UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Sistema de telemedida y sensorización mediante red LoRaWAN

Autor:

D. Gonzalo Blanco Rico

Tutor:

Dr. D. Mario Martínez Zarzuela

Valladolid, 2 de Septiembre de 2021

TÍTULO:	**Sistema de telemedida y sensorización mediante red LoRaWAN**
AUTOR:	**D. Gonzalo Blanco Rico**
TUTOR:	**Dr. D. Mario Martínez Zarzuela**
DEPARTAMENTO:	**Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

TRIBUNAL

PRESIDENTE:	**Dra. Dª. Miriam Antón Rodríguez**
SECRETARIO:	**Dr. D. Mario Martínez Zarzuela**
VOCAL:	**Dr. D. David González Ortega**
SUPLENTE 1:	**Dr. D. Carlos Gómez Peña**
SUPLENTE 2:	**Dr. D. Francisco J. Díaz Pernas**

FECHA:	**2 de Septiembre de 2021**
CALIFICACIÓN:	

Agradecimientos

A mis padres y mi hermana por apoyarme en este camino.

A mi novia por estar siempre ahí y hacerlo todo más fácil.

A mis compañeros por estar siempre dispuestos a ayudar.

A la empresa Energibid por apostar por mí y por este proyecto.

Resumen

Este proyecto ha sido desarrollado en colaboración con la empresa Energibid, que cuenta con una infraestructura para la recolección de datos desde los contadores de energía de sus distintos clientes, así como una plataforma web que ofrece una gran variedad de servicios para el análisis de estos datos.

El fin de este proyecto es adaptar este sistema a otros tipos de datos, como puede ser la medición de gas, temperatura, humedad, etc. Además, permitirá la reducción de costes en cuanto a los sistemas empleados para recoger y enviar estos datos mediante Internet, ya que se abarcarán grandes áreas con el mínimo equipo necesario y una sola conexión a Internet.

Para lograr esto, se propone un sistema de comunicaciones entre uno o varios sensores y un concentrador capaz de comunicarse con Internet. La principal ventaja de este sistema es que los sensores se conectan de manera inalámbrica con el concentrador mediante la red LoRa, es decir, un enlace radio de largo alcance. Esta tecnología permite un consumo mínimo de energía para los sensores, y permite abarcar una gran área de medidas con tan solo un concentrador.

Para este desarrollo se emplearán propuestas comerciales que solucionan la conexión del enlace radio, ofreciendo así mayor rendimiento y estabilidad.

La parte a desarrollar será la lógica necesaria para la recolección de estos datos desde los sensores, el envío de estos datos al servidor implementando la capacidad para gestionar la fiabilidad en la entrega, y la disponibilidad y accesibilidad de los datos de una manera sencilla una vez estos han sido decodificados y almacenados en el servidor.

Palabras claves

LoRa, enlace radio, LPWAN, LoRaWAN, servidor de red, pasarela, API HTTP.

Abstract

This Master Thesis has been developed in collaboration with the company Energibid, which count with an infrastructure for data collection from their clients electric meters, as well as a web platform able to offer a great variety of services for analyzing this data.

The target of this project is to adapt this existing system to other type of data as gas, temperature or humidity measurements. In addition, it will allow to reduce the costs of the systems used for collecting and sending this data through the Internet due to the capability of covering wide areas with minimum equipment and only just one Internet connection.

To achieve this, it is proposed a new communication system between one or multiple sensors, and a concentrator capable of establishing connection with the Internet. The main advantage of this system is that sensors are wirelessly connected to the concentrator via the LoRa Network, which consists of long-distance radio links. This technology allows a minimum energy consumption for the sensors while offering an enormous area covered with just one concentrator.

For easing the long-distance radio links and offering superior performance and reliability, commercial solutions are employed for the development of this project.

The part to develop is the logic needed for recollecting all this data from the sensors, as well as sending it to the centralized server implementing the capability to assure its delivery and keeping this data available and accessible in an effortless way once they have been decoded and stored in the server.

Keywords

LoRa, radio link, LPWAN, LoRaWAN, network server, gateway, API HTTP.

ÍNDICE

CAPÍTULO 1:
INTRODUCCIÓN

1. INTRODUCCIÓN

1.1. MOTIVACIÓN

La empresa Energibid cuenta con un largo recorrido en el sector de la telemedida, y ofrece un gran valor añadido a los datos recogidos gracias al análisis de estos. Se basa principalmente en el ámbito eléctrico, pero gracias a las nuevas mejoras en los sistemas de análisis y control de las telemedidas, el siguiente paso es poder entrar en otras áreas y pasar a monitorizar otros parámetros como pueden ser el consumo de agua y gas, o variables ambientales como temperatura, humedad o calidad del aire.

En los últimos años, el sector del Internet de las cosas (IoT) está cobrando gran relevancia gracias al avance tecnológico [1]. La posibilidad de obtener datos desde localizaciones remotas o de difícil acceso, o desde lugares donde es difícil contar con una conexión a Internet, permite realizar análisis avanzados en multitud de ámbitos. Esta capacidad también permite automatizar tareas de gestión y control, así como optimizar procesos; reduciendo los costes de los mismos.

Por otro lado, y debido tanto a las restricciones impuestas en materia de medio ambiente como a la reciente crisis sanitaria, se ha puesto de manifiesto la importancia de contar con datos ambientales que permitan llevar un control sobre los parámetros que permiten identificar como afecta la calidad del aire en la salud de las personas.

Con el fin de abrir estos mercados a esta empresa, se opta por desarrollar un sistema que permita sentar unas bases para posteriores desarrollos en estos ámbitos.

1.2. OBJETIVOS

Para llevar a cabo el desarrollo de este sistema, se define un escenario tipo sobre el que plantear una solución óptima y adaptada a las necesidades propuestas usando las nuevas tecnologías emergentes en el ámbito IoT.

Este escenario propuesto surge a raíz de la crisis sanitaria provocada por el COVID-19, y de las medidas impuestas desde el gobierno para asegurar la asistencia a las aulas de una manera segura. Se trata de un colegio en el que es necesario monitorizar parámetros ambientales como la temperatura y la humedad, pero especialmente la calidad del aire. Esto permitirá llevar un control de la misma, permitiendo saber al personal docente cuando es necesario ventilar cada una de las aulas en función de los niveles de CO_2 registrados.

