaz UART la trama en formato NMEA que transmite el receptor GPS.
- `void GPS_data(void)`: Extrae los datos de longitud y de latitud de las tramas NMEA.

**Generación del payload a enviar:** Un aspecto importante es el formato con el que se van a enviar los datos desde el nodo hasta el servidor de red, ya que en este último se debe aplicar una función en JavaScript para extraer la información de los datos cifrados por el protocolo LoRaWAN y así poder guardarlos en el formato adecuado en la base de datos. Todos los datos, temperatura, humedad, presión atmosférica, luminosidad, radiación UVA y UVB, CO2 y TVOC se manejan en formato float, excepto el valor del sensor de lluvia que es binario y se almacena en un byte. La transmisión se debe hacer en bytes por lo que hay que convertir los datos en formato float a enteros de 32 bits, para ello se multiplica el valor del sensor por la potencia de 10 necesaria para dejar al número sin decimales y no perder información. Por ejemplo, en el caso de valor de temperatura solo interesan 2 decimales por lo que hay que multiplicar el valor del sensor por 100, véase Fig. 55. Sin embargo, los valores de latitud y longitud tienen 7 decimales de interés y hay que multiplicar por  $10^7$ , véase Fig. 56. El dato del sensor de lluvia se formatea directamente con un byte, véase Fig. 57. Por lo tanto, por cada transmisión se enviarán 33 bytes de carga útil.

```
int32_t temperatura = (int32_t)temperatura_float*100;
AppData.Buff[i++] = temperatura >> 24;
AppData.Buff[i++] = temperatura >> 16;
AppData.Buff[i++] = temperatura >> 8;
AppData.Buff[i++] = temperatura;
```

Fig. 55. Código para dar formato al valor de la temperatura.

```
int32_t latitud = (int32_t)latitud_float*10000000;
AppData.Buff[i++] = latitud >> 24;
AppData.Buff[i++] = latitud >> 16;
AppData.Buff[i++] = latitud >> 8;
AppData.Buff[i++] = latitud;
```

Fig. 56. Código para dar formato al valor de la latitud.

```
int8_t lluvia = lluvia_on_off;
AppData.Buff[i++] = lluvia;
```

Fig. 57. Código para dar formato al valor del sensor de lluvia.

**Máquina de estados del programa:** La máquina de estados que gobierna todo el programa se basa en el esquema de la Fig. 58. Después de producirse un reset en el sistema, el nodo pasa a un estado llamado “Inicialización” donde se inicializa el sistema y la capa de bajo nivel de la comunicación LoRaWAN. En el estado “Configuración de sensores” se configuran los parámetros asociados a los sensores ambientales y al receptor GPS. El nodo está configurado para activarse vía OTAA y envía una petición de unirse a la red pasa a un estado de reposo llamado “Sleep”. Cuando ocurre un evento y el nodo tiene que enviar datos, se despierta saliendo del estado “Reposo” y pasa al estado “Lectura de sensores” donde se realiza la adquisición de los parámetros ambientales que proporciona cada uno de los sensores. El nodo comprueba que está unido correctamente a la red, para ir al estado “Envío de datos LoRa” donde envía los datos y vuelve otra vez al estado de reposo. Si el nodo no está unido a la red, debe intentar unirse pasando al estado “Join”.

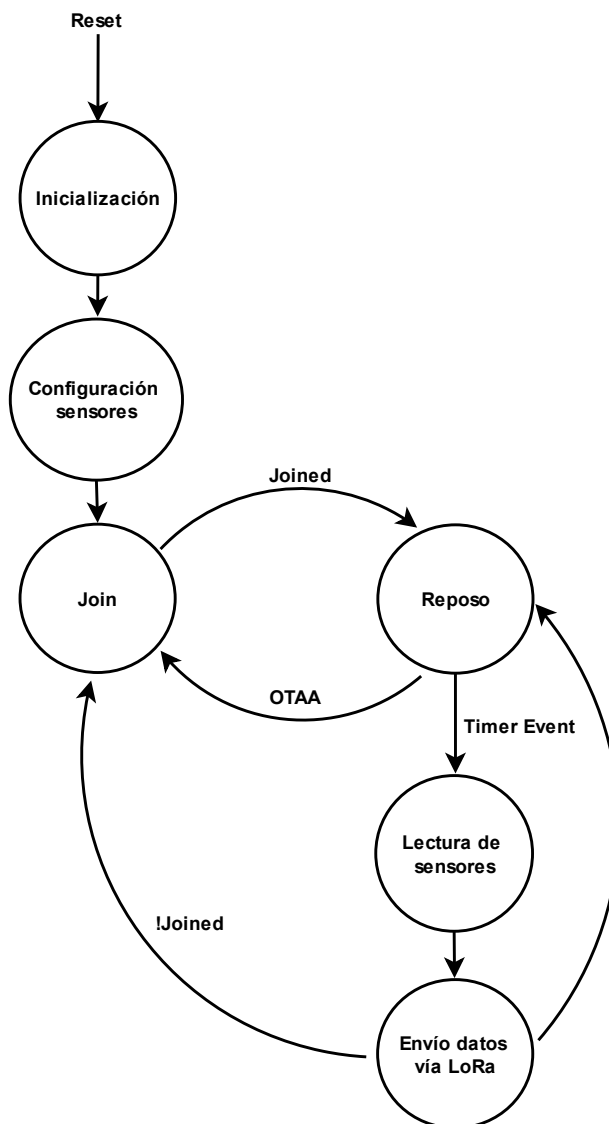


Fig. 58. Máquina de estados.

## 6.2. Software de configuración del gateway

El gateway Lorix One tiene instalado un programa llamado “packet forwarder” que es el encargado de reenviar los paquetes que recibe el concentrador de radiofrecuencia al servidor de red mediante una conexión UDP (User Datagram Protocol). También se encarga de transmitir los paquetes que

recibe desde el servidor de red hacia el nodo final. Si el gateway tiene instalado un módulo GPS, este programa también tiene la funcionalidad de sincronización de los nodos de una red LoRaWAN.

En el enlace de uplink, el gateway retransmite la información de los nodos a la que agrega metadatos e incluso puede añadir información del estado del sistema del gateway. En el enlace descendente, el gateway también puede incluir metadatos que agrega a los paquetes que recibe del servidor, y también puede recibir información de configuración del propio gateway desde el servidor de red.

En el gateway Lorix One el “packet forwarder” que hay instalado es el de Semtech que se comunica con el servidor de red mediante UDP. Este reenviador de paquetes fue el primero que se utilizó y en la actualidad hay otros que funcionan con distintas tecnologías. El “packet forwarder” de Semtech es simple y se basa en el intercambio de mensajes entre gateway y servidor de red con un formato parecido a JSON (JavaScript Object Notation) utilizando UDP.

En la Fig. 59 se puede ver la arquitectura de un gateway, donde se reciben la información del nodo a través de un concentrador que se comunica con una interfaz SPI con el microcontrolador del gateway en el que se instala el packet forwarder y se reenvían los mensajes implementando la pila UDP hacia el servidor.

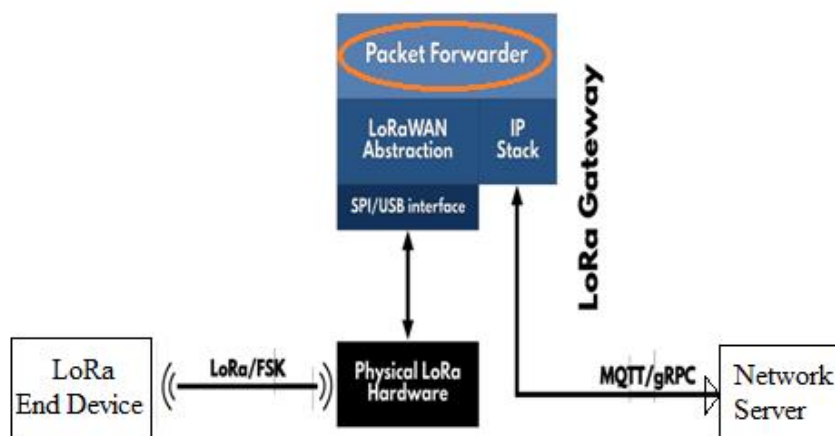


Fig. 59. Estructura gateway [38].

Para configurar el packet forwarder que viene instalado en el gateway Lorix One es necesario acceder al sistema mediante SSH o con un cable USB. En primer lugar, se debe realizar una configuración de red, para ello se modifica el fichero “/etc/network/interface” en el que se habilita DHCP y se cambia la dirección IP del gateway a una que previamente ha sido reservada en el router. Lorix One lleva incorporado de fábrica la opción de elegir el tipo de reenviador de paquetes a utilizar, esto se realiza con el fichero administrador de la nube ubicado en “/etc/init.d/clouds-manager.sh”, en el que se configura que se utilice el packet forwarder de Semtech. El siguiente paso es modificar los ficheros asociados al reenviador de paquetes a utilizar los cuales se encuentran en “/opt/lorix/clouds/packet-forwarder/”, para ello se modifica el fichero “local\_conf.json” en el que se especifica el identificador del gateway, la dirección IP del servidor donde se va a instalar ChirpStack (dirección IP pública proporcionada por AWS) y los puertos por donde se envían los mensajes de uplink y downlink, siendo el puerto 1700 para los dos propósitos. También se puede especificar la posición GPS del gateway ya que el Lorix One no dispone de receptor GPS para calcular su posición automáticamente. Y por último se puede configurar un email de contacto.

## 6.3. Software en AWS

Se utiliza Amazon Web Services para instalar una máquina virtual que procese todos los datos que recibe del nodo. En concreto, dentro de la máquina virtual se instala ChirpStack para gestionar las comunicaciones LoRaWAN, InfluxDB para almacenar los datos de interés y el framework Flask para desplegar la página web que permite la visualización de los datos.

En primer lugar, hay que crear una cuenta de usuario en Amazon Web Services. Después se selecciona el servicio web Amazon EC2 (Amazon Elastic Compute Cloud) donde se instala la máquina virtual. En este caso, se ha elegido Ubuntu Server 18.04 LTS, como se muestra en la Fig. 60.

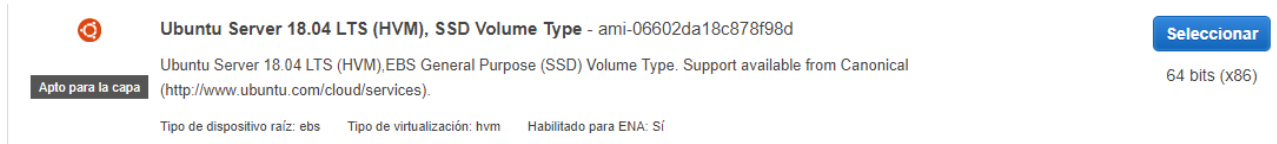


Fig. 60. Características del sistema operativo instalado en AWS.

Posteriormente, se elige el tipo de instancia. Hay varias opciones según el rendimiento que se le vaya a dar a la máquina virtual en cuanto a recursos se refiere. Se elige el tipo “t2.micro” que incluye 1 CPU y 1GB de memoria RAM, permitiendo escoger la capa gratuita que ofrece durante 12 meses de duración.

En la parte del servidor se ha instalado una máquina virtual en Amazon Web Services con el sistema operativo Ubuntu 18.04, una memoria RAM de 1GB y un SSD de 8GB. Con estas características es suficiente para instalar los componentes necesarios y además AWS ofrece un año de capa gratis para esta configuración. A esta opción se le puede añadir un almacenamiento de 8GB en disco SSD.

	Familia	Tipo	vCPU	Memoria ( GiB)	Almacenamiento de la instancia (GB)	Optimizado para EBS disponible	Desempeño de la red	Compatibilidad con IPv6
<input type="checkbox"/>	t2	t2.nano	1	0.5	EBS solo	-	De bajo a moderado	Sí
<input checked="" type="checkbox"/>	t2	t2.micro Apto para la capa gratuita	1	1	EBS solo	-	De bajo a moderado	Sí
<input type="checkbox"/>	t2	t2.small	1	2	EBS solo	-	De bajo a moderado	Sí
<input type="checkbox"/>	t2	t2.medium	2	4	EBS solo	-	De bajo a moderado	Sí
<input type="checkbox"/>	t2	t2.large	2	8	EBS solo	-	De bajo a moderado	Sí
<input type="checkbox"/>	t2	t2.xlarge	4	16	EBS solo	-	Moderada	Sí
<input type="checkbox"/>	t2	t2.2xlarge	8	32	EBS solo	-	Moderada	Sí

Fig. 61. Tipos de configuraciones para la máquina virtual.

Una vez configurada toda la instancia para acceder a ella se utiliza el protocolo SSH (Secure Shell), que permite acceder de forma remota a servidores mediante una conexión cifrada. También es necesario el uso de un archivo de claves con extensión “.pem” donde se almacena la clave privada para acceder a la instancia, mientras que la clave pública está almacenada en la instancia.

En la instancia es necesario configurar los grupos de seguridad que son una serie de reglas que permiten filtrar el tráfico entrante y saliente de la instancia por motivos de seguridad. Para la aplicación a desarrollar es necesario configurar las siguientes reglas:

- Puerto UDP 1700 de salida para enviar datos desde el servidor de red al gateway.
- Puerto UDP 1700 de entrada para enviar datos desde el gateway al servidor de red.
- Puerto TCP 22 para permitir la comunicación desde el exterior con SSH.
- Puerto TCP 5000 de entrada para visualizar la página web desplegada con Flask.



- Puerto TCP 8080 de entrada para tener acceso a la web de configuración de ChirpStack.

En este punto, ya está instalada la máquina virtual en la nube y ahora es necesario instalar los componentes que se van a utilizar en el proyecto: ChirpStack, InfluxDB y Flask.

## 6.3.1. ChirpStack

### 6.3.1.1. Introducción

ChirpStack es un conjunto de módulos software que permiten implementar la pila de comunicaciones LoRaWAN en un servidor privado. Los componentes que forman ChirpStack y la conexión entre ellos se describe en el diagrama de bloque de la Fig. 62.

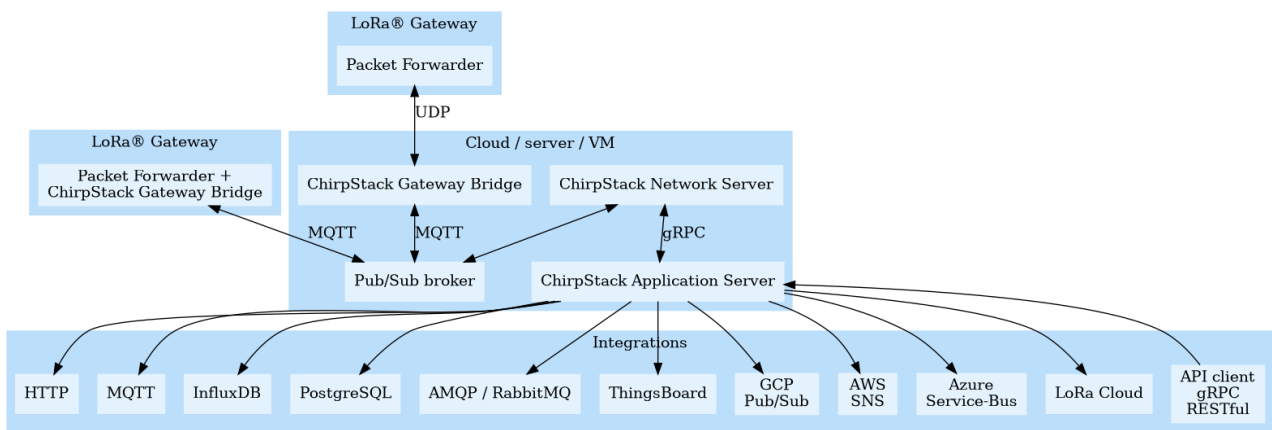


Fig. 62. Arquitectura de ChirpStack.

El primer elemento de la arquitectura es el Gateway Bridge que se emplaza entre el gateway y el bróker MQTT. Puede estar instalado directamente en el gateway o en el servidor, siendo esta última opción la elegida en este proyecto. Su función principal es enviar la información recibida al bróker para que se pueda acceder a ella desde el servidor de red.

En segundo lugar, se encuentra el servidor de red. Este módulo es el encargado de recibir/enviar la información de/hacia los Gateways, gestión de la autenticación de los dispositivos, gestión de la capa de acceso al medio del protocolo LoRaWAN, programación del envío de los mensajes de downlink y de establecer comunicación bidireccional con el servidor de aplicación. El servidor de red ChirpStack lleva incorporadas opciones de integración con HTTP, MQTT o bases de datos, es común que los desarrolladores quieran hacer aplicaciones a medida bajo sus propios patrones de diseño. Para ello, se puede comunicar el servidor de red con otro servidor para el envío de los datos por peticiones HTTP para que posteriormente se guarden esos datos en una base de datos y después se hagan consultas para extraer los datos y mostrarlos al usuario mediante una interfaz gráfica en una página web. Además, se pueden guardar la información directamente en bases de datos como InfluxDB o PostgreSQL. Por último, también ofrece integración con paneles de visualización de variables como ThingsBoard, lo que simplifica la visualización de la información ya que solo es necesario introducir la información del panel de ThingsBoard en ChirpStack y el formato adecuado de las tramas de datos.

En tercer lugar, se encuentra el servidor de aplicación, que mediante una interfaz web permite gestionar los usuarios, organizaciones, aplicaciones y dispositivos. Con ello se aísla cada nodo a diferentes niveles. Es el encargado de enviar los datos a las diferentes opciones de integración que ofrece ChirpStack. Además, se encarga de recibir los mensajes de downlink a través de la API. También gestiona los mensajes de petición de unión a la red (Join-Request), ya que es el encargado

de guardar la AppKey de cada dispositivo para el caso de activaciones OTAA. Otra funcionalidad del servidor de aplicación es el cifrado/descifrado de los payloads de aplicación cuando se envían/reciben de las distintas opciones de integración. Por último, dispone de una API gRPC para propósitos de integración, incluso se puede utilizar otra API REST HTTP, la cual se traduce a gRPC internamente según se describe en el diagrama de la Fig. 63.