Contar con la capacidad de medir de manera objetiva esta calidad del aire permitirá no solo asegurar que cada una de las clases cuenta con una ventilación suficiente, sino que permitirá reducir el gasto en calefacción en los meses más fríos del año. Esto es alcanzable gracias a que el sistema indicará cuando es necesario ventilar, por ejemplo, abriendo las ventanas; y cuando los niveles de CO_2 son correctos y es

posible cerrarlas, reduciendo así el escape de calor y en consecuencia reduciendo el gasto en calefacción por parte del centro docente.

Uno de los objetivos bases del proyecto es que este sistema cuente con un coste reducido debido al desembolso que conlleva desplegar esta infraestructura para cada una de las aulas de un centro. Si bien es posible conectar cada uno de los sensores a un led que indique la necesidad de ventilar en cada una de las clases, es primordial que se realice un control centralizado por parte de un supervisor, además de que de cara al posterior análisis es necesario enviar estos datos a un servidor que ofrezca estos servicios. Para este envío de datos es posible conectar cada uno de los sensores a Internet de manera directa, pero existen centros cuya zona de cobertura WiFi no alcanza a cubrir todas las aulas e instalar un cable a cada sensor supone una alternativa tediosa y poco funcional.

Como alternativa a estas opciones, se opta por el uso de la tecnología LoRa, que permite establecer comunicaciones de manera inalámbrica que abarcan grandes distancias, además de resistir las interferencias que puedan existir en este tipo de ámbitos. Esta tecnología permite la comunicación entre cada sensor con un concentrador por lo que todo el colegio puede realizar el envío de datos a Internet desde un único punto. Además, existen dispositivos con chips de bajo coste que permiten implementar esta tecnología, por lo que al realizar un despliegue completo se reduce enormemente el coste total.

1.3. FASES DE DESARROLLO

Para llevar a cabo el desarrollo completo de este sistema se definen una serie de fases que permitirán estudiar la tecnología LoRa, analizar el mercado existente para este tipo de implementaciones, proponer una alternativa sólida y adaptada a las necesidades propuestas y verificar que se cumplen los requisitos propuestos mediante una serie de pruebas de funcionamiento.

1.3.1. FASE DE INVESTIGACIÓN

Inicialmente es necesario realizar un proceso de investigación acerca de la tecnología LoRa para entender su funcionamiento, así como las capacidades que ofrece a la hora de comunicar dispositivos para el envío de datos. Dado que las características de esta tecnología no son suficientes para implementar un sistema completo de comunicaciones desde los sensores hasta un servidor central, es necesario contar con capas superiores que añadan funcionalidades a esta comunicación.

También es necesario obtener información acerca del hardware disponible para implementar este tipo de tecnologías de una manera sencilla y robusta, además de ofrecer herramientas que faciliten su desarrollo y permitan adaptar estos dispositivos a las necesidades del sistema.

Finalmente, y dado que los datos han de estar almacenados y accesibles desde Internet, es necesario analizar las distintas opciones disponibles para ofrecer este servicio de una manera sencilla y compatible con multitud de implementaciones de las capas superiores del sistema**.**

1.3.2. FASE DE DESARROLLO

Una vez analizadas las distintas opciones, ha de implementarse el sistema completo siguiendo las pautas y recomendaciones existentes para cada una de las tecnologías que lo componen.

En esta fase se realiza la implementación de los dispositivos de adquisición de datos mediante las herramientas ofrecidas por Arduino, así como la configuración de los equipos de comunicaciones necesarios para el uso y despliegue de una red basada en la tecnología LoRa siguiendo el estándar propuesto por la LoRa Alliance [2].

Es necesario también configurar la interfaz que entrega los datos obtenidos mediante el sistema desarrollado de manera que las capas superiores puedan acceder a estos. Estas capas superiores serán las encargadas de implementar la lógica de negocio que permita añadir valor a todo el sistema ofreciendo un producto comercial útil y funcional.

Con el fin de presentar un prototipo completo que resulte atractivo, en esta fase se incluye el desarrollo de una aplicación que permita demostrar las capacidades del sistema en su conjunto de una manera sencilla y amigable para el usuario final.

1.3.3. FASE DE PRUEBAS

Una vez completado el proceso de desarrollo, y dado que uno de los principales requerimientos del sistema es que éste sea fiable a la hora de entregar los datos leídos desde los sensores, se lleva a cabo una fase de pruebas en la que se evalúan las capacidades del sistema bajo circunstancias adversas para la comunicación inalámbrica, ya que es el punto crítico donde se darán la mayoría de fallos.

Para asegurarse de que el sistema responde de la forma esperada ante las distintas situaciones que pueden darse en un entorno realista, se analiza el comportamiento de este bajo situaciones en las que se pierden algunos de los datos leídos, los dispositivos se quedan colgados o se cortan las comunicaciones durante un largo periodo de tiempo.

Estas pruebas tienen como objetivo adaptar el sistema de manera que, aun dándose estas situaciones, no se produzca una pérdida de datos y estos sean enviados una vez desaparezcan estas condiciones adversas.

1.4. ESTRUCTURA DE LA MEMORIA

En el capítulo dos, se expone todo el proceso que compone la fase de investigación. A lo largo de este capítulo se presentan las distintas tecnologías a utilizar, así como sus alternativas; definiendo de una manera clara las ventajas e inconvenientes a la hora de escoger la que más cumpla con los requisitos del sistema a desarrollar.

También se describe de manera simplificada los estándares LoRa y LoRaWAN, que permiten implementar una red completa que ofrezca la posibilidad de comunicarse con los distintos dispositivos que la conforman.

A lo largo del capítulo tres se describen las razones que han permitido decantarse por cada una de las tecnologías necesarias, así como aquellas que determinan los requisitos que han de cumplir los dispositivos hardware para implementar estas tecnologías.

Añadido a esto, se presentan las soluciones obtenidas para el desarrollo, configuración e implementación de las distintas capas que componen el sistema, explicando de manera precisa cual es el funcionamiento interno de cada elemento que compone estas capas. Se describen también las herramientas y el código generado para implementar este funcionamiento en los distintos dispositivos de los que consta el sistema. En este aspecto, y dado que cuenta con una complejidad añadida, se expone y describe el código necesario para que los nodos finales que componen la red sean capaces de: Leer los datos necesarios desde los sensores existentes, comunicarse mediante la red LoRaWAN para que estos puedan ser enviados a través de Internet y responder de la manera esperada ante situaciones que dificulten o imposibiliten esta comunicación inalámbrica.

Para finalizar este capítulo, se realiza una breve descripción de la aplicación móvil que permite visualizar los datos obtenidos desde el sistema. Añadiendo además una explicación del código que permite cumplir con los requisitos establecidos para el funcionamiento interno de la misma.

Finalmente, y como colofón a este proyecto, se añade un capítulo donde se engloban las conclusiones obtenidas tras la realización de este trabajo. Aquí se incluyen las dificultades encontradas a lo largo de las fases de investigación y desarrollo, así como las habilidades y conocimientos adquiridos a lo largo de estas.

Este capítulo se complementa con las posibilidades que abre este sistema de cara a sentar las bases para desarrollar una solución mejorada y extendida que permita suplir las necesidades de la empresa Energibid.

CAPÍTULO 2:
REVISIÓN DE TECNOLOGÍAS

2. REVISIÓN DE TECNOLOGÍAS

En la actualidad, y gracias al crecimiento del sector IoT, están cobrando gran relevancia las tecnologías de comunicación LPWAN, debido a que las necesidades de ancho de banda no son elevadas, pero sí lo son las necesidades de cobertura o consumo energético. En la figura 1 se agrupan las distintas tecnologías existentes que permiten comunicaciones inalámbricas en función de su velocidad de transmisión y su rango (las cuales afectan directamente al consumo energético).

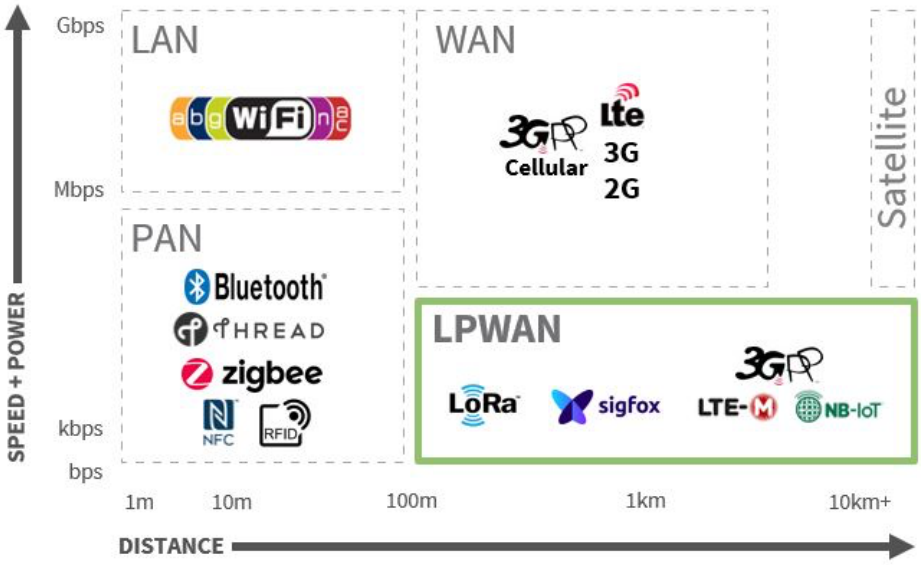


Figura 1. Tecnologías de comunicación inalámbricas según su capacidad y alcance

La tecnología LoRa ha sido desarrollada por la LoRa Alliance, en colaboración directa con Semtech, empresa propietaria de aquellas patentes que permiten diseñar e implementar chips de comunicaciones compatibles con este estándar. Respecto a otras tecnologías similares, cuenta con la ventaja de que opera en un rango de frecuencias sin licencia, por lo que sobresale como alternativa a la hora de diseñar sistemas de bajo costo. Añadido a esto, la filosofía de Semtech es diseñar chips cuyo precio sea reducido, permitiendo así explotar aún más esta ventaja que la hace destacar entre sus competidoras.

Si bien el uso de la tecnología LoRa está en crecimiento, aun no se usa de manera generalizada ya que es relativamente nueva y está en continua evolución. Esto es gracias a que existen multitud de estudios que buscan mejorar las características de esta tecnología ya sea optimizando la modulación empleada, mejorando el comportamiento del protocolo de comunicaciones LoRaWAN, o proponiendo modificaciones en el mismo para mejorar su seguridad, robustez y fiabilidad.

En cuanto al desarrollo de sistemas para la recogida de datos mediante sensores, especialmente enfocados a variables medioambientales, existen multitud de trabajos impulsados por la reciente crisis sanitaria, así como la creciente preocupación por los efectos de la polución en la salud. La mayoría de estudios y proyectos realizados buscan soluciones de bajo costo que permitan medir los parámetros ambientales necesarios para obtener la calidad del aire.

Tras la obtención de los datos, estos son enviados mediante conexiones WiFi a servidores en la nube en los cuales se presentan estos datos para su visualización o se ofrecen mediante APIs. Otra alternativa es enviar los datos captados a servidores locales en los que se presentan a través de interfaces web. Estos servidores locales suelen estar implementados en ordenadores de bajo costo como puede ser una Raspberry Pi.

En el trabajo [3] se lleva a cabo un desarrollo de un sistema de adquisición de datos medioambientales en un laboratorio de un centro escolar para su posterior estudio. Para este trabajo se emplea un microcontrolador equipado con sensores capaces de medir la calidad del aire. Este dispositivo se conecta mediante WiFi a Internet y es capaz de enviar los datos recolectados a un servidor para su posterior visualización en una aplicación móvil. Si bien el esquema de este sistema es sencillo y ofrece gran fiabilidad, su principal desventaja es que depende de la existencia de cobertura WiFi en la localización donde se desea realizar las mediciones. Además, la alimentación del microcontrolador se realiza mediante una fuente de alimentación, lo que implica más restricciones de cara a la instalación del equipo de medida.

En el estudio [4] se identifican los problemas más comunes de este tipo de sistemas. Entre estos destacan los relacionados al dispositivo de medida, es decir, los sensores. Estos producen alteraciones en las medidas debido a su construcción como pueden ser el rango de funcionamiento, la respuesta no lineal o la deriva tras la degradación del equipo. En este estudio se incluye también una comparativa entre las diferentes tecnologías de comunicación disponibles para el envío de los datos obtenidos como pueden ser WiFi, ZigBee o LoRa. Se comparan también las diferentes alternativas para la capa de aplicación y transporte entre las que destaca el protocolo MQTT, el cual es considerado el estándar de facto para las comunicaciones con dispositivos IoT.