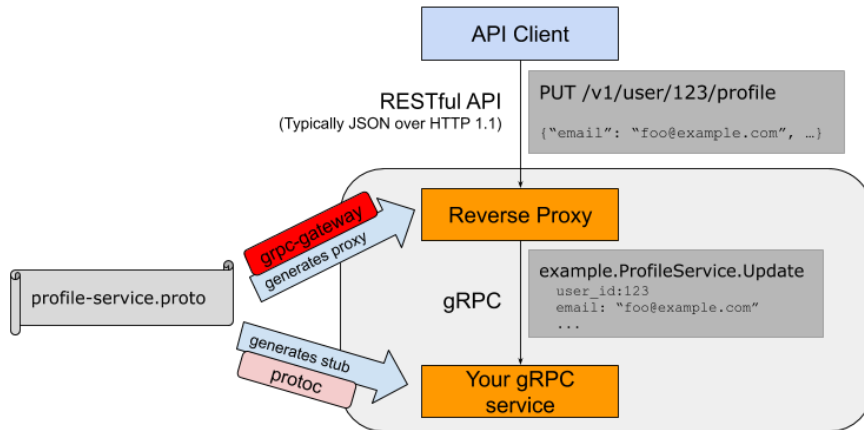


Fig. 63. Diagrama del proceso de traducción de HTTP a gRPC [40].

Para finalizar, se encuentran las integraciones que dispone ChirpStack, que se muestran en la Fig. 64, las cuales permiten comunicación con una aplicación exterior para el envío de datos o para recibir datos.

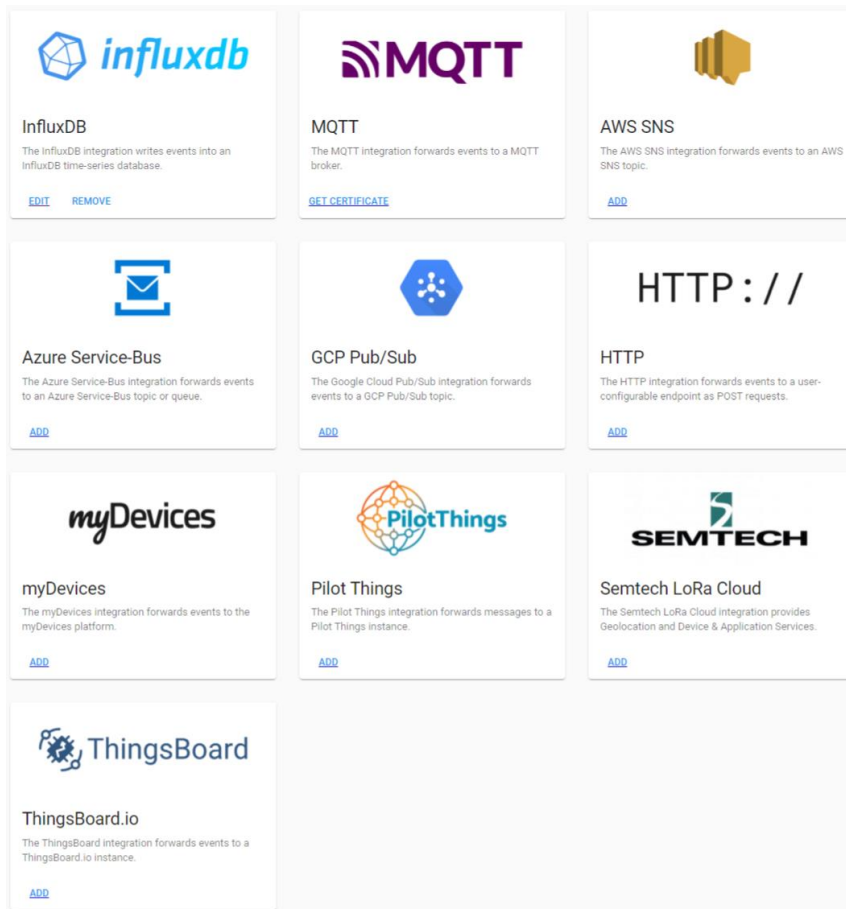


Fig. 64. Integraciones disponibles en ChirpStack.

### 6.3.1.2. Registro de un gateway

Tras la instalación de ChirpStack se puede gestionar la aplicación desde una página web. En primer lugar, hay que añadir el gateway Lorix One al sistema para poder recibir los datos que envía el nodo. Para ello hay que seguir una serie de pasos:

- Primero hay que definir la organización. En este caso se le ha dado el nombre “chirpstack”.

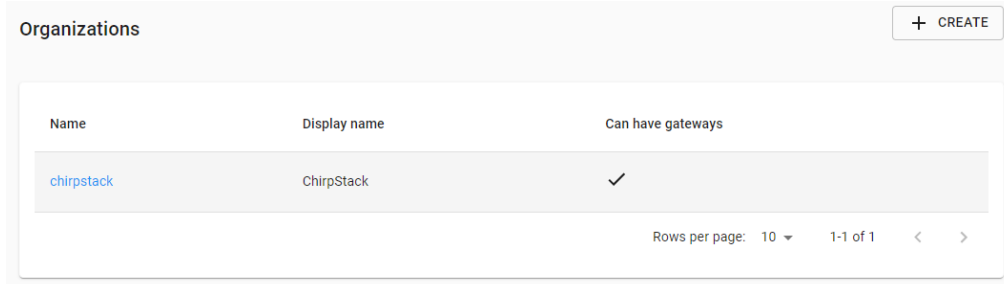


Fig. 65. Organización en ChirpStack.

- En segundo lugar, hay que añadir un servidor de red, asignándole un nombre “TFM” e indicando la dirección IP desde la que es accesible.

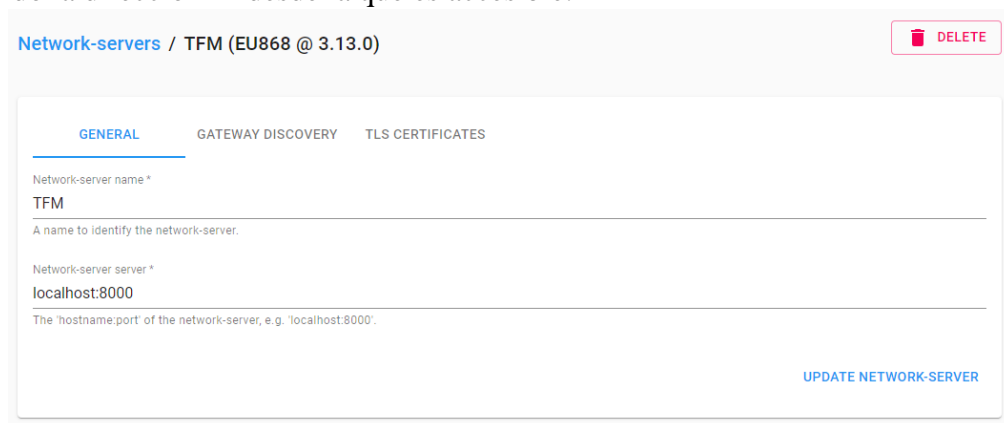


Fig. 66. Crear servidor de red en Chirpstack.

- Después, se define un perfil para el gateway, donde se configura el intervalo de tiempo en el que el gateway envía estadísticas al servidor de red de forma periódica.

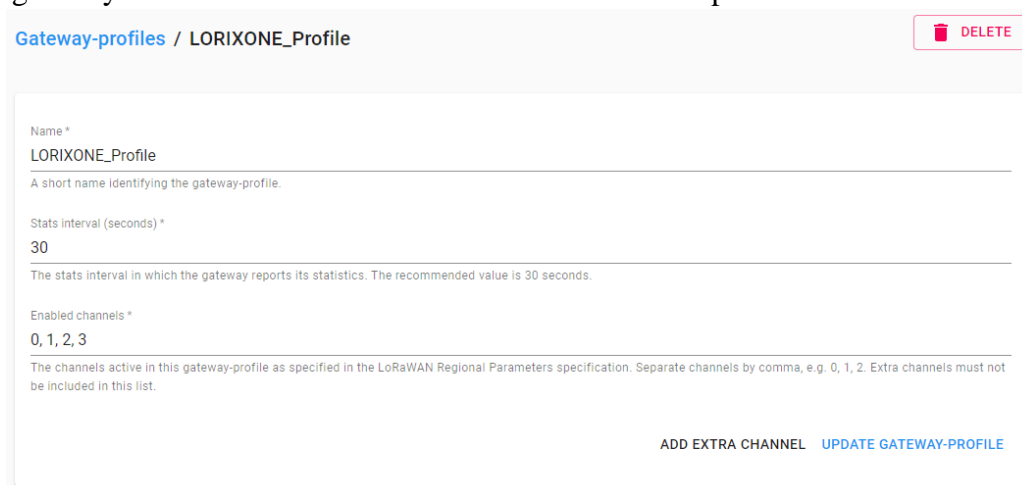


Fig. 67. Crear perfil de gateway en ChirpStack.

- Definición de un perfil de servicio con nombre “TFM-service”, donde se configura si el gateway puede introducir metadatos en los paquetes que reenvía. También se configura la máxima tasa de transmisión permitida para los dispositivos de la red.

The screenshot shows the 'Service-profiles / TFM-service' configuration page. At the top right, there is a 'DELETE' button with a trash icon. The main form contains the following fields and options:

- Service-profile name \***: TFM-service. A name to identify the service-profile.
- Add gateway meta-data**: GW metadata (RSSI, SNR, GW geoloc., etc.) are added to the packet sent to the application-server.
- Enable network geolocation**: When enabled, the network-server will try to resolve the location of the devices under this service-profile. Please note that you need to have gateways supporting the fine-timestamp feature and that the network-server needs to be configured in order to provide geolocation support.
- Device-status request frequency**: 0. Frequency to initiate an End-Device status request (request/day). Set to 0 to disable.
- Minimum allowed data-rate \***: 0. Minimum allowed data rate. Used for ADR.
- Maximum allowed data-rate \***: 5. Maximum allowed data rate. Used for ADR.
- Private gateways**: Gateways under this service-profile are private. This means that these gateways can only be used by devices under the same service-profile.

At the bottom right of the form, there is a blue button labeled 'UPDATE SERVICE-PROFILE'.

Fig. 68. Crear perfil de servicio en ChirpStack.

- En este punto, ya se puede incluir la información del gateway a la red. Donde se configura su identificador y se asocia a los perfiles creados anteriormente. Además, como el gateway Lorix One no lleva incluido un receptor GPS, se indica su altitud y coordenadas.

**Gateways / Create**

**GENERAL** TAGS METADATA

Gateway name \*  
**LORIXONE**  
The name may only contain words, numbers and dashes.

Gateway description \*  
 Gateway Lorix One UAH

---

Gateway ID \*  
**FC C2 3D FF FE 0F 62 25** MSB ↻

Network-server \*  
**TFM**  
Select the network-server to which the gateway will connect. When no network-servers are available in the dropdown, make sure a service-profile exists for this organization.

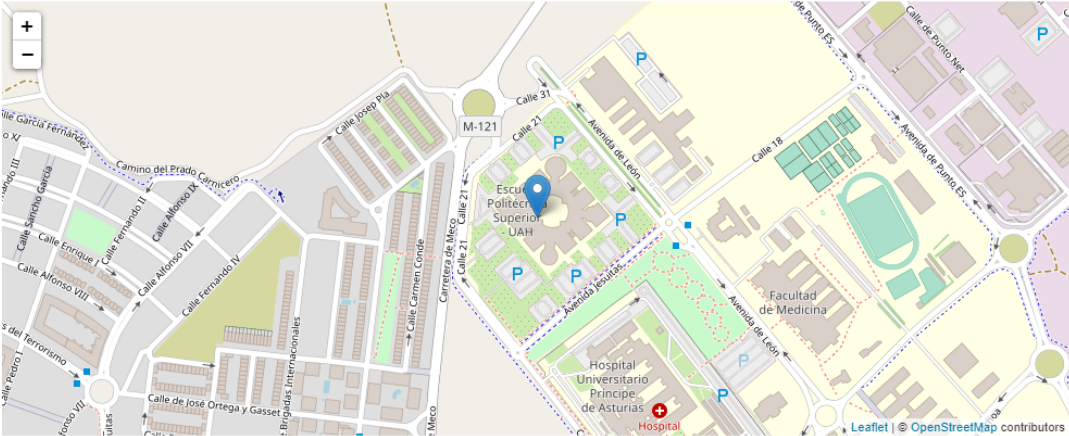
Service-profile  
**TFM-service**  
Select the service-profile under which the gateway must be added. The available service-profiles depend on the selected network-server, which must be selected first.

Gateway-profile  
**LORIXONE\_Profile**  
Optional. When assigning a gateway-profile to the gateway, ChirpStack Network Server will attempt to update the gateway according to the gateway-profile. Note that this does require a gateway with ChirpStack Concentrator.

**Gateway discovery enabled**  
When enabled (and ChirpStack Network Server is configured with the gateway discover feature enabled), the gateway will send out periodical pings to test its coverage by other gateways in the same network.

Gateway altitude (meters) \*  
**600**  
When the gateway has an on-board GPS, this value will be set automatically when the network has received statistics from the gateway.

Gateway location ([set to current location](#))



Drag the marker to the location of the gateway. When the gateway has an on-board GPS, this value will be set automatically when the network receives statistics from the gateway.

**ADD BOARD CONFIGURATION** **CREATE GATEWAY**

Fig. 69. Crear gateway en ChirpStack.

### 6.3.1.3. Registro de un dispositivo

Pasos para registrar el nodo en ChirpStack:

- Definición del perfil de dispositivos. Este perfil es común a un grupo de dispositivos con las mismas características de conexión y autenticación. Se le asigna el nombre “TFM-device-profile”, se configura la versión de LoRaWAN 1.0.2 con la revisión A. En la pestaña de JOIN” se indica que el nodo se va a activar mediante OTAA. En la sección de “CODEC” se implementa el codificador de los mensajes que recibe el servidor de red, implementado como una función JavaScript.

The screenshot shows the 'Device-profiles / TFM-device-profile' page in ChirpStack. It features a 'DELETE' button in the top right corner. Below the title, there are tabs for 'GENERAL', 'JOIN (OTAA / ABP)', 'CLASS-B', 'CLASS-C', 'CODEC', and 'TAGS'. The 'GENERAL' tab is active and contains the following fields:

- Device-profile name \***: TFM-device-profile (with a note: "A name to identify the device-profile.")
- LoRaWAN MAC version \***: 1.0.2 (with a note: "The LoRaWAN MAC version supported by the device.")
- LoRaWAN Regional Parameters revision \***: A (with a note: "Revision of the Regional Parameters specification supported by the device.")
- ADR algorithm \***: Default ADR algorithm (with a note: "The ADR algorithm that will be used for controlling the device data-rate.")
- Max EIRP \***: 0 (with a note: "Maximum EIRP supported by the device.")
- Uplink interval (seconds) \***: 10 (with a note: "The expected interval in seconds in which the device sends uplink messages. This is used to determine if a device is active or inactive.")

An 'UPDATE DEVICE-PROFILE' button is located at the bottom right of the form.

Fig. 70. Perfil de dispositivo en ChirpStack.

- Crear una aplicación que agrupa a dispositivos que comparten funcionalidad más específica. Se asigna el nombre “TFM-app” y una descripción.

The screenshot shows the 'Applications / TFM-app' page in ChirpStack. It features a 'DELETE' button in the top right corner. Below the title, there are tabs for 'DEVICES', 'APPLICATION CONFIGURATION', and 'INTEGRATIONS'. The 'APPLICATION CONFIGURATION' tab is active and contains the following fields:

- Application name \***: TFM-app (with a note: "The name may only contain words, numbers and dashes.")
- Application description \***: Application TFM

A note below the description field states: "Note: The payload codec fields have moved to the device-profile." An 'UPDATE APPLICATION' button is located at the bottom right of the form.

Fig. 71. Crear una aplicación en ChirpStack.

En la pestaña “INTEGRATIONS” se configuración la integración a la que se envían los datos recibidos por el servidor de red tras ser decodificados por el códec. En este caso se elige la base de datos InfluxDB en la que se especifica la dirección y puerto donde está instalada la

base de datos para escribir la información, el nombre de usuario y la contraseña, el nombre de la base de datos y la precisión de la marca de tiempo, configurada al valor máximo.

The screenshot shows the 'Update InfluxDB integration' configuration page. At the top, there are tabs for 'DEVICES', 'APPLICATION CONFIGURATION', and 'INTEGRATIONS'. The 'INTEGRATIONS' tab is active. The page title is 'Update InfluxDB integration'. Below the title, there are several input fields: 'API endpoint (write) \*' with the value 'http://localhost:8086/write', 'Username' with the value 'admin', 'Password' with masked characters '....', 'Database name \*' with the value 'chirpstack', 'Retention policy name' (empty), and 'Timestamp precision \*' with a dropdown menu set to 'Microsecond'. A note below the timestamp precision field states: 'It is recommended to use the least precise precision possible as this can result in significant improvements in compression.' At the bottom right, there is a blue button labeled 'UPDATE INTEGRATION'.

Fig. 72. Configuración de la integración con InfluxDB.

- Añadir el nodo a la aplicación donde se especifica el nombre y una descripción. Se le asigna la clave "devEU" y se asigna el dispositivo a un perfil de dispositivos creado anteriormente. Una vez creado se deben configurar parámetros adicionales como la clave de aplicación necesaria para la activación vía OTAA.

The screenshot shows the 'Create Device' page. At the top, there are tabs for 'GENERAL', 'VARIABLES', and 'TAGS'. The 'GENERAL' tab is active. The page title is 'Applications / TFM-app / Devices / Create'. Below the title, there are several input fields: 'Device name \*' with the value 'Nodo-TFM', 'Device description \*' with the value 'Nodo LoRaWAN', 'Device EUI \*' with the value 'ed 7b b5 8a 9c 74 58 54' and a refresh button, and 'Device-profile \*' with a dropdown menu set to 'TFM-device-profile'. There are two checkboxes: 'Disable frame-counter validation' and 'Device is disabled'. A note below the first checkbox states: 'Note that disabling the frame-counter validation will compromise security as it enables people to perform replay-attacks.' At the bottom right, there is a blue button labeled 'CREATE DEVICE'.

Fig. 73. Crear dispositivo en ChirpStack.