Debido a que este proyecto busca ofrecer una alternativa al sistema ya existente para la recolección de datos, se ha hecho un gran esfuerzo en buscar tecnologías que se adapten a las necesidades propuestas, y permitan facilitar tareas como el despliegue o mantenimiento de equipos. También supone un factor de peso el coste que puede tener la instalación de un sistema de telemedida.

Por estas razones se opta por explotar la tecnología LoRa, la cual ha sido desarrollada con el objetivo de reducir los costes operacionales, así como el coste energético. Esta decisión permitirá desplegar múltiples puntos de sensorización dentro de un área extenso limitando el coste asociado a las comunicaciones en aquellos lugares donde sea necesario emplear redes celulares. También permitirá reducir los costes

asociados a cada uno de los dispositivos de lectura de medidas dado que implementan chips LoRa, los cuales han sido diseñados de manera que su precio es mínimo.

2.1. LORA (CAPA FÍSICA)

LoRa es una tecnología de modulación de radio frecuencias desarrollada por SemTech. Esta empresa diseña y ensambla distintos chips que la implementan, por lo que existe una gran variedad de hardware compatible con la tecnología LoRa. El objetivo de esta empresa es estandarizar las comunicaciones *LPWAN (Low Power Wide Area Network)*, ya que LoRa ofrece comunicaciones de varios kilómetros bajo condiciones favorables. Además, cuenta con unos requisitos de consumo energético muy bajos, lo que hace a esta tecnología ideal para ser utilizada en lugares remotos, o de difícil acceso, con alimentación mediante una batería que puede llegar a durar años en algunos casos.

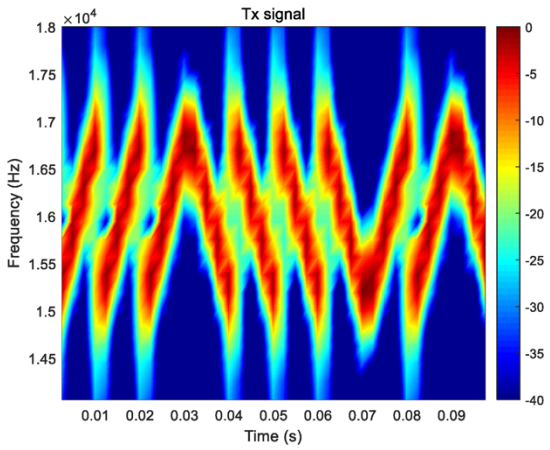


Figura 2. Barridos de frecuencia en la modulación LoRa

Está basada en la tecnología de modulación *Chirp Spread Spectrum* (CSS), que utiliza pulsos que realizan un barrido lineal de frecuencias (denominados *chirps*) utilizando un ancho de banda muy superior al que necesitaría bajo otras tecnologías [5]. Este tipo de modulación puede observarse en la figura 2, donde se aprecia claramente como es el envío de datos mediante la variación en el barrido de frecuencias realizado. Este tipo de modulación, anteriormente usada en radares, proporciona cobertura de varios kilómetros y alta tolerancia a las interferencias gracias a la tecnología de espectro ensanchado, que ofrece una alta tolerancia frente al ruido y las interferencias. Opera en una banda de frecuencias libre, situada en los 868 MHz en Europa, evitando así el sobrecoste que conlleva una licencia de uso del espectro de radiofrecuencias.

LoRa optimiza al máximo el consumo energético gracias al uso de factores de dispersión ortogonales, permitiendo así adaptar las características de transmisión de cada nodo en función de sus condiciones. Se definen así 6 posibles factores de dispersión

(nombrados desde SF7 hasta SF12), los cuales determinan la tasa de transmisión, la distancia de cobertura y la sensibilidad del receptor, alterando en consecuencia otros factores como puede ser el consumo energético o el tiempo de transmisión. Cabe destacar que debido a que los factores de dispersión son ortogonales, los datos enviados mediante uno de los factores de dispersión disponibles no interfieren con el resto.

En cuanto a la banda de frecuencias a utilizar, la ETSI la define en los 433 MHz, aunque en la práctica esta varía según la región. En Europa se encuentra en los 868 MHz, mientras que en América son 915 MHz y en China y otras regiones tanto 433 MHz como 470 MHz [6]. La regulación sobre el uso del espectro electromagnético de cada zona limita tanto la potencia como la cantidad de datos a transmitir ya que impone un máximo de tiempo de comunicación para evitar saturar el espectro. En Europa esta se define en el rango de 0,1 y 1% del día, en función del canal de transmisión, ofreciendo así un tiempo de envío máximo inferior a 15 minutos al día.

Esta tecnología permite definir una capa física de comunicaciones entre varios nodos, siguiendo un esquema punto a punto o punto a multipunto. El envío de información se hace en modo *broadcast*, ya que no permite direccionar las transmisiones hacia uno o varios receptores. Es por esto que usar esta tecnología sin extender su funcionamiento limita enormemente las capacidades esperadas para una comunicación eficiente, fiable y segura.

Para crear un sistema complejo que ofrezca funcionalidades añadidas basándose en las ventajas que ofrece LoRa, han de desarrollarse capas superiores. Especialmente la capa de enlace, que siguiendo el esquema OSI es la encargada de garantizar la seguridad, la entrega de datos, etc.

2.2. LORAWAN (CAPA DE ENLACE)

LoRaWAN es un estándar que define una arquitectura y un protocolo de comunicaciones basados en la tecnología LoRa, la capa física de radiofrecuencias que permite enlaces de largo alcance.

Se encarga de la gestión de la seguridad, ofreciendo distintos métodos para la autenticación de dispositivos, y mediante la encriptación de los datos enviados. También garantiza la fiabilidad en la entrega de datos, gracias al uso de asentimientos emitidos desde el servidor cada vez que se recibe un dato.

La estructura de comunicaciones en este estándar define por un lado los dispositivos situados al borde de la red con capacidad para enviar y recibir mensajes LoRa, y por otro lado un servidor centralizado capaz de comunicarse con cada uno de estos dispositivos. De esta manera, se establece un enlace *half-duplex* en el que se distinguen:

- ***Uplink***: Donde el dispositivo es el emisor y el servidor el receptor.
- ***Downlink***: En la cual el dispositivo recibe el mensaje enviado por el servidor.

Una de las principales ventajas de este estándar es que ofrece mayor flexibilidad a la hora del despliegue, ya que los dispositivos en el borde se conectan directamente con un servidor central, por lo que independientemente de donde se instalen, siempre existirá una comunicación directa.

2.2.1. ARQUITECTURA DE RED

El estándar LoRaWAN define una arquitectura para implementar esta tecnología. En este estándar se encapsula la lógica esperada de la capa de red dentro de una capa de enlace. Esto es debido a que para la entrega de mensajes se produce un reenvío entre los elementos que conforman esta red.

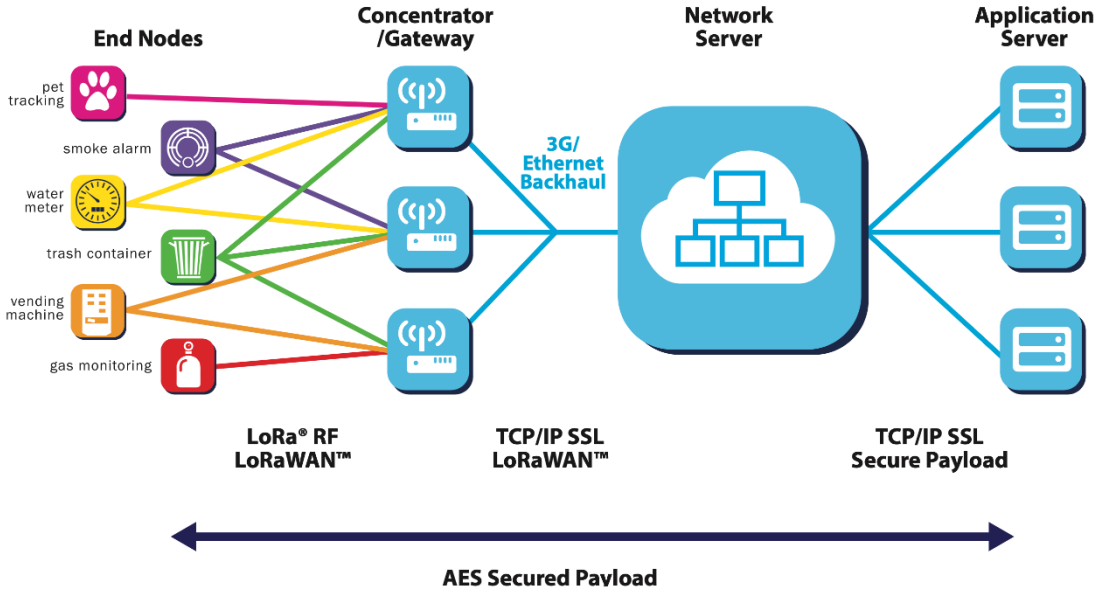


Figura 3. Arquitectura de red LoRaWAN

En la figura 3 se distinguen cada uno de los elementos que forman la red LoRaWAN. Si bien se han mencionado los dispositivos finales y el servidor centralizado de red, aparece un nuevo elemento crucial para la comunicación, y esta es la pasarela (*gateway*) que permite la comunicación entre los dispositivos, que se comunican mediante enlaces radio, y el servidor, el cual se comunica con la pasarela mediante IP.

También se identifica un nuevo elemento que es el servidor de aplicación. Esto se debe a que dentro de una misma red LoRaWAN pueden coexistir varias aplicaciones, ofreciendo así una gran flexibilidad ya que con una infraestructura se puede asignar cada dispositivo a una aplicación independiente [7].

2.2.2. SEGURIDAD

Este estándar utiliza encriptación AES-128 en modo CBC (cifrado por bloques). Se utilizan 2 claves: Una para asegurar la seguridad entre el dispositivo y el servidor, y otra para encriptar los datos.

- **Clave de sesión de red (*NwkSKey*):** Compartida por el dispositivo y el servidor de red para asegurar la privacidad y la integridad de la comunicación, creando una firma específica para cada dispositivo.
- **Clave de sesión de aplicación (*AppSKey*):** Similar a la de red, y permite encriptar/desencriptar el *payload* de datos de aplicación.

Cada mensaje es encriptado mediante una operación XOR entre el propio mensaje y un *key stream*, formado por ambas claves y un contador de mensajes (ya sea *uplink* o *downlink*).

A partir de estas dos claves, son creadas varias claves nuevas mediante distintas variantes del método de cifrado AES [8]. Cada una de estas claves generadas es empleada para una tarea de autenticación concreta, logrando así una mayor seguridad. La manera de obtener estas claves y el uso de cada una de ellas se puede observar en la figura 4.