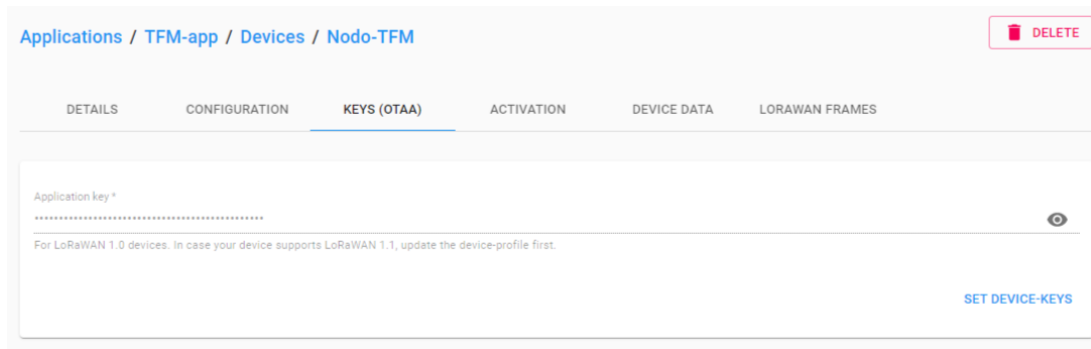


Fig. 74. Configuración de la clave de aplicación.

#### 6.3.1.4. Configuración del códec

En el perfil de dispositivos se encuentra la opción de implementar un decodificador a los datos que llegan cifrados por la comunicación LoRaWAN. Para ello, es necesario implementar una función escrita en JavaScript que permita extraer la información útil enviada por el nodo. En la variable de entrada “bytes” se encuentra toda la carga útil recibida por el servidor de red tras ser descriptada. Como en emisión los datos se han codificado eliminando la parte decimal para convertirlos a números enteros sin perder información, en decodificación hay que realizar el procedimiento a la inversa para recuperar el dato real. Por simplicidad, en la Fig. 75 se muestra la decodificación de los datos de temperatura, longitud y lluvia, los demás datos se decodifican de igual manera.

```
function Decode(bytes) {
  var index = 0;
  var temperatura = Number(((byte[index++] << 24 | byte[index++] << 16 |
    byte[index++] << 8 | byte[index++]) * 1e-2).toFixed(2));

  var longitud = Number(((byte[index++] << 24 | byte[index++] << 16 |
    byte[index++] << 8 | byte[index++]) * 1e-7).toFixed(7));
  ...
  var lluvia = Number(byte[index++]);
  ...
  var appData = {'temperatura': temperatura, 'humedad': humedad,
    'presion': presion, 'luminosidad': luminosidad,
    'uva': uva, 'uvb': uvb, 'co2': co2,
    'tvoc': tvoc, 'lluvia': lluvia,
    'latitud': latitud, 'longitud': longitud};
  return appData;
}
```

Fig. 75. Función para decodificar los datos que llegan a ChirpStack.

### 6.3.2. Base de datos InfluxDB

InfluxDB es una base de datos creada por la empresa InfluxData de código abierto bajo licencia MIT, por lo que puede ser utilizada con fines comerciales. Está desarrollada con el lenguaje de programación Go. Es una base de datos de series de tiempo, que consiste en un sistema capaz de guardar y proveer de series de tiempo de un conjunto de datos asociados a una marca de tiempo. Este tipo de bases de datos son ideales para el Internet de las Cosas cuando se requiere manejar una gran cantidad de datos ya que las consultas son mucho más rápidas que las clásicas bases de datos relacionales. InfluxDB tiene la opción de poder instalarse de forma privada en un servidor o también proporcionan una versión web que es administrada por la empresa con diferentes planes de suscripción.

Se ha instalado la versión 1.7.7 siguiendo los siguientes pasos:



1. `wget https://dl.influxdata.com/influxdb/releases/influxdb_1.7.7_amd64.deb`
2. `sudo dpkg -i influxdb_1.7.7_amd64.deb`
3. `sudo systemctl start influxdb`
4. `sudo systemctl enable influxdb.service`

Una vez instalada InfluxDB es necesario crear la base de datos donde se almacenan los datos tras ser decodificados. Para ello, desde el terminal se ejecuta el comando `influx` para iniciar InfluxDB. Posteriormente, con el comando `create database "chirpstack"` se crea la base de datos con nombre `chirpstack`. Hecho esto, cada vez que el nodo envía una trama de información, se decodifica en el servidor de red y se guardan los datos en estructuras, creadas automáticamente, dentro de la base de datos "chirpstack" con el formato que se retorna en la función de decodificación, como se muestra en la Fig. 76. Debido a ello, hay una estructura por cada medida con nombre "device\_frmpayload\_data\_x" siendo x el sensor asociado a la medida, en la que se almacena la marca de tiempo y el valor de la medida del sensor. También se crea una estructura llamada "device\_uplink" donde quedan registrados el tiempo, el nombre de la aplicación, el identificador devEUI, el nombre del dispositivo, el valor de la tasa de datos, el valor del contador de trama, la frecuencia, el RSSI y el SNR de cada transmisión. Esta estructura no se utiliza ya que no incluye datos de interés.

```
> show measurements
name: measurements
name
----
device_frmpayload_data_co2
device_frmpayload_data_humedad
device_frmpayload_data_latitud
device_frmpayload_data_lluvia
device_frmpayload_data_longitud
device_frmpayload_data_luminosidad
device_frmpayload_data_presion
device_frmpayload_data_temperatura
device_frmpayload_data_tvoc
device_frmpayload_data_uva
device_frmpayload_data_uvbi
device_frmpayload_data_uvbr
device_uplink
```

Fig. 76. Estructuras dentro de la base de datos "chirpstack".

### 6.3.3. Framework Flask

Flask es un micro framework escrito en Python para crear aplicaciones web dinámicas de forma sencilla con el lenguaje de programación Python. Es modular ya que si se necesitan nuevas funcionalidades nuevas para el proyecto solo hace falta instalar nuevas extensiones o plugins.

Un framework es una estructura que se sigue para la creación de una aplicación, por lo que todos los proyectos desarrollados bajo Flask seguirán la misma estructura, quedando todas las carpetas y ficheros organizados. Flask es un framework de tipo no full stack por lo que es necesario instalar extensiones para que sea funcional desde el inicio.

La página web queda dividida en dos bloques, el primero puede llamarse controlador, encargado de gestionar toda la página web y al que el usuario final no tiene acceso. El segundo, es la funcionalidad visual con la que el usuario final puede interactuar y también puede tener acceso a la estructura que forma la página web.

El primer paso para comenzar con Flask es crear un entorno virtual, de esta forma la aplicación desarrollada queda aislada de otras aplicaciones Python. El directorio de Flask debe tener una estructura bien definida para tener acceso a todas las rutas de los ficheros que se van a utilizar. Para ello se define la siguiente estructura del proyecto:

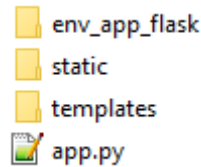


Fig. 77. Estructura dl proyecto Flask.

En la carpeta “env\_app\_flask” están todos los archivos necesarios para desplegar el entorno virtual y aislar la aplicación. En la carpeta “static” se guardan los archivos que utilizan recursos estáticos, como hojas de estilos CSS, imágenes y archivos JavaScript. En la carpeta “templates” se almacenan las plantillas HTML. En el directoria principal se encuentra el archivo app.py que se utiliza para implementar las funciones en Python del proyecto.

Tras ello se pueden instalar los paquetes de Python que se van a utilizar mediante el comando “pip3 install” que son: Flask, folium, influxdb y python-highcharts. En la Fig. 78 se muestran los módulos de los paquetes que se utilizan en el proyecto.

```
from flask import Flask, render_template, make_response
from influxdb import InfluxDBClient
from highcharts import Highchart
import folium
import random
import os
import shutil
```

Fig. 78. Módulos de los paquetes Python utilizado.

### 6.3.3.1. Acceso a la información de InfluxDB.

Para recuperar los datos almacenados en la base de datos es necesario realizar peticiones desde Flask a InfluxDB utilizando las funciones del paquete “influxdb”. Para ello es necesario especificar el host donde se encuentra instalada la base de datos, en este caso en el propio servidor, el puerto de conexión y el nombre de la base de datos, que es “chirpstack”. El siguiente paso es realizar la petición para lo que es necesario indicar la estructura que contiene los datos a consultar, en concreto se seleccionan todos los valores de la estructura “device\_frmpayload\_data\_mydata” siendo “mydata” el sensor del que queremos extraer las medidas. Con esta consulta se devuelve una lista Python con la estructura de pares de tiempo-valor. Para especificar que el tiempo lo devuelva en formato Unix hay que pasarle como parámetro la variable “epoch” con valor de nanosegundos. Tras obtener toda la lista con los valores mediante un bucle se accede a cada valor para introducirlo a las funciones de HighChart para ser graficado. En el código de la Fig. 79 se muestra la programación para poder obtener los valores de la base de datos.

```

user = 'admin'
password = 'admin'
dbname = 'chirpstack'
dbuser = 'smly'
dbuser_password = 'my_secret_password'
query1 = 'select * from device_frmpayload_data_mydata;'
query = 'select value from device_frmpayload_data_mydata;'
client = InfluxDBClient('localhost', 8086, database='chirpstack')

result = client.query(query, epoch='ns')

points = list(result.get_points(measurement='device_frmpayload_data_mydata'))

for x in range(len(points)):
    print(points[x].get('time'))
    print(int(points[x].get('value')))

```

Fig. 79. Código de acceso a InfluxDB.

### 6.3.3.2. Representación gráfica de los datos con HighCharts

HighCharts es una librería para representar gráficamente información que está escrita en JavaScript. En este proyecto se utiliza para mostrar de forma visual los datos de todos los sensores en una página web. Para ello, desde Flask usando el paquete “python-highcharts” se configuran los parámetros asociados a cada gráfico y se le pasan los datos a graficar como pares tiempo-valor en la variable “data”. En la Fig. 80 se muestra el código en Flask para implementar la visualización del gráfico de temperatura.

```

H = Highchart()

H.set_options('chart', {'zoomType': 'x'})
H.set_options('xAxis', {'type': 'datetime'})

H.set_options('yAxis', {'title': {'text': '°C'}})
H.set_options('legend', {'enabled': False})

H.set_options('title', {'text': 'Temperatura'})
H.set_options('legend', {'enabled': False})
H.set_options('credits', {'enabled': False})

H.add_data_set(data, 'line', 'Temperatura')
H.set_options('plotOptions', {
    'line': {
        }
    })

H.htmlcontent
H.save_file('templates/grafico')

```

Fig. 80. Creación del gráfico del valor de la temperatura con HighCharts.

El gráfico creado se guarda en la carpeta “templates” por lo que para insertarlo en la plantilla HTML primero hay que extraerlo del archivo creado y después renderizarlo, para ello se debe ejecutar el código que se muestra en la Fig. 81.

```

@app.route('/get_grafico')
def get_grafico():
    r = int(random.triangular(0,100))
    t = "templates/grafico_{i}.html"
    for i in range(0,100):
        f = t.format(i=i)
        if os.path.exists(f):
            os.remove(f)
    f = t.format(i=r)
    shutil.copy("templates/grafico.html", f)

    r = make_response(render_template(os.path.split(f)[1]))
    r.cache_control.max_age = 0
    r.cache_control.no_cache = True
    r.cache_control.no_store = True
    r.cache_control.must_revalidate = True
    r.cache_control.proxy_revalidate = True
    return r

```

Fig. 81. Extracción y renderizado del gráfico de la temperatura.

Por último, para insertar el gráfico en la plantilla HTML se utiliza el componente “iframe” que permite incrustar objetos dentro de la página web. Por ejemplo, en la Fig. 82 se muestra el código para insertar el gráfico de la temperatura en una sección de la plantilla HTML.

```

<!-- Temperatura-->
<section class="bg-light" id="temperatura">
  <div class="container px-4">
    <div class="row gx-4 justify-content-center">
      <div class="col-lg-8">
        <h2>Temperatura</h2>
        <iframe class="grafico", src="/get_grafico" width="900" height="700"></iframe>
      </div>
    </div>
  </div>
</section>

```

Fig. 82. Código HTML para insertar el gráfico de la temperatura.

### 6.3.3.3. Visualización de la posición en un mapa

Las coordenadas GPS permiten ubicar el nodo en un mapa. Para ello se utiliza el paquete de Python “folium” que facilita la visualización de mapas de Leaflet utilizando mosaicos de OpenStreetMap. Simplemente hay que pasarle los datos de latitud y longitud que se han extraído previamente de la base de datos InfluxDB, especifica el mosaico a utilizar, en este caso se utiliza OpenStreetMap y se añade un marcador con la posición del nodo en el que si se hace clic aparece un pop-up con “Nodo LoRaWAN”. Tras realizar la configuración del mapa, se guarda en la carpeta de plantillas como se puede observar en el código de la Fig. 83.

```

coordenadas = (latitud, longitud)

folium_map = folium.Map(
    location = coordenadas,
    zoom_start = 17
)

folium.TileLayer('openstreetmap').add_to(folium_map)

folium.Marker(
    [latitud, longitud],
    popup = "Nodo LoRaWAN",
    icon = folium.Icon(color="red")
).add_to(folium_map)

folium_map.save('templates/map.html')

```

Fig. 83. Código para crear el mapa.

Al igual que ocurría con los gráficos de HighCharts es necesario extraer y renderizar el mapa, véase el código de la Fig. 84, para insertarlo en la plantilla de HTML principal mediante un componente “iframe” como se muestra en el código de la Fig. 85.

```

@app.route('/get_map')
def get_map():
    r = int(random.triangular(0,100))
    t = "templates/map_{i}.html"
    for i in range(0,100):
        f = t.format(i=i)
        if os.path.exists(f):
            os.remove(f)
    f = t.format(i=r)
    shutil.copy("templates/map.html", f)

    r = make_response(render_template(os.path.split(f)[1]))
    r.cache_control.max_age = 0
    r.cache_control.no_cache = True
    r.cache_control.no_store = True
    r.cache_control.must_revalidate = True
    r.cache_control.proxy_revalidate = True
    return r

```

Fig. 84. Extracción y renderizado del mapa.

```

<!-- GPS-->
<section id="mapa">
    <div class="container px-4">
        <div class="row gx-4 justify-content-center">
            <div class="col-lg-8">
                <h2>Posición GPS del nodo</h2>
                <iframe class="map", src="/get_map" width="810" height="600"></iframe>
            </div>
        </div>
    </div>
</section>

```

Fig. 85. Código HTML para insertar el mapa.

#### 6.3.3.4. Página web utilizando plantilla de Bootstrap

La página web donde se visualizan los datos se ha diseñado partiendo de la plantilla “Scrollong Nav” [41] que está disponible para su descarga gratuita en la página Start Bootstrap. Es una plantilla sencilla de modificar, que está basada en el diseño en una sola página web con distintas secciones a las que

se puede acceder mediante un menú que hay en la parte superior o desplazándose hasta llegar a la sección deseada. Los archivos que forman la plantilla se describen a continuación:

- *index.html*: Archivo que contiene la plantilla HTML donde se definen las diferentes secciones que forman la página web.
- *style.css*: Hoja de estilos CSS.
- *script.js*: Script que proporciona la funcionalidad JavaScript a la página web.

La página web tienen un encabezado de color negro que se va desplazando a la vez que nos desplazamos por la página web, quedando siempre en la parte superior de la pantalla. Desde él, se puede ir a las secciones que componen la página web haciendo un clic en el dato que se quiere visualizar.

También se muestra una sección principal de color azul en la que se detallan el título, autor y tutor del Trabajo Fin de Máster, ya que es la primera sección visible cuando se accede a la página web.



Fig. 86. Encabezado de la página web.

La sección de visualización del mapa se muestra el punto de coordenadas donde se encuentra el dispositivo LoRa. Si se pincha en el punto de coordenadas GPS aparece un desplegable con la información “Nodo LoRaWAN”.

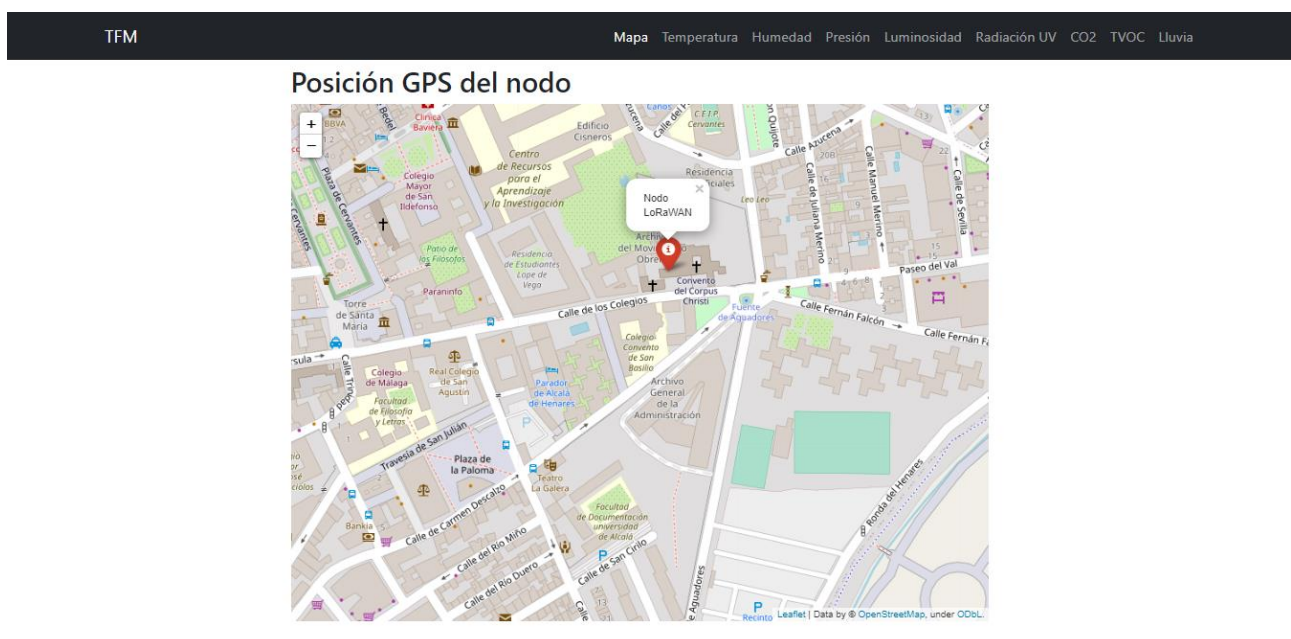


Fig. 87. Mapa con la posición del nodo en la página web.