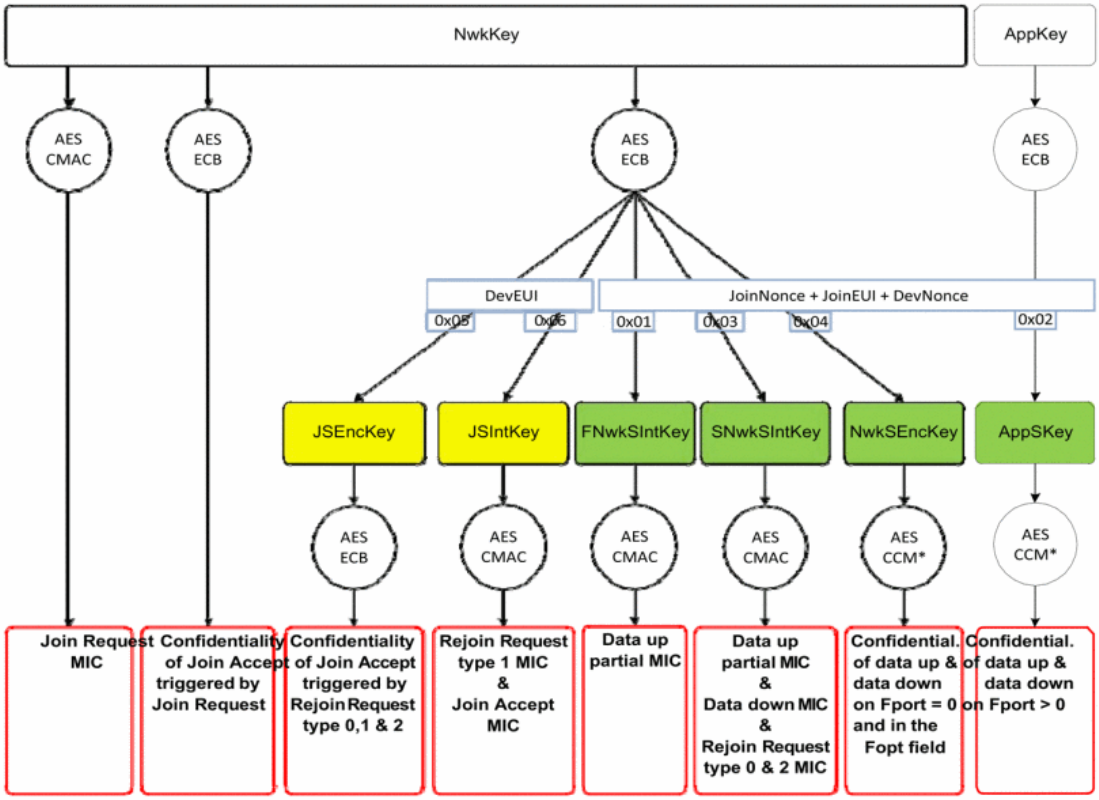


Figura 4. Esquema de claves del estándar LoRaWAN

Tras definir estas claves en el dispositivo a conectar a la red, este se une a la red siguiendo dos esquemas de identificación:

- ***Activated By Personalization (ABP):*** El dispositivo es configurado con un identificador único e inmutable. Este permite identificarse en la red, pero debido a que es almacenado de manera persistente, es vulnerable a modificaciones intencionadas o suplantación de identidad.
- ***Over-The-Air Activated (OTAA):*** El dispositivo cuenta con una serie de claves fijas que permiten la comunicación con el servidor para que este le asigne un identificador dinámico. Este proceso se realiza cada vez que el dispositivo se conecta a la red, y además de automatizar el proceso de asignación de identificadores, ofrece mayor seguridad que el método ABP.

2.2.3. CLASES DE DISPOSITIVOS

Se definen tres clases de dispositivos, atendiendo a la capacidad que estos tienen para recibir mensajes *downlink* provenientes del servidor. Todo dispositivo mantiene abierta una ventana de recepción tras cada transmisión con el fin de recibir la confirmación de este envío (ACK).

Sin embargo, puede ser necesario que estos dispositivos reciban información procedente del servidor, por ejemplo, una respuesta desde la capa de aplicación. Este tiempo de respuesta es superior al definido para la confirmación de los envíos, por lo que será necesario definir ventanas de recepción adicionales para recibir esos mensajes *downlink* dentro de un rango de latencia admisible para la aplicación a diseñar. Este sistema de ventanas permite optimizar el consumo de energía de los dispositivos, manteniendo la opción de que además de enviar, reciban datos desde el servidor.

Dado que el envío en el sentido *downlink* depende de la capacidad de los dispositivos para abrir su ventana de recepción, estos mensajes se añaden a una cola en la cual se irán enviando cuando se abra la ventana de recepción del dispositivo al cual va dirigido el mensaje.

De esta manera, se definen tres clases de dispositivos en función de la necesidad de recibir datos desde el servidor:

- **Clase A**: Comunicación bidireccional en la que el dispositivo cuenta con 2 ventanas de recepción tras la transmisión de datos.
- **Clase B**: Comunicación bidireccional en la cual el dispositivo cuenta con ventanas de recepción adicionales, sincronizadas temporalmente con el Gateway, por lo que es posible en envío de información *downlink* sin depender de las ventanas posteriores a cada envío.
- **Clase C**: Permite la recepción continua de datos, puesto que siempre están a la escucha. Esto conlleva un mayor gasto de energía.

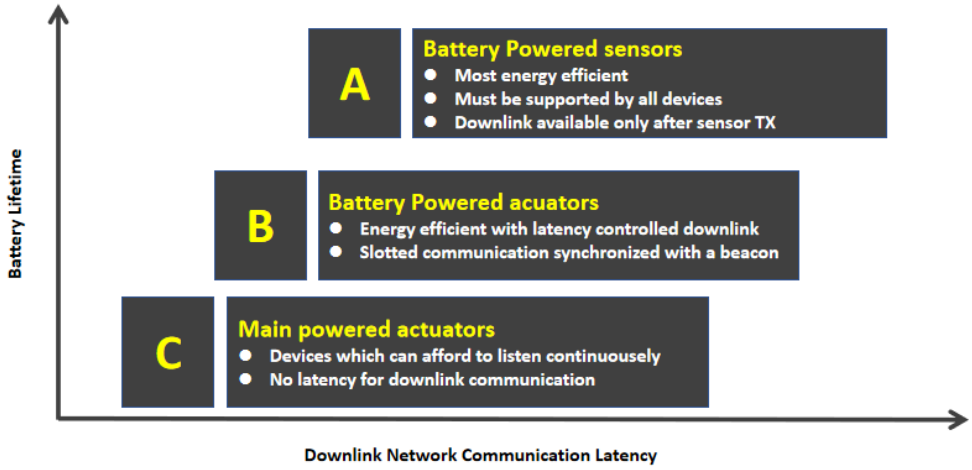


Figura 5. Tipos de dispositivos según el estándar LoRaWAN

El uso de cada una de las clases está directamente relacionado con la función que realizará el dispositivo en la aplicación final. Como se puede apreciar en la figura 5, aquellos dispositivos alimentados mediante baterías que no requieran la recepción de información (como puede ser un sensor) serán de clase A, mientras que los dispositivos con restricciones en el tiempo de retardo a la hora de recibir mensajes (como puede ser un actuador) serán de clase C.

2.2.4. FUNCIONAMIENTO

Los dispositivos han de darse de alta en el servidor de red para que puedan comunicarse. Al registrarlos, han de ser asignados a una aplicación. Esto permitirá insertar las claves necesarias en el dispositivo para las dos modalidades de autenticación. Cuando estos se inicien, se validarán frente al servidor, y este comenzará a recibir y entregar la información enviada desde estos.

Para que la comunicación sea posible ha de existir un *gateway* que reenvíe los mensajes en ambos sentidos, permitiendo así un enlace *half-duplex*. Este *gateway* es un elemento crucial de la arquitectura, ya que de él dependerá la cobertura del enlace LoRa. Cabe destacar que, para el envío de datos de aplicación, esta pasarela es invisible ya que la comunicación se realiza entre el dispositivo y el servidor de red, independientemente del *gateway* que intervenga en ella.