Las demás secciones se corresponden con la visualización de los datos de los sensores. Por ejemplo, si se pincha en la sección de temperatura se muestra la gráfica como se puede ver en la Fig. 88

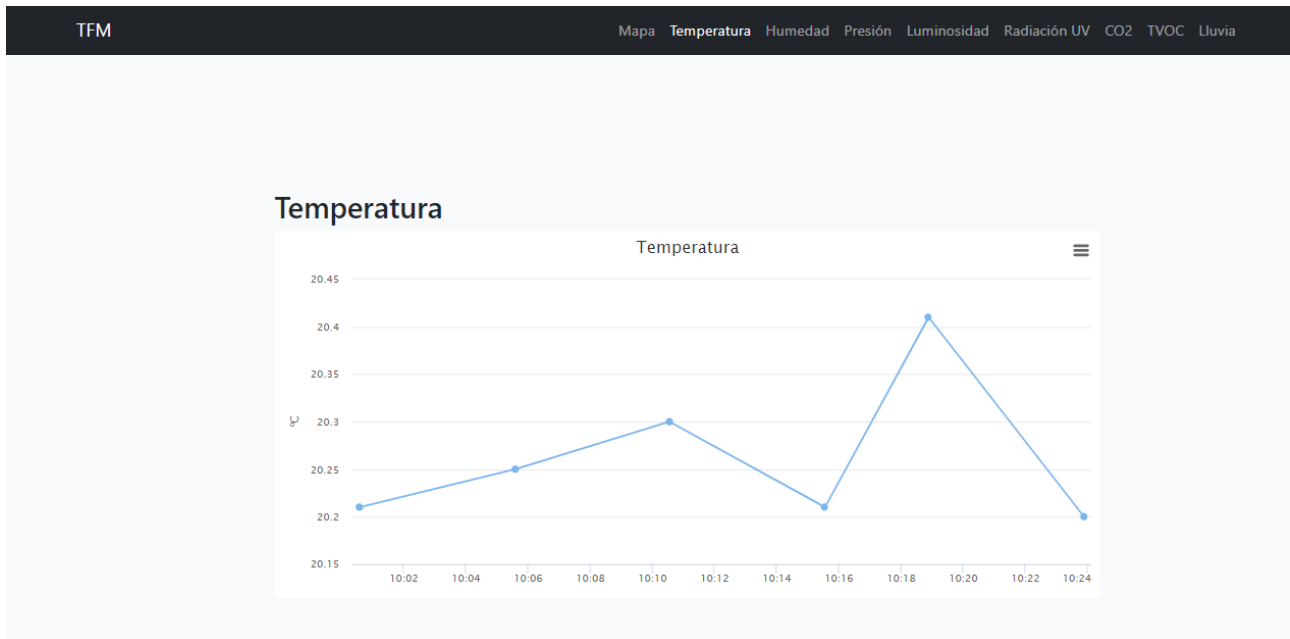


Fig. 88. Representación gráfica de los datos en la página web.

La página web también dispone de un pie de página que sirve para indicar el final de la visualización de los datos y, por tanto, el final de la página web.

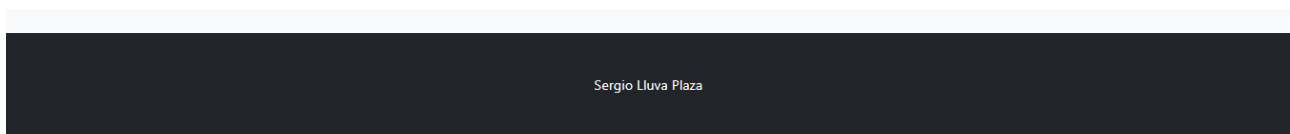


Fig. 89. Pie de página de la página web.





## 7. Conclusiones y trabajo futuro

En esta sección se desarrollan las conclusiones obtenidas tras la finalización del Trabajo Fin de Máster, así como se proponen una serie de nuevas funcionalidades que podrían ser aplicadas al proyecto.

### 7.1. Conclusiones

Se ha desarrollado un sistema basado en un microcontrolador STM32 capaz de leer la información de sensores que proporcionan información de parámetros ambientales, para su posterior transmisión haciendo uso de LoRa y LoRaWAN. La información enviada por el dispositivo se recibe en el gateway LoRaWAN y es reenviada a un servidor de Internet desplegado en Amazon Web Services. En el servidor se encuentra instalado ChirpStack, que implementa una pila basada en LoRaWAN para recibir la información y gestionar la comunicación mediante el servidor de red. Posteriormente, se almacena la información en la base de datos InfluxDB instalada de forma local en el servidor y enlazada directamente con ChirpStack utilizando una de las opciones de integración que tiene disponibles. La información de los sensores se recupera desde el framework Flask mediante peticiones a la base de datos para ser representada gráficamente en una página web utilizando una plantilla basada en Bootstrap para que sea una tarea sencilla y que la página web se adapte a todos los dispositivos.

Para llegar a conseguir implementar todo el desarrollo, en primer lugar, se ha realizado el diseño hardware del nodo LoRa implementado posteriormente en una PCB para que quede todo integrado en una sola placa. Todo ello tras implementar el dispositivo mediante un prototipado basado en tarjetas de evaluación independientes que se han conectado entre sí para verificar su correcto funcionamiento. El nodo LoRa por sí solo no dispone de funcionalidad, por lo que es necesario programarlo para que lea la información de los sensores y envíe los datos mediante LoRaWAN, para ello, se han utilizado librerías y drivers proporcionados por los fabricantes de cada dispositivo y se han modificado e integrado para que funcionen correctamente.

El gateway Lorix One también es necesario configurarlo con los parámetros adecuados para que se comunique correctamente con el nodo LoRa y con el servidor de red. Para ello se configuran los identificadores, dirección IP del servidor y parámetros asociados a la capa física LoRa y al protocolo LoRaWAN.

En la parte del servidor se han integrado varios componentes, en concreto ChirpStack, InfluxDB y Flask. ChirpStack se configura para que se reciban los datos desde el nodo y que el gateway se comunique con él. Haciendo uso de las integraciones, se selecciona InfluxDB para que se almacenen los datos directamente desde ChirpStack. Desde Flask, utilizando Python, se hacen consultas a la base de datos para recoger los datos, se procesan y se grafican haciendo uso de HighCharts mediante gráficos interactivos en una página web que se despliega desde Flask en la que también se incrusta un mapa de OpenStreetMap que muestra la posición del nodo.

Con la realización de este Trabajo Fin de Máster se ha desarrollado un proyecto que incluye tareas de Ingeniería Electrónica, Ingeniería de Sistemas de Telecomunicación e Ingeniería Telemática que son las principales áreas en las que se forman a los estudiantes del Máster en Ingeniería de Telecomunicación de la Universidad de Alcalá.

## 7.2. Trabajo futuro

Los principales objetivos se han cumplido durante la realización del proyecto, pero hay aportaciones que pueden realizarse que pueden ser de interés para llegar a un sistema con nuevas funcionalidades. En concreto, las nuevas funcionalidades se describen a continuación:

- Adaptar el código del nodo para recibir mensajes desde el servidor. De esta forma se puede implementar una comunicación bidireccional que puede servir para configurar parámetros en el nodo como puede ser el tiempo de envío de datos.
- Implementar un sistema de login de usuarios en la parte de la aplicación web, lo que dotaría de seguridad al sistema.
- Incluir más sensores que proporcionen información adicional sobre parámetros ambientales.
- Analizar el consumo de energía de todo el sistema.
- Analizar la escalabilidad del sistema LoRaWAN junto con un análisis de colisiones de paquetes cuando hay una alta densidad de nodos.

## Bibliografía

- [1] Internet of Things. Cisco, marzo 2020. Disponible online: <https://www.cisco.com/c/en/us/products/collateral/se/internet-of-things/at-a-glance-c45-731471.html> (último acceso: 5 septiembre 2021).
- [2] Singh, R.K., Puluckul, P.P., Berkvens, R., Weyn, M., “Energy Consumption Analysis of LPWAN Technologies and Lifetime Estimation for IoT Application,” Sensors 2020.
- [3] Kais Mekki, Eddy Bajic, Frederic Chaxel, Fernand Meyer, “A comparative study of LPWAN technologies for large-scale IoT deployment,” ICT Express, Volume 5, Issue 1, 2019, Pages 1-7, ISSN 2405-9595, <https://doi.org/10.1016/j.icte.2017.12.005>.
- [4] Semtech, “SX1276/77/78/79 Product Datasheet,” pp. 1-132, 2020.
- [5] Github, “Arduino-LMIC library (“MCCI LoRaWAN LMIC Library”)”. Disponible online: <https://github.com/mcci-catena/arduino-lmic> (último acceso: 6 septiembre 2021).
- [6] Github, “LoRaMAC-node”. Disponible online: <https://github.com/Lora-net/LoRaMac-node> (último acceso: 6 septiembre 2021).
- [7] Dragino, octubre 2020. Disponible online: <https://www.dragino.com/products/lora-lorawan-end-node.html> (último acceso: 6 septiembre 2021).
- [8] Dragino, “LBT1 -- LoRaWAN BLE Indoor Tracker”. Disponible online: <https://www.dragino.com/products/lora-lorawan-end-node/item/165-lbt1.html> (último acceso: 5 septiembre 2021).
- [9] ST Microelectronics, “LoRaWAN software expansion for STM32Cube (UM2073)” Disponible online: <https://www.st.com/en/embedded-software/i-cube-lrwan.html> (último acceso: 6 septiembre 2021).
- [10] Kerlin, “Gateway Wirnet Station” Disponible online: <https://www.kerlink.com/product/wirnet-station/> (último acceso: 6 septiembre 2021).
- [11] MultiTech, “MultiTech Conduit. Programmable Gateway for the Internet of Things. Product Datasheet,” pp. 1-4, 2021.
- [12] The Things Network, “The Things Indoor Gateway” Disponible online: <https://www.thethingsnetwork.org/docs/gateways/thethingsindoor/> (último acceso: 6 septiembre 2021).
- [13] Wifx, “Lorix One Gateway” Disponible online: <https://www.lorixone.io/en> (último acceso: 6 septiembre 2021).

- [14] Github, “ESP32 1 channel Gateway” Disponible online: <https://github.com/things4u/ESP-1ch-Gateway-v5.0> (último acceso: 6 septiembre 2021).
- [15] Github, “RAK Wireless” Disponible online: [https://github.com/RAKWireless/rak\\_common\\_for\\_gateway](https://github.com/RAKWireless/rak_common_for_gateway) (último acceso: 6 septiembre 2021).
- [16] Gateway Raspberry Pi and RAK833. Disponible online: <https://circuitmaker.com/Projects/Details/Craig-Peacock-4/RAK833-LoRaWAN-Concentrator-Hat> (último acceso: 6 septiembre 2021).
- [17] TTN, “The Things Network”. Disponible online: <https://www.thethingsnetwork.org/> (último acceso: 6 septiembre 2021).
- [18] ChirpStack, “ChirpStack, open-source LoRaWAN® Network Server stack”. Disponible online: <https://www.chirpstack.io/> (último acceso: 6 septiembre 2021).
- [19] Loriot, “Loriot Network Server”. Disponible online: <https://www.loriot.io/> (último acceso: 6 septiembre 2021).
- [20] LoRa Documentation, “What are LoRa and LoRaWAN?”. Disponible online: <https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan> (último acceso: 6 septiembre 2021).
- [21] LoRa Alliance. Disponible online: <https://lora-alliance.org/> (último acceso: 6 septiembre 2021).
- [22] The Things Network, “Device Classes”. Disponible online: <https://www.thethingsnetwork.org/docs/lorawan/classes/> (último acceso: 6 septiembre 2021).
- [23] A. Augustin, J. Yi, T. Clausen, and W. Townsley, “A Study of LoRa: Long Range & Low Power Networks for the Internet of Things,” *Sensors*, vol. 16, no. 9, p. 1466, Sep. 2016.
- [24] Semtech, “Why LoRa?”. Disponible online: <https://www.semtech.com/lora/why-lora> (último acceso: 6 septiembre 2021).
- [25] Moko Smart, “¿Cuál es la tecnología detrás de la frecuencia LoRa?”. Disponible online: <https://www.mokosmart.com/es/lora-frequency/> (último acceso: 6 septiembre 2021).
- [26] LoRa Tools, “Air Time Calculator”. Disponible online: <https://loratools.nl/#/airtime> (último acceso: 6 septiembre 2021).
- [27] Vishay Semiconductors, “VEML6075 Product Datasheet,” pp. 1-11, 2016.
- [28] Bosch Sensortec, “BME680 Product Datasheet,” pp. 1-50, 2017.
- [29] Github, “BME680 driver”. Disponible online: [https://github.com/BoschSensortec/BME680\\_driver](https://github.com/BoschSensortec/BME680_driver) (último acceso: 6 septiembre 2021).
- [30] Vishay Semiconductors, “VEML7700 Product Datasheet,” pp. 1-15, 2021.
- [31] AMS, “CCS811 Product Datasheet,” pp. 1-36, 2016.
- [32] Quectel, “L80 Hardware Design,” pp. 1-49, 2016.
- [33] Quectel, “L80 GPS Protocol Specification,” pp. 1-44, 2014.
- [34] Github, “Heltec Automation. ESP32 LoRaWAN”. Disponible online: [https://github.com/HelTecAutomation/ESP32\\_LoRaWAN](https://github.com/HelTecAutomation/ESP32_LoRaWAN) (último acceso: 6 septiembre 2021).
- [35] ST Microelectronics, “Low-power wireless Nucleo pack with Nucleo-L073RZ and LoRa expansion board”. Disponible online: [https://www.st.com/content/st\\_com/en/products/evaluation-tools/product-evaluation-tools/stm32-nucleo-expansion-boards/p-nucleo-lrwan1.html](https://www.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/stm32-nucleo-expansion-boards/p-nucleo-lrwan1.html) (último acceso: 6 septiembre 2021).
- [36] Peregrine Semiconductor, “PE4259 Product Datasheet,” pp. 1-10, 2016.

- [37] LoRaMAC Documentation, “Architecture”. Disponible online: <https://stackforce.github.io/LoRaMac-doc/LoRaMac-doc-v4.5.2/index.html> (último acceso: 6 septiembre 2021).
- [38] RF Wireless World, “LoRaWAN Packet Forwarder”. Disponible online: <https://www.rfwireless-world.com/Terminology/LoRaWAN-Packet-Forwarder.html> (último acceso: 6 septiembre 2021).
- [39] Lora-net, “Packet Forwarder”. Disponible online: [https://github.com/Lora-net/packet\\_forwarder](https://github.com/Lora-net/packet_forwarder) (último acceso: 6 septiembre 2021).
- [40] gRCP ecosystem, “gRPC-Gateway”. Disponible online: <https://github.com/grpc-ecosystem/grpc-gateway> (último acceso: 6 septiembre 2021).
- [41] Start Bootstrap, “Scrolling Nav Template”. Disponible online: <https://startbootstrap.com/template/scrolling-nav> (último acceso: 6 septiembre 2021).



## Anexos

### Anexo A. Presupuesto

En esta sección se presentan los costes asociados al proyecto realizado. Los diferentes costes se han dividido en tres secciones: costes de equipo y software, coste de material y coste de personal.

#### A.1. Costes de equipo y software

Los costes asociados al equipo informático necesario para realizar el proyecto, así como los costes asociados al software necesario se muestran en la Tabla II.

Concepto	Precio	Amortización	Uso	Total
HP Notebook 15-DA0149NS Intel Core i7-7500U/12GB/256GB SSD/15.6"	800 €	48 meses	6 meses	100 €
Keil uVision 5 (versión de evaluación)	0 €	N/A	N/A	0 €
Microsoft Office 365 (licencia universitaria)	0 €	N/A	N/A	0 €
Altium Designer 19.0.5 (versión de evaluación)	0 €	N/A	N/A	0 €
Máquina virtual en AWS (capa gratuita)	0 €	N/A	N/A	0 €
<b>Total</b>				<b>100 €</b>

Tabla II. Costes de equipo y software.

#### A.2. Costes de material

Los costes asociados al material utilizado para el diseño de la PCB y los elementos necesarios para la realización del proyecto se detallan en la Tabla III.

Concepto	Unidades	Precio/Unidad	Total
Gateway Lorix One	1	500 €	500 €
Fabricación PCB	1	25 €	25 €
Componentes para el ensamblaje de la PCB	1	30 €	30 €

<b>Panel solar</b>	1	5 €	5 €
<b>Tarjeta de desarrollo NUCLEO-L152RE</b>	1	12 €	12 €
<b>Tarjeta de desarrollo SX1276MB1MAS</b>	1	85 €	85 €
<b>Depurador ST-Link</b>	1	5 €	5 €
<b>Receptor GPS L80-M39</b>	1	13 €	13 €
<b>Tarjeta con sensor BME680</b>	1	7 €	7 €
<b>Tarjeta con sensor VEML6075</b>	1	4 €	4 €
<b>Tarjeta con sensor VEML7700</b>	1	5 €	5 €
<b>Tarjeta con sensor CSS811</b>	1	10 €	10 €
<b>Sensor de lluvia</b>	1	2 €	2 €
<b>Total</b>			703 €

Tabla III. Costes de material.

### A.3. Costes de personal

En la Tabla IV se muestran los costes asociados a la mano de obra del trabajo.

<b>Concepto</b>	<b>Horas</b>	<b>€/Hora</b>	<b>Total</b>
<b>Horas ingeniero</b>	300	30	9000 €
<b>Total</b>			9000 €

Tabla IV. Costes de personal.

### A.4. Presupuesto total

El presupuesto final del desarrollo del proyecto es el resultado de la suma de los costes de equipo y software, los costes de material y los costes de personal, a lo que hay que añadir el IVA (Impuesto sobre el Valor Añadido) de un 21% sobre los distintos costes.

<b>Concepto</b>	<b>Total</b>
<b>Costes de equipo y software</b>	100 €
<b>Costes de material</b>	703 €
<b>Costes de personal</b>	9000 €
<b>IVA (21%)</b>	2058.63 €
<b>Total</b>	11861.63 €

Tabla V. Presupuesto total.



## Anexo B: Pliego de condiciones

En esta sección se enumeran las características de los equipos utilizados y las versiones del software utilizado durante la realización del proyecto.