Esta pasarela recibe todos los mensajes enviados desde los dispositivos de su área de cobertura y los reenvía vía IP al servidor de red. Cuando los mensajes llegan al servidor de red, este los descifra mediante la clave de sesión de red, elimina los mensajes duplicados y los entrega a la aplicación en la que esté registrado el dispositivo. Esto permite, y además se recomienda, que se solapen zonas de cobertura de cada

gateway, ofreciendo así una pasarela de respaldo para que el dispositivo no se quede sin comunicaciones si uno de los *gateways* dejara de prestar servicio.

Existen multitud de implementaciones para estas pasarelas, variando en la cantidad de canales disponibles para comunicaciones simultaneas, el área de cobertura que ofrecen, y el protocolo de comunicación que usa para comunicarse con el servidor de red.

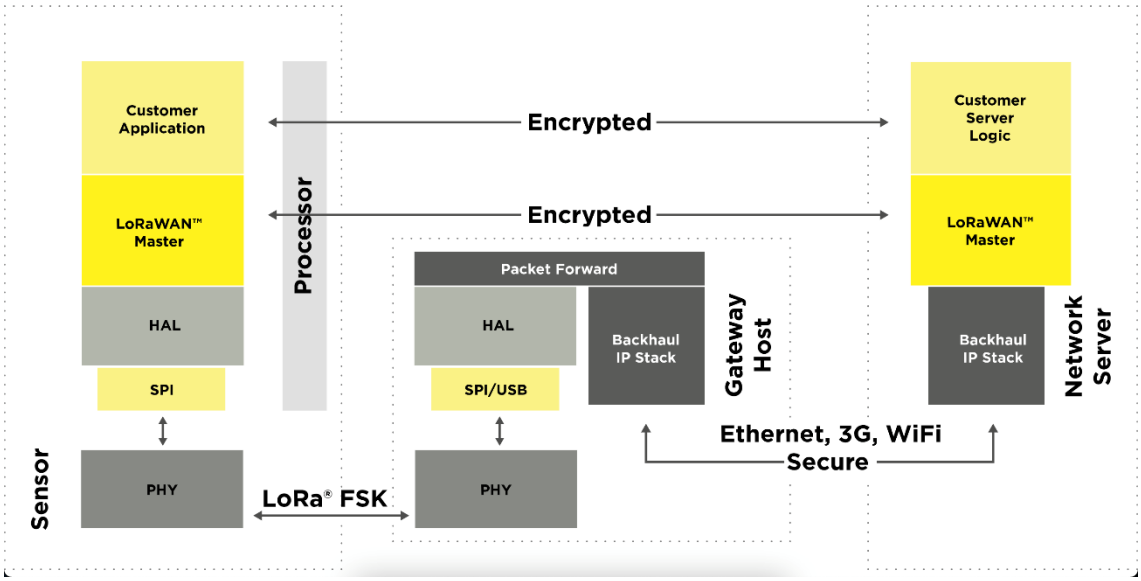


Figura 6. Esquema lógico del estándar LoRaWAN

En la figura 6 se describe un esquema lógico de la arquitectura de la red LoRaWAN dividido por capas. En este, se identifican:

- **Nivel de aplicación:** Encargado del envío y recepción de los datos útiles para la aplicación a desarrollar. Este se encripta mediante la clave de aplicación
- **Nivel de red:** Encargado de ofrecer fiabilidad en la entrega de mensajes, los cuales se encriptan mediante la clave de red.
- ***Backhaul IP Stack*:** Encargado de la comunicación entre el *gateway* y el servidor de red para el reenvío de mensajes.
- ***Packet forwarder*:** Encargado del reenvío de mensajes LoRaWAN para las comunicaciones entre el servidor y los dispositivos situados en el borde de la red en ambos sentidos.
- ***Hardware Abstraction Layer* (HAL):** Permite separar y abstraer la implementación física de la comunicación LoRa del resto de capas superiores.
- **Capa física:** Enlace radio usando la tecnología LoRa para el envío de los mensajes entre los dispositivos y el *gateway*.

2.2.5. PACKET FORWARDER Y BACKHAUL IP

Si bien todo dispositivo utiliza el protocolo LoRaWAN para comunicarse, no ocurre lo mismo en la comunicación entre el *gateway* y el servidor, que depende del *Backhaul IP*. El estándar LoRaWAN define dos implementaciones de este protocolo: Semtech UDP Packet Forwarder y LoRa Basics Station. Las principales características de ambos están recogidas en la tabla 1.

Existen otras implementaciones que no forman parte del estándar, y que solamente son compatibles con ciertos servidores de red. Estas son propietarias y su uso implica limitaciones para modificar el servidor de red. Aunque pueden ofrecer funcionalidades interesantes, se recomienda seguir el estándar para mantener la compatibilidad.

Cualquiera de las implementaciones existentes incluye un *packet forwarder*, aunque este no limita la compatibilidad debido a que funciona de manera aislada dentro del *gateway.*

2.2.5.1. SEMTECH UDP PACKET FORWARDER

Se trata de la primera implementación de este tipo de protocolo y basa su comunicación en el protocolo Semtech UDP. Esta caracterizado por su simplicidad ya que codifica los mensajes en un formato similar a JSON y los envía mediante UDP [9].

Su principal desventaja es la seguridad ya que la autenticación del *gateway* frente al servidor se realiza mediante un Identificador Único Extendido (EUI), el cual puede ser modificado en la propia pasarela. Además, esta comunicación no es encriptada por lo que no ofrece privacidad.

Otras desventajas son su fiabilidad, puesto que la comunicación se realiza mediante un intercambio UDP de dos sentidos. Finalmente, y puesto que utiliza un formato similar pero no igual al JSON, puede llegar a suponer problemas a la hora de tratar los datos, ya que permite índices duplicados (algo imposible en JSON).

Gracias a su simplicidad y su largo recorrido, esta implementación está disponible para la mayoría de hardware, aunque su uso está ya desaconsejado.

2.2.5.1. LORA BASICS STATION

Esta implementación es el nuevo estándar propuesto por Semtech [10]. Proporciona solución a las principales desventajas del estándar anterior, especialmente en el ámbito de la seguridad, para la cual utiliza un sistema de *tokens* para la autenticación y TLS para el envío de mensajes. También ofrece mejoras en la fiabilidad de entrega de mensajes, ya que basa su funcionamiento en Sockets web, los cuales funcionan sobre TCP.