Equipo utilizado:

- Ordenador portátil HP Notebook 15-DA0149NS Intel Core i7 7500U 12GB de RAM y 256GB de SSD. Sistema operativo: Windows 10
- Máquina virtual en AWS con 1GB de RAM y 8GB de SDD. Sistema operativo: Ubuntu Server 18.04 LTS

Software utilizado:

- Keil uVision 5 V5.28.0.0
- Microsoft Office 365
- Altium Designer 19.0.15 (Build 446)

Versiones de herramientas utilizadas:

- Python 3.6.9
- Flask 2.0.1
- InfluxDB 1.7.7
- ChirpStack 3.15.0
- Librería LoRaWAN de ST Microelectronics V1.3.1

# Anexo C. Código de programación del microcontrolador

## Anexo C1. Código del fichero principal (main.c)

```

#include "hw.h"
#include "low_power_manager.h"
#include "lora.h"
#include "bsp.h"
#include "timeServer.h"
#include "vcom.h"
#include "version.h"

//Periféricos
#include "i2c.h"
#include "uart.h"

//BME680
#include "bme680.h"
#include "bme_stm32_hal.h"

//VEML6075
#include "VEML6075.h"

//VEML7700
#include "VEML7700.h"

//CCS811
#include "CCS811.h"
#include "CCS811_defs.h"

//GPS
#include "GPS.h"
#include "string.h"
/* Private typedef -----*/
/* Private define -----*/

/*!
 * Defines the application data transmission duty cycle. value in [ms].
 */
#define APP_TX_DUTYCYCLE 300000 //5 minutos
/*!
 * LoRaWAN Adaptive Data Rate
 * @note Please note that when ADR is enabled the end-device should be static
 */
#define LORAWAN_ADR_STATE LORAWAN_ADR_OFF
/*!
 * LoRaWAN Default data Rate Data Rate
 * @note Please note that LORAWAN_DEFAULT_DATA_RATE is used only when ADR is disabled
 */
#define LORAWAN_DEFAULT_DATA_RATE DR_5
/*!
 * LoRaWAN application port
 * @note do not use 224. It is reserved for certification
 */
#define LORAWAN_APP_PORT 2
/*!
 * LoRaWAN default endNode class port
 */
#define LORAWAN_DEFAULT_CLASS CLASS_A
/*!
 * LoRaWAN default confirm state
 */
#define LORAWAN_DEFAULT_CONFIRM_MSG_STATE LORAWAN_UNCONFIRMED_MSG
/*!
 * User application data buffer size
 */
#define LORAWAN_APP_DATA_BUFF_SIZE 33

//UART DEBUG
static uint8_t tx_buffer[100];

//BME680
struct bme680_dev gas_sensor;
struct bme680_field_data data;

```

```

int8_t rslt = BME680_OK;
uint16_t meas_period;

//VEML6075
float uva_float = 0;
float uvb_float = 0;
float uv = 0;
uint16_t id;

//CCS811
uint8_t CCS811_TestBuffer[10] = {0};
extern bool baselineCMD;
int8_t rslt1 = CCS811_OK;

//VEML7700
uint16_t als, white;

//Lluvia
uint8_t pin_lluvia = 0;

/*!
 * User application data
 */
static uint8_t AppDataBuff[LORAWAN_APP_DATA_BUFF_SIZE];

/*!
 * User application data structure
 */
//static lora_AppData_t AppData={ AppDataBuff, 0 ,0 };
lora_AppData_t AppData = { AppDataBuff, 0, 0 };

/* Private macro -----*/
/* Private function prototypes -----*/

/* call back when LoRa endNode has received a frame*/
static void LORA_RxData(lora_AppData_t *AppData);

/* call back when LoRa endNode has just joined*/
static void LORA_HasJoined(void);

/* call back when LoRa endNode has just switch the class*/
static void LORA_ConfirmClass(DeviceClass_t Class);

/* call back when server needs endNode to send a frame*/
static void LORA_TxNeeded(void);

/* LoRa endNode send request*/
static void Send(void *context);

/* start the tx process*/
static void LoraStartTx(TxEventType_t EventType);

/* tx timer callback function*/
static void OnTxTimerEvent(void *context);

/* tx timer callback function*/
static void LoraMacProcessNotify(void);

/* Private variables -----*/
/* load Main call backs structure*/
static LoRaMainCallback_t LoRaMainCallbacks = {
    HW_GetTemperatureLevel,
    HW_GetUniqueId,
    HW_GetRandomSeed,
    LORA_RxData,
    LORA_HasJoined,
    LORA_ConfirmClass,
    LORA_TxNeeded,
    LoraMacProcessNotify
};

LoraFlagStatus LoraMacProcessRequest = LORA_RESET;
LoraFlagStatus AppProcessRequest = LORA_RESET;

static TimerEvent_t TxTimer;

/* !
 *Initialises the Lora Parameters

```

```

*/
static LoRaParam_t LoRaParamInit = {LORAWAN_ADR_STATE,
                                     LORAWAN_DEFAULT_DATA_RATE,
                                     LORAWAN_PUBLIC_NETWORK
                                     };

/* Private functions -----*/
//CCS811
int8_t user_ccs811_read(uint8_t id, uint8_t reg_addr, uint8_t *data, uint16_t len);
int8_t user_ccs811_write(uint8_t id, uint8_t reg_addr, uint8_t *data, uint16_t len);
void user_delay_ms(uint32_t period);

//GPS UART
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    GPS_CallBack();
}

int main(void)
{
    //BME680 enlace funciones
    gas_sensor.dev_id = BME680_I2C_ADDR_SECONDARY;
    gas_sensor.intf = BME680_I2C_INTF;
    gas_sensor.read = bme680_i2c_read;
    gas_sensor.write = bme680_i2c_write;
    gas_sensor.delay_ms = bme680_delay_ms;
    gas_sensor.amb_temp = 25;

    /* STM32 HAL library initialization*/
    HAL_Init();

    /* Configure the system clock*/
    SystemClock_Config();

    /* Configure the debug mode*/
    DBG_Init();

    /* Configure the hardware*/
    HW_Init();

    /* USER CODE BEGIN 1 */
    //Iniciación de periféricos
    MX_I2C1_Init();
    MX_USART3_UART_Init();

    //Iniciación del pin de entrada del sensor de lluvia PC2
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    /* GPIO Ports Clock Enable */
    __HAL_RCC_GPIOC_CLK_ENABLE();
    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_2, GPIO_PIN_RESET);
    /*Configure GPIO pin : PC2 */
    GPIO_InitStruct.Pin = GPIO_PIN_2;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

    //BME680
    rslt = bme680_init(&gas_sensor);
    uint8_t set_required_settings;
    //Configuración
    gas_sensor.tph_sett.os_hum = BME680_OS_2X;
    gas_sensor.tph_sett.os_pres = BME680_OS_4X;
    gas_sensor.tph_sett.os_temp = BME680_OS_8X;
    gas_sensor.tph_sett.filter = BME680_FILTER_SIZE_3;
    //Perfil de calentamiento
    gas_sensor.gas_sett.run_gas = BME680_ENABLE_GAS_MEAS;
    gas_sensor.gas_sett.heatr_temp = 320; /* degree Celsius */
    gas_sensor.gas_sett.heatr_dur = 150; /* milliseconds */
    //Power mode
    gas_sensor.power_mode = BME680_FORCED_MODE;

```

```

//Configuración
set_required_settings = BME680_OST_SEL | BME680_OSP_SEL | BME680_OSH_SEL | BME680_FILTER_SEL
| BME680_GAS_SENSOR_SEL;
rslt = bme680_set_sensor_settings(set_required_settings,&gas_sensor);
rslt = bme680_set_sensor_mode(&gas_sensor);

//VEML6075
VEML6075_ShutDown( &hi2c1, true );
HAL_Delay(50);
VEML6075_ShutDown( &hi2c1, false );
HAL_Delay(50);
VEML6075_WhoAmI( &hi2c1, &id );

//CCS811
rslt = CCS811_OK;
struct ccs811_dev ccs811;
struct ccs811_data ccs811Samples;
ccs811.dev_id = CCS811_ADDR;
ccs811.read = user_ccs811_read;
ccs811.write = user_ccs811_write;
ccs811.delay_ms = user_delay_ms;
ccs811.mode = CCS811_MODE1;
rslt = ccs811_init(&ccs811);

//VEML7700
setALSConf(0x0000);
setPowerSaving(0x0000);

//GPS
GPS_Init();

/*Disbale Stand-by mode*/
LPM_SetOffMode(LPM_APPLI_Id, LPM_Disable);

/* Configure the Lora Stack*/
LORA_Init(&LoRaMainCallbacks, &LoRaParamInit);

LORA_Join();

LoraStartTx(TX_ON_TIMER) ;

while (1)
{
if (AppProcessRequest == LORA_SET)
{
/*reset notification flag*/
AppProcessRequest = LORA_RESET;
/*Send*/
Send(NULL);
}
if (LoraMacProcessRequest == LORA_SET)
{
/*reset notification flag*/
LoraMacProcessRequest = LORA_RESET;
LoRaMacProcess();
}
/*If a flag is set at this point, mcu must not enter low power and must loop*/
DISABLE_IRQ();

/* if an interrupt has occurred after DISABLE_IRQ, it is kept pending
* and cortex will not enter low power anyway */
if ((LoraMacProcessRequest != LORA_SET) && (AppProcessRequest != LORA_SET))
{
#ifdef LOW_POWER_DISABLE
LPM_EnterLowPower();
#endif
}

ENABLE_IRQ();

/* USER CODE BEGIN 2 */
/* USER CODE END 2 */
}
}

//Lectura I2C CCS811
int8_t user_ccs811_read(uint8_t id, uint8_t reg_addr, uint8_t *data, uint16_t len)

```

```

{
    int8_t ret = 0;

    if( HAL_I2C_Master_Transmit( &hi2c1, (uint16_t)(id<<1), &reg_addr, 1, 100 ) != HAL_OK )
    {
        return -1;
    }
    HAL_Delay(1);
    if( HAL_I2C_Master_Receive( &hi2c1, (uint16_t)(id<<1), data, len, 100 ) != HAL_OK )
    {
        return -1;
    }

    return 0;
}

//Escritura I2C CCS811
int8_t user_ccs811_write(uint8_t id, uint8_t reg_addr, uint8_t *data, uint16_t len)
{
    int8_t ret = 0;
    uint8_t* i2c_data;

    i2c_data[0] = reg_addr;
    memcpy( &i2c_data[1], data, len);

    if( HAL_I2C_Master_Transmit( &hi2c1, (uint16_t)(id<<1), i2c_data, len+1, 100 ) != HAL_OK )
    {
        return -1;
    }

    free(i2c_data);
    return ret;
}

//Delay CCS811
void user_delay_ms(uint32_t period)
{
    HAL_Delay(period);
}

void LoraMacProcessNotify(void)
{
    LoraMacProcessRequest = LORA_SET;
}

static void LORA_HasJoined(void)
{
    LORA_RequestClass(LORAWAN_DEFAULT_CLASS);
}

static void Send(void *context)
{
    struct ccs811_dev ccs811;
    struct ccs811_data ccs811Samples;

    //Lectura de los datos de todos los sensores
    //BME680
    bme680_get_profile_dur(&meas_period, &gas_sensor);
    bme680_delay_ms(meas_period); /* Delay till the measurement is ready */
    rslt = bme680_get_sensor_data(&data, &gas_sensor);
    if(data.status & BME680_GASM_VALID_MSK)
    if (gas_sensor.power_mode == BME680_FORCED_MODE) {
        rslt = bme680_set_sensor_mode(&gas_sensor);
    }

    //VEML6075
    VEML6075_getUVA(&hi2c1, &uva_float);
    VEML6075_getUVB(&hi2c1, &uvb_float);
    VEML6075_calculateUVIndex(&hi2c1, &uv);

    //CCS811
    ccs811_get_data(&ccs811, &ccs811Samples);

    //VEML7700
    als = getALS();
}

```

```

//GPS
GPS_Process();

//Lluvia
pin_lluvia = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_2);

//Conversión para formar la trama
int32_t latitud = GPS.GPGGA.Latitude * 10000000.0f;
int32_t longitud = GPS.GPGGA.Longitude * 10000000.0f;
int32_t temperatura = data.temperature * 100.0f;
int32_t humedad = data.humidity * 100.0f;
int32_t presion = data.pressure * 100.0f;
int32_t luminosidad = als;
int32_t uva = uva float;
int32_t uvb = uvb float;
int32_t co2 = ccs811Samples.eCO2;
int32_t tvoc = ccs811Samples.VOCS;
int8_t lluvia = pin_lluvia;

AppData.Port = LORAWAN_APP_PORT;

if (LORA_JoinStatus() != LORA_SET)
{
    /*Not joined, try again later*/
    LORA_Join();
    return;
}

uint32_t i = 0;
//Latitud
AppData.Buff[i++] = latitud >> 24;
AppData.Buff[i++] = latitud >> 16;
AppData.Buff[i++] = latitud >> 8;
AppData.Buff[i++] = latitud;
//Longitud
AppData.Buff[i++] = longitud >> 24;
AppData.Buff[i++] = longitud >> 16;
AppData.Buff[i++] = longitud >> 8;
AppData.Buff[i++] = longitud;
//Temperatura
AppData.Buff[i++] = temperatura >> 24;
AppData.Buff[i++] = temperatura >> 16;
AppData.Buff[i++] = temperatura >> 8;
AppData.Buff[i++] = temperatura;
//Humedad
AppData.Buff[i++] = humedad >> 24;
AppData.Buff[i++] = humedad >> 16;
AppData.Buff[i++] = humedad >> 8;
AppData.Buff[i++] = humedad;
//Presión
AppData.Buff[i++] = presion >> 24;
AppData.Buff[i++] = presion >> 16;
AppData.Buff[i++] = presion >> 8;
AppData.Buff[i++] = presion;
//Luminosidad
AppData.Buff[i++] = luminosidad >> 24;
AppData.Buff[i++] = luminosidad >> 16;
AppData.Buff[i++] = luminosidad >> 8;
AppData.Buff[i++] = luminosidad;
//UVA
AppData.Buff[i++] = uva >> 24;
AppData.Buff[i++] = uva >> 16;
AppData.Buff[i++] = uva >> 8;
AppData.Buff[i++] = uva;
//UVB
AppData.Buff[i++] = uvb >> 24;
AppData.Buff[i++] = uvb >> 16;
AppData.Buff[i++] = uvb >> 8;
AppData.Buff[i++] = uvb;
//CO2
AppData.Buff[i++] = co2 >> 24;
AppData.Buff[i++] = co2 >> 16;
AppData.Buff[i++] = co2 >> 8;
AppData.Buff[i++] = co2;
//TVOC

```

```

    AppData.Buff[i++] = tvoc >> 24;
    AppData.Buff[i++] = tvoc >> 16;
    AppData.Buff[i++] = tvoc >> 8;
    AppData.Buff[i++] = tvoc;
    //Lluvia
    AppData.Buff[i++] = lluvia;

AppData.BuffSize = i;

LORA_send(&AppData, LORAWAN_DEFAULT_CONFIRM_MSG_STATE);

/* USER CODE END 3 */
}

static void LORA_RxData(lora_AppData_t *AppData)
{

}

static void OnTxTimerEvent(void *context)
{
    /*Wait for next tx slot*/
    TimerStart(&TxTimer);

    AppProcessRequest = LORA_SET;
}

static void LoraStartTx(TxEventType_t EventType)
{
    if (EventType == TX_ON_TIMER)
    {
        /* send everytime timer elapses */
        TimerInit(&TxTimer, OnTxTimerEvent);
        TimerSetValue(&TxTimer, APP_TX_DUTYCYCLE);
        OnTxTimerEvent(NULL);
    }
    else
    {
        /* send everytime button is pushed */
        GPIO_InitTypeDef initStruct = {0};

        initStruct.Mode = GPIO_MODE_IT_RISING;
        initStruct.Pull = GPIO_PULLUP;
        initStruct.Speed = GPIO_SPEED_HIGH;

        HW_GPIO_Init(USER_BUTTON_GPIO_PORT, USER_BUTTON_PIN, &initStruct);
        HW_GPIO_SetIrq(USER_BUTTON_GPIO_PORT, USER_BUTTON_PIN, 0, Send);
    }
}

static void LORA_ConfirmClass(DeviceClass_t Class)
{
    /*Optionnal*/
    /*informs the server that switch has occurred ASAP*/
    AppData.BuffSize = 0;
    AppData.Port = LORAWAN_APP_PORT;

    LORA_send(&AppData, LORAWAN_UNCONFIRMED_MSG);
}

static void LORA_TxNeeded(void)
{
    AppData.BuffSize = 0;
    AppData.Port = LORAWAN_APP_PORT;

    LORA_send(&AppData, LORAWAN_UNCONFIRMED_MSG);
}

```



## Anexo C2. Código de configuración del bus I2C (i2c.c)

```

#include "i2c.h"

I2C_HandleTypeDef hi2c1;

/* I2C1 init function */
void MX_I2C1_Init(void)
{
    hi2c1.Instance = I2C1;
    hi2c1.Init.ClockSpeed = 100000;
    hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
    hi2c1.Init.OwnAddress1 = 0;
    hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
    hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
    hi2c1.Init.OwnAddress2 = 0;
    hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
    hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
    if (HAL_I2C_Init(&hi2c1) != HAL_OK)
    {
        Error_Handler();
    }
}

void HAL_I2C_MspInit(I2C_HandleTypeDef* i2cHandle)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    if(i2cHandle->Instance==I2C1)
    {
        __HAL_RCC_GPIOB_CLK_ENABLE();
        /**I2C1 GPIO Configuration
        PB8      -> I2C1_SCL
        PB9      -> I2C1_SDA
        */
        GPIO_InitStruct.Pin = GPIO_PIN_8|GPIO_PIN_9;
        GPIO_InitStruct.Mode = GPIO_MODE_AF_OD;
        GPIO_InitStruct.Pull = GPIO_PULLUP;
        GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
        GPIO_InitStruct.Alternate = GPIO_AF4_I2C1;
        HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

        /* I2C1 clock enable */
        __HAL_RCC_I2C1_CLK_ENABLE();
    }
}

void HAL_I2C_MspDeInit(I2C_HandleTypeDef* i2cHandle)
{
    if(i2cHandle->Instance==I2C1)
    {
        /* Peripheral clock disable */
        __HAL_RCC_I2C1_CLK_DISABLE();

        /**I2C1 GPIO Configuration
        PB8      -> I2C1_SCL
        PB9      -> I2C1_SDA
        */
        HAL_GPIO_DeInit(GPIOB, GPIO_PIN_8|GPIO_PIN_9);
    }
}

```

## Anexo C3. Código de configuración de la interfaz UART (uart.c)

```

#include "uart.h"

UART_HandleTypeDef huart3;

/* USART3 init function */
void MX_USART3_UART_Init(void)
{
    huart3.Instance = USART3;
    huart3.Init.BaudRate = 115200;
    huart3.Init.WordLength = UART_WORDLENGTH_8B;
    huart3.Init.StopBits = UART_STOPBITS_1;
    huart3.Init.Parity = UART_PARITY_NONE;
    huart3.Init.Mode = UART_MODE_TX_RX;
    huart3.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    huart3.Init.OverSampling = UART_OVERSAMPLING_16;
    if (HAL_UART_Init(&huart3) != HAL_OK)
    {
        Error_Handler();
    }
}

void HAL_UART3_MspInit(UART_HandleTypeDef* uartHandle)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};

    /* USART3 clock enable */
    __HAL_RCC_USART3_CLK_ENABLE();

    __HAL_RCC_GPIOB_CLK_ENABLE();
    /**USART3 GPIO Configuration
    PB10      -----> USART3_TX
    PB11      -----> USART3_RX
    */
    GPIO_InitStruct.Pin = GPIO_PIN_10|GPIO_PIN_11;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_PULLUP;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF7_USART3;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
}

void HAL_UART3_MspDeInit(UART_HandleTypeDef* uartHandle)
{
    /* Peripheral clock disable */
    __HAL_RCC_USART3_CLK_DISABLE();

    /**USART3 GPIO Configuration
    PB10      -----> USART3_TX
    PB11      -----> USART3_RX
    */
    HAL_GPIO_DeInit(GPIOB, GPIO_PIN_10|GPIO_PIN_11);
}

```

## Anexo C4. Código del sensor BME680 (BME680.c)

```

#include <stdlib.h>
#include <string.h>
#include "bme680.h"
#include "stm3211xx_hal.h"
#include "bme_stm32_hal.h"

extern I2C_HandleTypeDef hi2c1;

void bme680_delay_ms(uint32_t period)
{
    HAL_Delay(period);
}

int8_t bme680_i2c_read(uint8_t dev_id, uint8_t reg_addr, uint8_t *reg_data, uint16_t len)
{
    int8_t ret = 0;

    if( HAL_I2C_Master_Transmit( &hi2c1, (uint16_t)(dev_id<<1), &reg_addr, 1, 100 ) != HAL_OK )
    {
        return -1;
    }
    HAL_Delay(1);
    if( HAL_I2C_Master_Receive( &hi2c1, (uint16_t)(dev_id<<1), reg_data, len, 100 ) != HAL_OK )
    {
        return -1;
    }

    return ret;
}

int8_t bme680_i2c_write(uint8_t dev_id, uint8_t reg_addr, uint8_t *reg_data, uint16_t len)
{
    int8_t ret = 0;
    uint8_t* i2c_data;

    i2c_data = malloc( len+1 );
    i2c_data[0] = reg_addr;
    memcpy( &i2c_data[1], reg_data, len);

    if( HAL_I2C_Master_Transmit( &hi2c1, (uint16_t)(dev_id<<1), i2c_data, len+1, 100 ) != HAL_OK )
    {
        return -1;
    }

    free(i2c_data);
    return ret;
}

int8_t bme680_begin(struct bme680_dev* dev)
{
    dev->dev_id = BME680_I2C_ADDR_SECONDARY;
    dev->intf = BME680_I2C_INTF;
    dev->read = bme680_i2c_read;
    dev->write = bme680_i2c_write;
    dev->delay_ms = bme680_delay_ms;
    dev->amb_temp = 25;

    if( bme680_init(dev) != BME680_OK )
    {
        dev = NULL;
        return 0;
    }

    uint8_t set_required_settings;

    dev->tph_sett.os_hum = BME680_OS_2X;
    dev->tph_sett.os_pres = BME680_OS_4X;
    dev->tph_sett.os_temp = BME680_OS_8X;
    dev->tph_sett.filter = BME680_FILTER_SIZE_3;

    dev->gas_sett.run_gas = BME680_ENABLE_GAS_MEAS;
    dev->gas_sett.heatr_temp = 320; /* degree Celsius */
    dev->gas_sett.heatr_dur = 150; /* milliseconds */

```

```
dev->power_mode = BME680_FORCED_MODE;

set_required_settings = BME680_OST_SEL | BME680_OSP_SEL | BME680_OSH_SEL | BME680_FILTER_SEL
                       | BME680_GAS_SENSOR_SEL;

if( bme680_set_sensor_settings(set_required_settings, dev) != BME680_OK )
{
    return 0;
}

if( bme680_set_sensor_mode(dev) != BME680_OK )
{
    return 0;
}

return 1;
}
```

## Anexo C5. Código del sensor VEML6075 (VEML6075.c)

```

#include "VEML6075.h"

static HAL_StatusTypeDef VEML6075_ReadRegister(I2C_HandleTypeDef *hi2c, uint8_t adr, uint8_t reg,
uint16_t *regval)
{
    uint8_t val[2];
    HAL_StatusTypeDef status;

    status = HAL_I2C_Mem_Read(hi2c, adr<<1, reg , I2C_MEMADD_SIZE_8BIT, val, 2, 100);

    if (status == HAL_OK)
    {
        *regval = val[0] | val[1] << 8;
    }
    return status;
}

static HAL_StatusTypeDef VEML6075_WriteRegister(I2C_HandleTypeDef *hi2c, uint8_t adr, uint8_t reg,
uint16_t regval)
{
    uint8_t val[2];
    val[1] = (regval >> 8) & 0xff;
    val[0] = regval & 0xff;
    HAL_StatusTypeDef status = HAL_I2C_Mem_Write(hi2c, adr<<1, reg,
I2C_MEMADD_SIZE_8BIT, val, 2, 100);
    return status;
}

HAL_StatusTypeDef VEML6075_WhoAmI(I2C_HandleTypeDef *hi2c, uint16_t *idval)
{
    return VEML6075_ReadRegister(hi2c, VEML6075ADDRESS, VEML6075_ID, idval);
}

HAL_StatusTypeDef VEML6075_UVVAL(I2C_HandleTypeDef *hi2c, VEML6075_PARAM_t param, uint16_t *val)
{
    switch (param)
    {
        case VEML6075_PARAM_UVA:
            return VEML6075_ReadRegister(hi2c, VEML6075ADDRESS, VEML6075_UVA_DATA, val);

        case VEML6075_PARAM_UVB:
            return VEML6075_ReadRegister(hi2c, VEML6075ADDRESS, VEML6075_UVB_DATA, val);

        case VEML6075_PARAM_UVCOMP1:
            return VEML6075_ReadRegister(hi2c, VEML6075ADDRESS, VEML6075_UVCOMP1_DATA,
val);

        case VEML6075_PARAM_UVCOMP2:
            return VEML6075_ReadRegister(hi2c, VEML6075ADDRESS, VEML6075_UVCOMP2_DATA,
val);

        case VEML6075_PARAM_DARK:
            return VEML6075_ReadRegister(hi2c, VEML6075ADDRESS, VEML6075_DARK_DATA, val);
    }
    return HAL_ERROR;
}

HAL_StatusTypeDef VEML6075_ShutDown(I2C_HandleTypeDef *hi2c, bool sd)
{
    uint16_t regval = VEML6075_CONF_DEFAULT | (sd ? VEML6075_CONF_SD : 0);

    return VEML6075_WriteRegister(hi2c, VEML6075ADDRESS, VEML6075_UV_CONF, regval);
}

HAL_StatusTypeDef VEML6075_getUVA(I2C_HandleTypeDef *hi2c, float *uva)
{
    HAL_StatusTypeDef status;
    float comp_vis;
    float comp_ir;
    float comp_uva;
    uint16_t raw_vis, raw_ir, raw_uva, raw_dark;

```

```

status = VEML6075_UVVAL( hi2c, VEML6075_PARAM_UVA, &raw_uva);
    if (status == HAL_OK)
    {
        // get rest of the readouts
        status += VEML6075_UVVAL( hi2c, VEML6075_PARAM_UVCOMP1, &raw_vis);
        status += VEML6075_UVVAL( hi2c, VEML6075_PARAM_UVCOMP2, &raw_ir);
        status += VEML6075_UVVAL( hi2c, VEML6075_PARAM_DARK, &raw_dark);
    } else return HAL_ERROR;

comp_vis = (float)raw_vis - (float)raw_dark;
comp_ir  = (float)raw_ir - (float)raw_dark;
comp_uva = (float)raw_uva - (float)raw_dark;

comp_uva -= (VEML6075_UVI_UVA_VIS_COEFF * comp_vis) - (VEML6075_UVI_UVA_IR_COEFF * comp_ir);

*uva = comp_uva;

return status;
}

HAL_StatusTypeDef VEML6075_getUVB(I2C_HandleTypeDef *hi2c, float *uvb)
{
    HAL_StatusTypeDef status;
    float comp_vis;
    float comp_ir;
    float comp_uvb;
    uint16_t raw_vis, raw_ir, raw_uvb, raw_dark;

    status = VEML6075_UVVAL( hi2c, VEML6075_PARAM_UVB, &raw_uvb);
    if (status == HAL_OK)
    {
        // get rest of the readouts
        status += VEML6075_UVVAL( hi2c, VEML6075_PARAM_UVCOMP1, &raw_vis);
        status += VEML6075_UVVAL( hi2c, VEML6075_PARAM_UVCOMP2, &raw_ir);
        status += VEML6075_UVVAL( hi2c, VEML6075_PARAM_DARK, &raw_dark);
    } else return HAL_ERROR;

    comp_vis = (float)raw_vis - (float)raw_dark;
    comp_ir  = (float)raw_ir - (float)raw_dark;
    comp_uvb = (float)raw_uvb - (float)raw_dark;

    comp_uvb -= (VEML6075_UVI_UVB_VIS_COEFF * comp_vis) - (VEML6075_UVI_UVB_IR_COEFF * comp_ir);

    *uvb = comp_uvb;

    return status;
}

HAL_StatusTypeDef VEML6075_calculateUVIndex(I2C_HandleTypeDef *hi2c, float *uvindex)
{
    float uva_weighted;
    float uvb_weighted;
    HAL_StatusTypeDef status;

    status = VEML6075_getUVA(hi2c, &uva_weighted);
    uva_weighted *= VEML6075_UVI_UVA_RESPONSE;
    status += VEML6075_getUVB(hi2c, &uvb_weighted);
    uvb_weighted *= VEML6075_UVI_UVB_RESPONSE;

    if (status == HAL_OK)
    {
        *uvindex = (uva_weighted + uvb_weighted) / 2.0;
    }
    return status;
}

HAL_StatusTypeDef VEML6075_getreadouts(I2C_HandleTypeDef *hi2c, VEML6075_readout_t *reading)
{
    HAL_StatusTypeDef status;
    float comp_vis;
    float comp_ir;
    float comp_uva;
    float comp_uvb;

```

```

float uva_weighted;
float uvb_weighted;

uint16_t raw_vis, raw_ir, raw_uva, raw_uvb, raw_dark;

status = VEML6075_UVVAL( hi2c, VEML6075_PARAM_UVA, &raw_uva);
    if (status == HAL_OK)
    {
        status += VEML6075_UVVAL( hi2c, VEML6075_PARAM_UVCOMP1, &raw_vis);
        status += VEML6075_UVVAL( hi2c, VEML6075_PARAM_UVCOMP2, &raw_ir);
        status += VEML6075_UVVAL( hi2c, VEML6075_PARAM_DARK, &raw_dark);
        status += VEML6075_UVVAL( hi2c, VEML6075_PARAM_UVB, &raw_uvb);
    } else return HAL_ERROR;

comp_vis = (float)raw_vis - (float)raw_dark;
comp_ir  = (float)raw_ir  - (float)raw_dark;
comp_uva = (float)raw_uva - (float)raw_dark;
comp_uvb = (float)raw_uv - (float)raw_dark;

comp_uva -= (VEML6075_UVI_UVA_VIS_COEFF * comp_vis) - (VEML6075_UVI_UVA_IR_COEFF * comp_ir);
comp_uvb -= (VEML6075_UVI_UVB_VIS_COEFF * comp_vis) - (VEML6075_UVI_UVB_IR_COEFF * comp_ir);

uva_weighted = comp_uva*VEML6075_UVI_UVA_RESPONSE;
uvb_weighted = comp_uv - (float)raw_dark*VEML6075_UVI_UVB_RESPONSE;

reading->uva = comp_uva;
reading->uvb = comp_uv;
reading->uvindex = (uva_weighted + uvb_weighted) / 2.0;

return status;
}

```

## Anexo C6. Código del sensor CCS811 (CCS811.c)

```

#include "CCS811.h"

int8_t ccs811_init(struct ccs811_dev *dev)
{
    int8_t rslt;
    uint8_t data[4] = {0x11, 0xE5, 0x72, 0x8A}; //Reset key

    /* chip id read try count */
    uint8_t try_count = 5;
    uint8_t chip_id = 0;

    while (try_count)
    {
        dev->delay_ms(50);
        rslt = dev->read(dev->dev_id, CSS811_HW_ID_ADDR, &chip_id, 1);

        /* Check for chip id validity */
        if ((rslt == CCS811_OK) && (chip_id == CSS811_HW_ID))
        {
            dev->chip_id = chip_id;
            // sw reset

            rslt = dev->write(dev->dev_id, CSS811_SW_RESET, data, 4);
            dev->delay_ms(100);
            /* Start sensor */

            rslt = dev->write(dev->dev_id, CSS811_APP_START_ADDR, 0, 0);
            dev->delay_ms(70);

            if (rslt == CCS811_OK)
            {
                /* Set mode */
                rslt = dev->write(dev->dev_id, CSS811_MEAS_MODE_ADDR, &dev->mode, 1);
                dev->delay_ms(50);
            }
            break;
        }

        /* Wait for 1 ms */
        dev->delay_ms(100);
        --try_count;
    }

    /* Chip id check failed */
    if (!try_count)
    {
        rslt = 5; //BME280_E_DEV_NOT_FOUND;
    }

    return rslt;
}

int8_t ccs811_set_EnvironmentalData(uint32_t relativeHumidity, uint32_t temperature, struct
ccs811_dev *dev)
{
    uint8_t envData[4];
    uint8_t rslt = 0;

    envData[0] = (relativeHumidity + 250) / 500;
    envData[1] = 0; //CCS811 only supports increments of 0.5 so bits 7-0 will always be zero

    temperature += 25000; //Add the 25C offset

    envData[2] = (temperature + 250) / 500;
    envData[3] = 0;

    rslt = dev->write(dev->dev_id, CSS811_ENV_DATA, envData, 4);

    return rslt;
}

int8_t ccs811_get_baseline(struct ccs811_dev *dev, struct ccs811_data *dataObj)
{
    uint8_t baselineReg[2];
    uint8_t rslt;

```



```

rslt = dev->read(dev->dev_id, CSS811_BASELINE_ADDR, baselineReg, 2);

uint16_t baselineNew = ((uint16_t)baselineReg[0] << 8) | baselineReg[1];

dataObj->baseline = baselineNew;
return 1;
}
int8_t ccs811_set_baseline(struct ccs811_dev *dev, struct ccs811_data *dataObj)
{
    uint8_t rslt = 0;
    uint16_t hardBaseline = 13449;
    uint8_t data[2];

    // data[0] = (uint8_t)((dev->baseline >> 8) & 0x00FF); //TODO do something about dynamic
    // data[1] = (uint8_t)(dev->baseline & 0x00FF);

    data[0] = (uint8_t)((hardBaseline >> 8) & 0x00FF); //TODO do something about dynamic
    data[1] = (uint8_t)(hardBaseline & 0x00FF);
    rslt = dev->write(dev->dev_id, CSS811_BASELINE_ADDR, data, 2);
    return rslt;
}
int8_t ccs811_get_data(struct ccs811_dev *dev, struct ccs811_data *data)
{
    int8_t rslt;
    bool dataReady = false;
    uint32_t tvoc = 0;
    uint32_t CO2 = 0;

    //! Debug Variables
    static uint32_t dataareadycntr = 0;
    static uint32_t tryReadCntr = 0;
    //!

    /* Array to store the eco2, tvoc */
    uint8_t alg_data[4] = {0};

    //! Debug Variables
    tryReadCntr++;

    /* Read the pressure and temperature data from the sensor */

    dataReady = isDataReady(dev);

    if ((dataReady))
    {
        rslt = dev->read(dev->dev_id, CSS811_ALG_RESULT_ADDR, alg_data, 4);

        uint8_t co2MSB = alg_data[0];
        uint8_t co2LSB = alg_data[1];
        uint8_t tvocMSB = alg_data[2];
        uint8_t tvocLSB = alg_data[3];
        CO2 = ((uint32_t)co2MSB << 8) | co2LSB;
        tvoc = ((uint32_t)tvocMSB << 8) | tvocLSB;
        data->eCO2 = CO2;
        data->VOCS = tvoc;

        //! Debug Variables
        dataareadycntr++;
        /* Parse the read data from the sensor */
    }

    dev->delay_ms(1);
    return rslt;
}
bool ccs811_check_error(struct ccs811_dev *dev)
{
    uint8_t reg_data[4] = {0};
    uint8_t statusReg = 0;
    uint8_t errorId = 0;
    bool isErrorClear = true;
    static uint32_t errorCounter = 0;

    dev->read(dev->dev_id, CSS811_STATUS_ADDR, &statusReg, 1);
    if (((statusReg & 0x1) << 0) == 1)
    {
        dev->read(dev->dev_id, CSS811_ERROR_ID_ADDR, &errorId, 1);
    }
}

```

```
        isErrorClear = false;
        errorCounter++;
    }

    return isErrorClear;
}
static bool isDataReady(struct ccs811_dev *dev)
{
    bool dataReady = false;
    uint8_t rslt = 0;
    uint8_t status = 0;

    rslt = dev->read(dev->dev_id, CSS811_STATUS_ADDR, &status, 1);
    if (ccs811_check_error)
    {
        dataReady = (status >> 3) & (0x1);
    }

    return dataReady;
}
```

## Anexo C7. Código del sensor VEML7700 (VEML7700.c)

```

#include "VEML7700.h"

#define ADDR_VEML7700 0x10

#define CMD_ALS_CONF      0x00
#define CMD_PWR_SAVINGS  0x03
#define CMD_ALS           0x04

extern I2C_HandleTypeDef hi2c1;

//Configuración luz ambiente
void setALSConf(uint16_t conf)
{
    uint8_t data[3] ;
    data[0] = CMD_ALS_CONF ;
    data[1] = conf & 0xFF ;
    data[2] = (conf >> 8) & 0xFF ;
    writeRegs(data, 3) ;
}

//Modo de bajo consumo
void setPowerSaving(uint16_t ps)
{
    uint8_t data[3] ;
    data[0] = CMD_PWR_SAVINGS ;
    data[1] = ps & 0xFF ;
    data[2] = (ps >> 8) & 0xFF ;
    writeRegs(data, 3) ;
}

//Leer luz ambiente
uint16_t getALS(void)
{
    uint16_t als = 0 ;
    uint8_t cmd = CMD_ALS ;
    uint8_t data[2] ;
    readRegs(cmd, data, 2) ;
    als = (data[1] << 8) | data[0] ;
    return( als ) ;
}

//Lectura I2C
void readRegs(uint8_t addr, uint8_t * data, int len)
{
    HAL_I2C_Master_Transmit( &hi2c1, (uint16_t)(ADDR_VEML7700<<1), &addr, 1, 100 );
    HAL_Delay(1);
    HAL_I2C_Master_Receive( &hi2c1, (uint16_t)(ADDR_VEML7700<<1), data, len, 1000 );
}

//Escritura I2C
void writeRegs(uint8_t * data, int len)
{
    HAL_I2C_Master_Transmit( &hi2c1, (uint16_t)(ADDR_VEML7700<<1), data, len, 1000 );
}

```

## Anexo C8. Código del receptor GPS (GPS.c)

```

#include "GPS.h"
#include "uart.h"
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define UART_GPS                                huart3
#define _GPS_DEBUG                              0

GPS_t GPS;

double grados_decimal(float grados)
{
    double minimo = 0.0;
    double decimal_grados = 0.0;

    minimo = fmod((double)grados, 100.0);

    grados = (int) ( grados / 100 );
    decimal_grados = grados + ( minimo / 60 );

    return decimal_grados;
}

void Config_GPS(void)
{
    GPS.indiceRecepcion=0;
    HAL_UART_Receive_IT(&UART_GPS,&GPS.rxTmp,1);
}

void GPS_CallBack(void)
{
    GPS.LastTime=HAL_GetTick();
    if(GPS.indiceRecepcion < sizeof(GPS.rxBuffer)-2)
    {
        GPS.rxBuffer[GPS.indiceRecepcion] = GPS.rxTmp;
        GPS.indiceRecepcion++;
    }
    HAL_UART_Receive_IT(&UART_GPS,&GPS.rxTmp,1);
}

void GPS_data(void)
{
    if( (HAL_GetTick()-GPS.LastTime>50) && (GPS.indiceRecepcion>0))
    {
        char *str;
        #if (_GPS_DEBUG==1)
        printf("%s",GPS.rxBuffer);
        #endif
        str=strstr((char*)GPS.rxBuffer,"$GPGGA,");
        if(str!=NULL)
        {
            memset(&GPS.GPGGA,0,sizeof(GPS.GPGGA));

            sscanf(str,"$GPGGA,%2hhd%2hhd%2hhd.%3hd,%f,%c,%f,%c,%hhd,%hhd,%f,%f,%c,%hd,%s,%%2s\r\n",&GPS
            .GPGGA.UTC_Hour,&GPS.GPGGA.UTC_Min,&GPS.GPGGA.UTC_Sec,&GPS.GPGGA.UTC_MicroSec,&GPS.GPGGA.Latitude,&G
            PS.GPGGA.NS_Indicator,&GPS.GPGGA.Longitude,&GPS.GPGGA.EW_Indicator,&GPS.GPGGA.PositionFixIndicator,&
            GPS.GPGGA.SatellitesUsed,&GPS.GPGGA.HDOP,&GPS.GPGGA.MSL_Altitude,&GPS.GPGGA.MSL_Units,&GPS.GPGGA.Age
            ofDiffCorr,&GPS.GPGGA.DiffRefStationID,&GPS.GPGGA.CheckSum);
            if(GPS.GPGGA.NS_Indicator==0)
                GPS.GPGGA.NS_Indicator='-';
            if(GPS.GPGGA.EW_Indicator==0)
                GPS.GPGGA.EW_Indicator='-';
            if(GPS.GPGGA.Geoid_Units==0)
                GPS.GPGGA.Geoid_Units='-';
            if(GPS.GPGGA.MSL_Units==0)
                GPS.GPGGA.MSL_Units='-';
            GPS.GPGGA.LatitudeDecimal=grados_decimal(GPS.GPGGA.Latitude);
            GPS.GPGGA.LongitudeDecimal=grados_decimal(GPS.GPGGA.Longitude);
        }
        memset(GPS.rxBuffer,0,sizeof(GPS.rxBuffer));
    }
}

```

```
        GPS.indiceRecepcion=0;
    }
    HAL_UART_Receive_IT(&UART_GPS,&GPS.rxTmp,1);
}
```

## Anexo D. Código HTML de la página web (index.html)

```

<!DOCTYPE html>
<html lang="en">

  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />
    <meta name="description" content="" />
    <meta name="author" content="" />
    <title>TFM</title> <!--Titulo de la página-->
    <link rel="icon" type="image/x-icon" href="/static/assets/favicon.ico" />
    <!-- Core theme CSS (includes Bootstrap)-->
    <link href="/static/css/styles.css" rel="stylesheet" />
  </head>

  <body id="page-top">
    <!--Barra de navegación-->
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark fixed-top" id="mainNav">
      <div class="container px-4">
        <a class="navbar-brand" href="#page-top">TFM</a>
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-
target="#navbarResponsive" aria-controls="navbarResponsive" aria-expanded="false" aria-label="Toggle
navigation"><span class="navbar-toggler-icon"></span></button>
        <div class="collapse navbar-collapse" id="navbarResponsive">
          <ul class="navbar-nav ms-auto">
            <li class="nav-item"><a class="nav-link" href="#mapa">Mapa</a></li>
            <li class="nav-item"><a class="nav-link"
href="#temperatura">Temperatura</a></li>
            <li class="nav-item"><a class="nav-link" href="#humedad">Humedad</a></li>
            <li class="nav-item"><a class="nav-link"
href="#presion">Presión</a></li>
            <li class="nav-item"><a class="nav-link"
href="#luminosidad">Luminosidad</a></li>
            <li class="nav-item"><a class="nav-link"
href="#uv">Radiación UV</a></li>
            <li class="nav-item"><a class="nav-link"
href="#co2">CO2</a></li>
            <li class="nav-item"><a class="nav-link"
href="#tvoc">TVOC</a></li>
            <li class="nav-item"><a class="nav-link"
href="#lluvia">Lluvia</a></li>
          </ul>
        </div>
      </div>
    </nav>

    <!--Cabecera-->
    <header class="bg-primary bg-gradient text-white">
      <div class="container px-4 text-center">
        <h1 class="fw-bolder">Trabajo Fin de Máster</h1>
        <p class="lead">Desarrollo de un sistema remoto de adquisición de datos basado en
LoRaWAN para aplicaciones IoT</p>
        <p class="lead"></p>
        <p class="lead">Autor: Sergio Lluva Plaza</p>
        <p class="lead">Tutor: José Manuel Villadangos Carrizo</p>
      </div>
    </header>

    <!-- GPS-->
    <section id="mapa">
      <div class="container px-4">
        <div class="row gx-4 justify-content-center">
          <div class="col-lg-8">
            <h2>Posición GPS del nodo</h2>
            <iframe class="map", src="/get_map" width="810"
height="600"></iframe>
          </div>
        </div>
      </div>
    </section>

    <!-- Temperatura-->
    <section class="bg-light" id="temperatura">
      <div class="container px-4">
        <div class="row gx-4 justify-content-center">

```

```

        <div class="col-lg-8">
            <h2>Temperatura</h2>
            <iframe class="grafico_temperatura",
src="/get_grafico_temperatura" width="900" height="700"></iframe>
        </div>
    </div>
</section>

<!--Humedad-->
<section id="humedad">
    <div class="container px-4">
        <div class="row gx-4 justify-content-center">
            <div class="col-lg-8">
                <h2>Humedad</h2>
                <iframe class="grafico_humedad",
src="/get_grafico_humedad" width="900" height="700"></iframe>
            </div>
        </div>
    </div>
</section>

<!--Presión-->
<section class="bg-light" id="presion">
    <div class="container px-4">
        <div class="row gx-4 justify-content-center">
            <div class="col-lg-8">
                <h2>Presión atmosférica</h2>
                <iframe class="grafico_presion",
src="/get_grafico_presion" width="900" height="700"></iframe>
            </div>
        </div>
    </div>
</section>

<!--Luminosidad-->
<section id="luminosidad">
    <div class="container px-4">
        <div class="row gx-4 justify-content-center">
            <div class="col-lg-8">
                <h2>Luminosidad</h2>
                <iframe class="grafico_luminosidad",
src="/get_grafico_luminosidad" width="900" height="700"></iframe>
            </div>
        </div>
    </div>
</section>

<!--Radiación UV-->
<section class="bg-light" id="uv">
    <div class="container px-4">
        <div class="row gx-4 justify-content-center">
            <div class="col-lg-8">
                <h2>Radiación UV</h2>
                <iframe class="grafico_uv", src="/get_grafico_uv"
width="900" height="700"></iframe>
            </div>
        </div>
    </div>
</section>

<!--CO2-->
<section id="co2">
    <div class="container px-4">
        <div class="row gx-4 justify-content-center">
            <div class="col-lg-8">
                <h2>Nivel de CO2</h2>
                <iframe class="grafico_co2", src="/get_grafico_co2"
width="900" height="700"></iframe>
            </div>
        </div>
    </div>
</section>

<!--TVOC-->
<section class="bg-light" id="tvoc">
    <div class="container px-4">

```

```

        <div class="row gx-4 justify-content-center">
            <div class="col-lg-8">
                <h2>Nivel de TVOC</h2>
                <iframe class="grafico_tvoc", src="/get_grafico_tvoc"
width="900" height="700"></iframe>
            </div>
        </div>
    </div>
</section>

<!--Lluvia-->
<section id="lluvia">
    <div class="container px-4">
        <div class="row gx-4 justify-content-center">
            <div class="col-lg-8">
                <h2>Lluvia</h2>
                <iframe class="grafico_lluvia",
src="/get_grafico_lluvia" width="900" height="700"></iframe>
            </div>
        </div>
    </div>
</section>

<!--Pie de página-->
<footer class="py-5 bg-dark">
    <div class="container px-4"><p class="m-0 text-center text-white">Sergio Lluva
Plaza</p></div>
</footer>

<!-- Bootstrap core JS-->
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.0/dist/js/bootstrap.bundle.min.js"></script>
<!-- Core theme JS-->
<script src="/static/js/scripts.js"></script>

<!-- Añadido para highcharts-->
<script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.8.2/jquery.min.js"></script>
<script src="http://code.highcharts.com/highcharts.js"></script>
<script src="../static/js/main.js"></script>

</body>
</html>

```



## Anexo E. Código de la aplicación Flask (app.py)

```

from flask import Flask, render_template, make_response
from influxdb import InfluxDBClient
from highcharts import Highchart
import folium
import random
import os
import shutil

app = Flask(__name__)

@app.route('/')
@app.route('/index')
def index():

    #Obtener datos de InfluxDB
    user = 'admin'
    password = 'admin'
    dbname = 'chirpstack'
    dbuser = 'smlly'
    dbuser_password = 'my_secret_password'

    query_latitud = 'select * from device_frmpayload_data_latitud group by * order by desc limit 1;'
    query_longitud = 'select * from device_frmpayload_data_longitud group by * order by desc limit
1;'
    query_temperatura = 'select value from device_frmpayload_data_mydata;'
    query_humedad = 'select value from device_frmpayload_data_humedad;'
    query_presion = 'select value from device_frmpayload_data_presion;'
    query_luminosidad = 'select value from device_frmpayload_data_luminosidad;'
    query_uva = 'select value from device_frmpayload_data_uva;'
    query_uvb = 'select value from device_frmpayload_data_uvb;'
    query_co2 = 'select value from device_frmpayload_data_co2;'
    query_tvoc = 'select value from device_frmpayload_data_tvoc;'
    query_lluvia = 'select value from device_frmpayload_data_lluvia;'

    client = InfluxDBClient('localhost', 8086, database='chirpstack')

    result_latitud = client.query(query_latitud, epoch='ns')
    result_longitud = client.query(query_longitud, epoch='ns')
    result_temperatura = client.query(query_temperatura, epoch='ns')
    result_humedad = client.query(query_humedad, epoch='ns')
    result_presion = client.query(query_presion, epoch='ns')
    result_luminosidad = client.query(query_luminosidad, epoch='ns')
    result_uva = client.query(query_uva, epoch='ns')
    result_uvb = client.query(query_uvb, epoch='ns')
    result_co2 = client.query(query_co2, epoch='ns')
    result_tvoc = client.query(query_tvoc, epoch='ns')
    result_lluvia = client.query(query_lluvia, epoch='ns')

    points_latitud = list(result_latitud.get_points(measurement='device_frmpayload_data_latitud'))
    points_longitud =
list(result_longitud.get_points(measurement='device_frmpayload_data_longitud'))
    points_temperatura =
list(result_temperatura.get_points(measurement='device_frmpayload_data_temperatura'))
    points_humedad = list(result_humedad.get_points(measurement='device_frmpayload_data_humedad'))
    points_presion = list(result_presion.get_points(measurement='device_frmpayload_data_presion'))
    points_luminosidad =
list(result_luminosidad.get_points(measurement='device_frmpayload_data_luminosidad'))
    points_uva = list(result_uva.get_points(measurement='device_frmpayload_data_uva'))
    points_uvb = list(result_uvb.get_points(measurement='device_frmpayload_data_uvb'))
    points_co2 = list(result_co2.get_points(measurement='device_frmpayload_data_co2'))
    points_tvoc = list(result_tvoc.get_points(measurement='device_frmpayload_data_tvoc'))
    points_lluvia = list(result_lluvia.get_points(measurement='device_frmpayload_data_lluvia'))

    data_latitud = points_latitud[x].get('value')

    data_longitud = points_longitud[x].get('value')

    for x in range(len(points_temperatura)):
        data_temperatura.append([points_temperatura[x].get('time'),
int(points_temperatura[x].get('value'))])

    for x in range(len(points_humedad)):
        data_humedad.append([points_humedad[x].get('time'), int(points_humedad[x].get('value'))])

```

```

for x in range(len(points_presion)):
    data_presion.append([points_presion[x].get('time'), int(points_presion[x].get('value'))])

for x in range(len(points_luminosidad)):
    data_luminosidad.append([points_luminosidad[x].get('time'),
int(points_luminosidad[x].get('value'))])

for x in range(len(points_uva)):
    data_uva.append([points_uva[x].get('time'), int(points_uva[x].get('value'))])

for x in range(len(points_uvuv)):
    data_uvuv.append([points_uvuv[x].get('time'), int(points_uvuv[x].get('value'))])

for x in range(len(points_co2)):
    data_co2.append([points_co2[x].get('time'), int(points_co2[x].get('value'))])

for x in range(len(points_tvoc)):
    data_tvoc.append([points_tvoc[x].get('time'), int(points_tvoc[x].get('value'))])

for x in range(len(points_lluvia)):
    data_lluvia.append([points_lluvia[x].get('time'), int(points_lluvia[x].get('value'))])

#Mapa
if data_latitud==0&&data_longitud==0:
    data_latitud = latitud_anterior;
    data_longitud = longitud_anterior;

data_latitud = latitud_anterior
data_longitud = longitud_anterior
coordenadas = (data_latitud, data_longitud)

folium_map = folium.Map(
    location = coordenadas,
    zoom_start = 17
)
folium.TileLayer('openstreetmap').add_to(folium_map)
folium.Marker(
    [latitud, longitud],
    popup = "Nodo LoRaWAN",
    icon = folium.Icon(color="red")
).add_to(folium_map)
folium_map.save('templates/map.html')

#Temperatura
H_temperatura = Highchart()
H_temperatura.set_options('chart', {'zoomType': 'x'})
H_temperatura.set_options('xAxis', {'type': 'datetime'})
H_temperatura.set_options('yAxis', {'title': {'text': '°C'}})
H_temperatura.set_options('legend', {'enabled': False})
H_temperatura.set_options('title', {'text': 'Temperatura'})
H_temperatura.set_options('legend', {'enabled': False})
H_temperatura.set_options('credits', {'enabled': False})
H_temperatura.add_data_set(data_temperatura, 'line', 'Temperatura')
H_temperatura.set_options('plotOptions', {
    'line': {
        }
    })
H_temperatura.htmlcontent
H_temperatura.save_file('templates/grafico_temperatura')

#Humedad
H_humedad = Highchart()
H_humedad.set_options('chart', {'zoomType': 'x'})
H_humedad.set_options('xAxis', {'type': 'datetime'})
H_humedad.set_options('yAxis', {'title': {'text': '%'}})
H_humedad.set_options('legend', {'enabled': False})
H_humedad.set_options('title', {'text': 'Humedad'})
H_humedad.set_options('legend', {'enabled': False})
H_humedad.set_options('credits', {'enabled': False})
H_humedad.add_data_set(data_humedad, 'line', 'Humedad')
H_humedad.set_options('plotOptions', {
    'line': {
        }
    })
H_humedad.htmlcontent
H_humedad.save_file('templates/grafico_humedad')

```

```

H_humedad.htmlcontent
H_humedad.save_file('templates/grafico_humedad')

#Presion
H_presion = Highchart()
H_presion.set_options('chart', {'zoomType': 'x'})
H_presion.set_options('xAxis', {'type': 'datetime'})
H_presion.set_options('yAxis', {'title': {'text': 'hPa'}})
H_presion.set_options('legend', {'enabled': False})
H_presion.set_options('title', {'text': 'Presión'})
H_presion.set_options('legend', {'enabled': False})
H_presion.set_options('credits', {'enabled': False})
H_presion.add_data_set(data_presion, 'line', 'Presión')
H_presion.set_options('plotOptions', {
    'line': {
        }
    })
H_presion.htmlcontent
H_presion.save_file('templates/grafico_presion')

#Luminosidad
H_luminosidad = Highchart()
H_luminosidad.set_options('chart', {'zoomType': 'x'})
H_luminosidad.set_options('xAxis', {'type': 'datetime'})
H_luminosidad.set_options('yAxis', {'title': {'text': 'lux'}})
H_luminosidad.set_options('legend', {'enabled': False})
H_luminosidad.set_options('title', {'text': 'Luminosidad'})
H_luminosidad.set_options('legend', {'enabled': False})
H_luminosidad.set_options('credits', {'enabled': False})
H_luminosidad.add_data_set(data_presion, 'line', 'Luminosidad')
H_luminosidad.set_options('plotOptions', {
    'line': {
        }
    })
H_luminosidad.htmlcontent
H_luminosidad.save_file('templates/grafico_Luminosidad')

#Radiación UV
H_uv = Highchart()
H_uv.set_options('chart', {'zoomType': 'x'})
H_uv.set_options('xAxis', {'type': 'datetime'})
H_uv.set_options('yAxis', {'title': {'text': 'UV'}})
H_uv.set_options('legend', {'enabled': False})
H_uv.set_options('title', {'text': 'UV'})
H_uv.set_options('legend', {'enabled': False})
H_uv.set_options('credits', {'enabled': False})
H_uv.add_data_set(data_uv, 'line', 'UV')
H_uv.set_options('plotOptions', {
    'line': {
        }
    })
H_uv.htmlcontent
H_uv.save_file('templates/grafico_uv')

#CO2
H_co2 = Highchart()
H_co2.set_options('chart', {'zoomType': 'x'})
H_co2.set_options('xAxis', {'type': 'datetime'})
H_co2.set_options('yAxis', {'title': {'text': 'ppm'}})
H_co2.set_options('legend', {'enabled': False})
H_co2.set_options('title', {'text': 'CO2'})
H_co2.set_options('legend', {'enabled': False})
H_co2.set_options('credits', {'enabled': False})
H_co2.add_data_set(data_co2, 'line', 'CO2')
H_co2.set_options('plotOptions', {
    'line': {
        }
    })
H_co2.htmlcontent
H_co2.save_file('templates/grafico_co2')

#TVOC

```

```

H_tvoc = Highchart()
H_tvoc.set_options('chart', {'zoomType': 'x'})
H_tvoc.set_options('xAxis', {'type': 'datetime'})
H_tvoc.set_options('yAxis', {'title': {'text': 'ppb'}})
H_tvoc.set_options('legend', {'enabled': False})
H_tvoc.set_options('title', {'text': 'TVOC'})
H_tvoc.set_options('legend', {'enabled': False})
H_tvoc.set_options('credits', {'enabled': False})
H_tvoc.add_data_set(data_co2, 'line', 'TVOC')
H_tvoc.set_options('plotOptions', {
    'line': {

    }
})
H_tvoc.htmlcontent
H_tvoc.save_file('templates/grafico_tvoc')

#Lluvia
H_lluvia = Highchart()
H_lluvia.set_options('chart', {'zoomType': 'x'})
H_lluvia.set_options('xAxis', {'type': 'datetime'})
H_lluvia.set_options('yAxis', {'title': {'text': ''}})
H_lluvia.set_options('legend', {'enabled': False})
H_lluvia.set_options('title', {'text': 'Lluvia'})
H_lluvia.set_options('legend', {'enabled': False})
H_lluvia.set_options('credits', {'enabled': False})
H_lluvia.add_data_set(data_co2, 'line', 'Lluvia')
H_lluvia.set_options('plotOptions', {
    'line': {

    }
})
H_lluvia.htmlcontent
H_lluvia.save_file('templates/grafico_lluvia')

return render_template('index.html')

@app.route('/get_map')
def get_map():
    r = int(random.triangular(0,100))
    t = "templates/map_{i}.html"
    for i in range(0,100):
        f = t.format(i=i)
        if os.path.exists(f):
            os.remove(f)
    f = t.format(i=r)
    shutil.copy("templates/map.html", f)

    r = make_response(render_template(os.path.split(f)[1]))
    r.cache_control.max_age = 0
    r.cache_control.no_cache = True
    r.cache_control.no_store = True
    r.cache_control.must_revalidate = True
    r.cache_control.proxy_revalidate = True
    return r

#Render gráfico temperatura
@app.route('/get_grafico_temperatura')
def get_grafico_temperatura():
    r = int(random.triangular(0,100))
    t = "templates/grafico_temperatura{i}.html"
    for i in range(0,100):
        f = t.format(i=i)
        if os.path.exists(f):
            os.remove(f)
    f = t.format(i=r)
    shutil.copy("templates/grafico_temperatura.html", f)

    r = make_response(render_template(os.path.split(f)[1]))
    r.cache_control.max_age = 0
    r.cache_control.no_cache = True
    r.cache_control.no_store = True
    r.cache_control.must_revalidate = True
    r.cache_control.proxy_revalidate = True
    return r

```

```

#Render gráfico humedad
@app.route('/get_grafico_humedad')
def get_grafico_humedad():
    r = int(random.triangular(0,100))
    t = "templates/grafico_humedad{i}.html"
    for i in range(0,100):
        f = t.format(i=i)
        if os.path.exists(f):
            os.remove(f)
    f = t.format(i=r)
    shutil.copy("templates/grafico_humedad.html", f)

    r = make_response(render_template(os.path.split(f)[1]))
    r.cache_control.max_age = 0
    r.cache_control.no_cache = True
    r.cache_control.no_store = True
    r.cache_control.must_revalidate = True
    r.cache_control.proxy_revalidate = True
    return r

#Render gráfico presión
@app.route('/get_grafico_presion')
def get_grafico_presion():
    r = int(random.triangular(0,100))
    t = "templates/grafico_presion{i}.html"
    for i in range(0,100):
        f = t.format(i=i)
        if os.path.exists(f):
            os.remove(f)
    f = t.format(i=r)
    shutil.copy("templates/grafico_presion.html", f)

    r = make_response(render_template(os.path.split(f)[1]))
    r.cache_control.max_age = 0
    r.cache_control.no_cache = True
    r.cache_control.no_store = True
    r.cache_control.must_revalidate = True
    r.cache_control.proxy_revalidate = True
    return r

#Render gráfico luminosidad
@app.route('/get_grafico_luminosidad')
def get_grafico_luminosidad():
    r = int(random.triangular(0,100))
    t = "templates/grafico_luminosidad{i}.html"
    for i in range(0,100):
        f = t.format(i=i)
        if os.path.exists(f):
            os.remove(f)
    f = t.format(i=r)
    shutil.copy("templates/grafico_luminosidad.html", f)

    r = make_response(render_template(os.path.split(f)[1]))
    r.cache_control.max_age = 0
    r.cache_control.no_cache = True
    r.cache_control.no_store = True
    r.cache_control.must_revalidate = True
    r.cache_control.proxy_revalidate = True
    return r

#Render gráfico radiación UV
@app.route('/get_grafico_uv')
def get_grafico_uv():
    r = int(random.triangular(0,100))
    t = "templates/grafico_uv{i}.html"
    for i in range(0,100):
        f = t.format(i=i)
        if os.path.exists(f):
            os.remove(f)
    f = t.format(i=r)
    shutil.copy("templates/grafico_uv.html", f)

    r = make_response(render_template(os.path.split(f)[1]))
    r.cache_control.max_age = 0
    r.cache_control.no_cache = True
    r.cache_control.no_store = True
    r.cache_control.must_revalidate = True

```

```

    r.cache_control.proxy_revalidate = True
    return r

#Render gráfico CO2
@app.route('/get_grafico_co2')
def get_grafico_co2():
    r = int(random.triangular(0,100))
    t = "templates/grafico_co2{i}.html"
    for i in range(0,100):
        f = t.format(i=i)
        if os.path.exists(f):
            os.remove(f)
    f = t.format(i=r)
    shutil.copy("templates/grafico_co2.html", f)

    r = make_response(render_template(os.path.split(f)[1]))
    r.cache_control.max_age = 0
    r.cache_control.no_cache = True
    r.cache_control.no_store = True
    r.cache_control.must_revalidate = True
    r.cache_control.proxy_revalidate = True
    return r

#Render gráfico TVOC
@app.route('/get_grafico_tvoc')
def get_grafico_tvoc():
    r = int(random.triangular(0,100))
    t = "templates/grafico_tvoc{i}.html"
    for i in range(0,100):
        f = t.format(i=i)
        if os.path.exists(f):
            os.remove(f)
    f = t.format(i=r)
    shutil.copy("templates/grafico_tvoc.html", f)

    r = make_response(render_template(os.path.split(f)[1]))
    r.cache_control.max_age = 0
    r.cache_control.no_cache = True
    r.cache_control.no_store = True
    r.cache_control.must_revalidate = True
    r.cache_control.proxy_revalidate = True
    return r

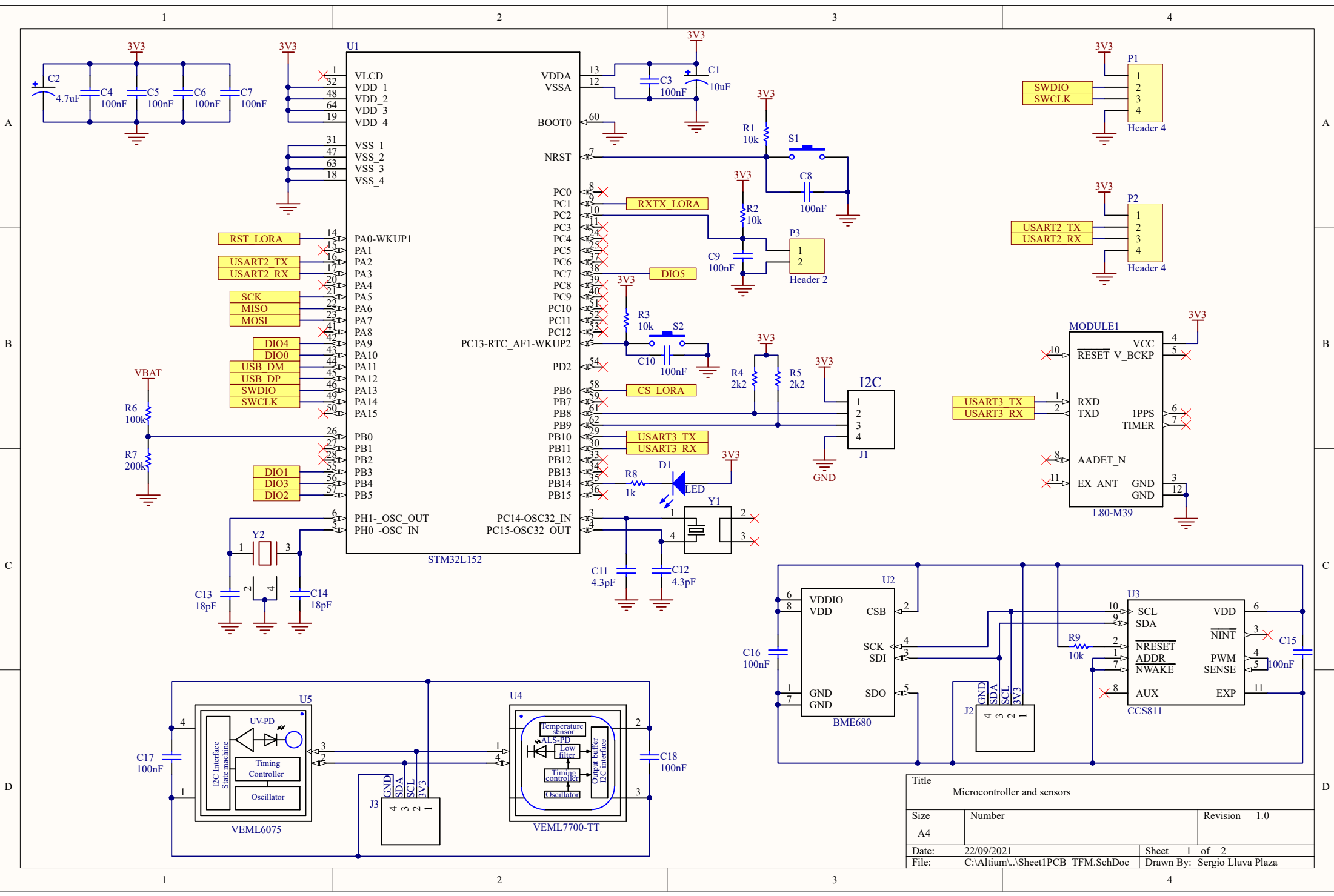
#Render gráfico lluvia
@app.route('/get_grafico_lluvia')
def get_grafico_lluvia():
    r = int(random.triangular(0,100))
    t = "templates/grafico_lluvia{i}.html"
    for i in range(0,100):
        f = t.format(i=i)
        if os.path.exists(f):
            os.remove(f)
    f = t.format(i=r)
    shutil.copy("templates/grafico_lluvia.html", f)

    r = make_response(render_template(os.path.split(f)[1]))
    r.cache_control.max_age = 0
    r.cache_control.no_cache = True
    r.cache_control.no_store = True
    r.cache_control.must_revalidate = True
    r.cache_control.proxy_revalidate = True
    return r

if __name__ == "__main__":
    app.run(debug = True, host='0.0.0.0', passthrough_errors=True)

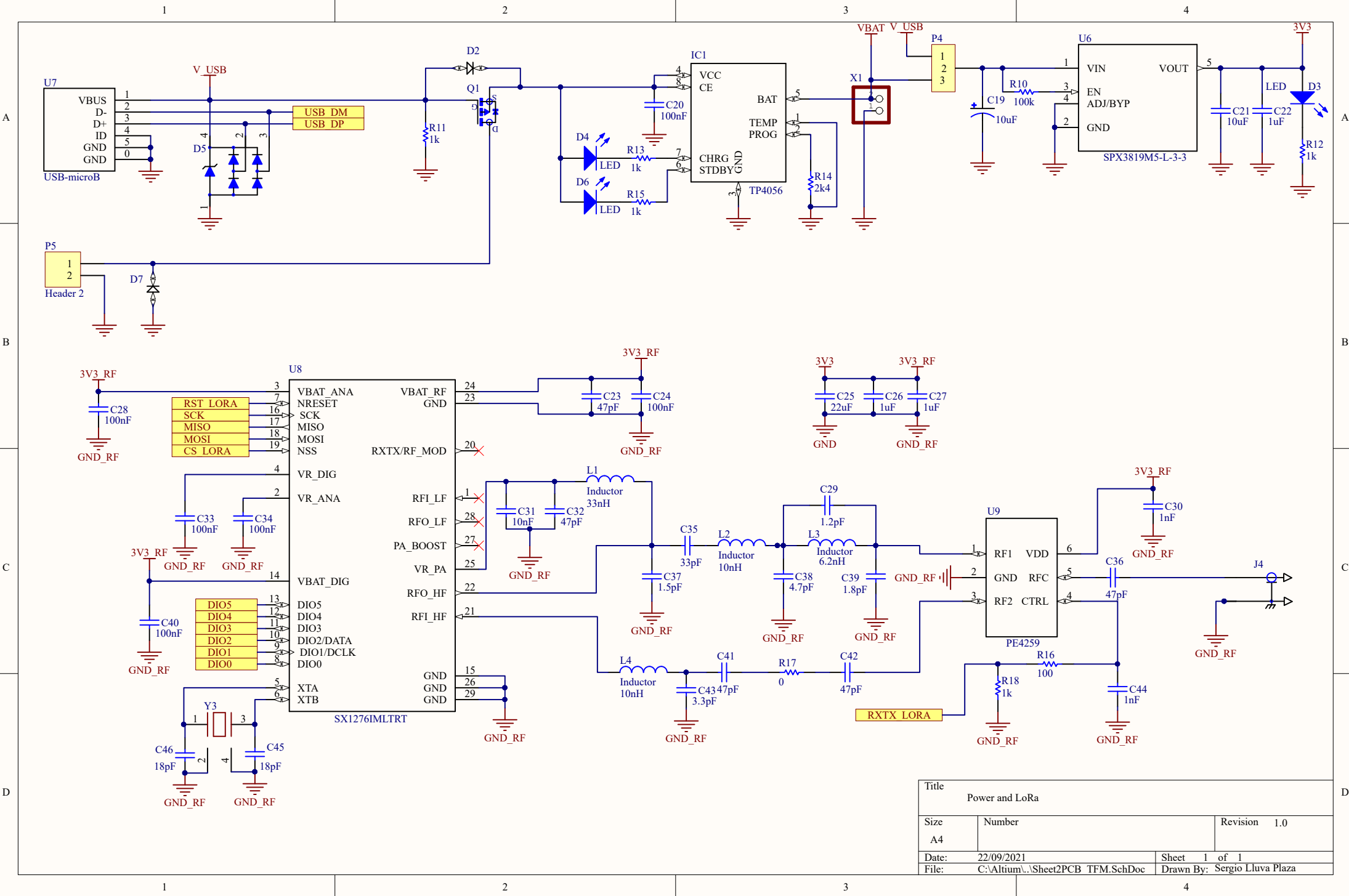
```

## Anexo F. Captura de esquema para el diseño de la PCB



Title			Microcontroller and sensors		
Size	Number		Revision		1.0
A4					
Date:	22/09/2021		Sheet		1 of 2
File:	C:\Altium\...\Sheet1PCB TFM.SchDoc		Drawn By:		Sergio Lluya Plaza



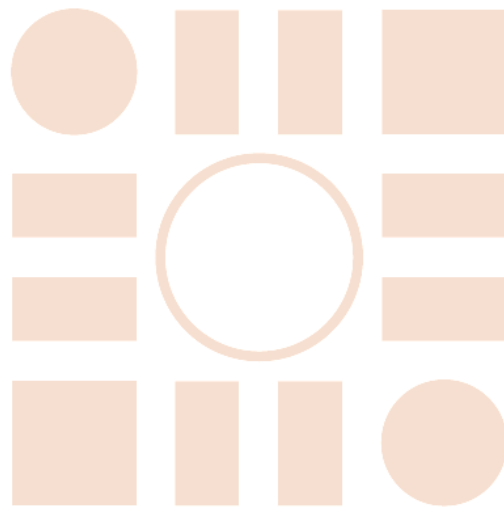


Title		
Power and LoRa		
Size	Number	Revision
A4		1.0
Date:	22/09/2021	Sheet 1 of 1
File:	C:\Altium\...\Sheet2PCB TFM.SchDoc	Drawn By: Sergio Lluya Plaza





Universidad de Alcalá  
Escuela Politécnica Superior



ESCUELA POLITECNICA  
SUPERIOR