Ofrece soporte para los dispositivos de clase B, ya que mediante localización GPS es capaz de sincronizar el envío de señalización de manera sincronizada entre toda la red LoRaWAN para permitir a estos dispositivos abrir la ventana de recepción de manera sincronizada con todas las pasarelas que forman la red. Ofrece también un sistema de configuración y actualización remota y automática (CUPS) para este protocolo, permitiendo así reducir el coste de mantenimiento de estos elementos de la red.

Esta implementación ha sido diseñada para permitir la máxima compatibilidad, por lo que puede ser instalada en *gateways* con entornos Linux, y soporta los nuevos diseños estandarizados para el desarrollo del *hardware* necesario. Cabe destacar que el estándar LoRa Basics Station ha comenzado a usarse en el año 2020, por lo que actualmente su compatibilidad no es tan amplia como lo será en los próximos años.

	Semtech UDP Packet Forwarder	**LoRa Basics Station**
Autenticación	Mediante EUI. Fácil de suplantar	Basado en *tokens*. Más seguro
Cifrado de mensajes	Sin cifrar	Cifrado mediante TLS
Fiabilidad	Intercambio UDP	Sockets web sobre TCP
Mantenimiento remoto	Nulo	Sistema de configuración y actualización automático (CUPS)
Compatibilidad	La mayoría de hardware	Actualmente reducida, pero se espera que sea muy superior en un futuro

Tabla 1. Comparativa entre Backhauls IP incluidos en el estándar LoRaWAN

2.3. SERVIDOR DE RED

Dentro de la arquitectura descrita en el estándar LoRaWAN, debe existir un servidor único y centralizado encargado de la gestión de toda la red. Este servidor, además, actúa como concentrador de mensajes enviados por todos los dispositivos desde el borde de la red, y es capaz de direccionar mensajes procedentes de capas superiores a cada uno de los dispositivos.

Entre las tareas que realiza este elemento de la red, destacan:

- **Autenticación y gestión de la seguridad:** Se encarga de verificar la identidad de cada dispositivo que se une a la red ya sea un nodo final de esta, o una pasarela. Además, gracias a las claves tanto de red como de aplicación, asegura la privacidad e integridad de los datos enviados a través de esta red.
- **Capa de enlace:** Lleva a cabo las tareas propias de la capa de enlace del modelo OSI como es la fiabilidad en la entrega de datos, o la eliminación de mensajes duplicados.

- **Comunicación con capas superiores:** Como ya se ha mencionado, tras desencriptar cada mensaje, este es entregado al servidor de aplicación correspondiente para que sea decodificado.
- ***Scheduling* en el sentido *downlink*:** Tras recibir un mensaje dirigido a uno de los nodos finales desde las capas superiores, el servidor de red lo añade a una cola de envíos para que sea enviado a su destinatario cuando el dispositivo abra su ventana de recepción.

Aunque de manera lógica está separado del servidor de red, el servidor de aplicación suele implementarse junto con el de red. Sus funciones no afectan en gran medida a la red, sino que actúa como interfaz con las capas superiores de la aplicación global. Este servidor de aplicación es el encargado de decodificar y validar el *payload* de datos dirigido a estas capas superiores.

A la hora de decantarse por un servidor u otro, es importante atender a su compatibilidad con el estándar LoRaWAN. Si bien los dispositivos van a ser siempre compatibles, no ocurre lo mismo con las pasarelas, puesto que existen varios protocolos para conectarlas con el servidor.

2.3.1. IMPLEMENTACIÓN PROPIA

Siguiendo el concepto tradicional de un servidor, estas funcionalidades pueden ser implementadas en un servidor propio, lo que conlleva un mayor control de toda la red LoRaWAN. Además, si la finalidad de este es conectarse con otros servicios también alojados en el mismo servidor, la eficiencia del sistema global aumentará considerablemente. En esta línea y dado que internamente estos servidores funcionan mediante un sistema de mensajes a los que distintos servicios pueden suscribirse denominado MQTT, es posible crear todo un sistema de traspaso de datos siguiendo esta filosofía.

Aunque esta opción ofrezca mayor flexibilidad y personalización, el mantenimiento de este tipo de servidores lleva consigo un trabajo crítico para no poner en riesgo ni la seguridad ni el correcto funcionamiento del sistema. Este mantenimiento incluye gestión de la seguridad, recuperación de desastres (mediante copias de seguridad, redundancia de recursos, etc), actualizar periódicamente el software, instalar nuevo y mejor hardware para compensar el crecimiento del sistema, etc.

2.3.1.1. CHIRPSTACK

Una de las implementaciones *open-source* más populares es ChirpStack, la cual ofrece además del servidor de red y aplicación, toda la pila necesaria para desplegar una red LoRaWAN. Además, permite modificar ligeramente la arquitectura de red para adaptarla a las necesidades de cada aplicación [11]. Entre estas modificaciones, llaman la atención las siguientes:

- ***Gateway OS**:* Ofrece la posibilidad de implementar tanto la pasarela, como el servidor de red y aplicación en el mismo dispositivo. Está preparado para soportar sistemas con base Linux como puede ser una Raspberry Pi, por lo que además del sistema LoRaWAN, se podrían implementar también otros servicios como una base de datos, o una aplicación web, permitiendo así crear todo un sistema IoT controlado desde un ordenador de bajo costo, haciéndolo ideal para proyectos piloto, o de pequeño tamaño.
- ***Gateway Bridge:*** Permite expandir las capacidades del *Backhaul IP* (encargado de la conexión entre el servidor y cada pasarela) para que se comuniquen siguiendo el protocolo MQTT. Esto permite integrarlo con cualquier aplicación que soporte este protocolo, ofreciendo aún más versatilidad a la hora de implementar todo un sistema basado en la red LoRaWAN.
- ***Concentratord (daemon):*** Otorga al *gateway* capacidad para comunicarse con servicios externos para facilitar la interacción con su hardware más allá de la configuración básica para la red LoRa. Esto es posible gracias a la librería ZeroMQ, que permite exponer una API, en este caso no se basa en HTTP sino en sockets, para que otros servicios se conecten y puedan interactuar con la pasarela.

ChirpStack ofrece compatibilidad con ambas implementaciones de *Backhaul IP* propuestas en el estándar LoRaWAN, además de ofrecer su propia implementación con las ventajas previamente expuestas. En la figura 7 se describe una arquitectura que ha sido alterada respecto a la propuesta en el estándar LoRaWAN, pero que es posible implementar mediante el bloque de software que ofrece ChirpStack, permitiendo así una mayor flexibilidad para el sistema y una gran variedad de integraciones con capas superiores.

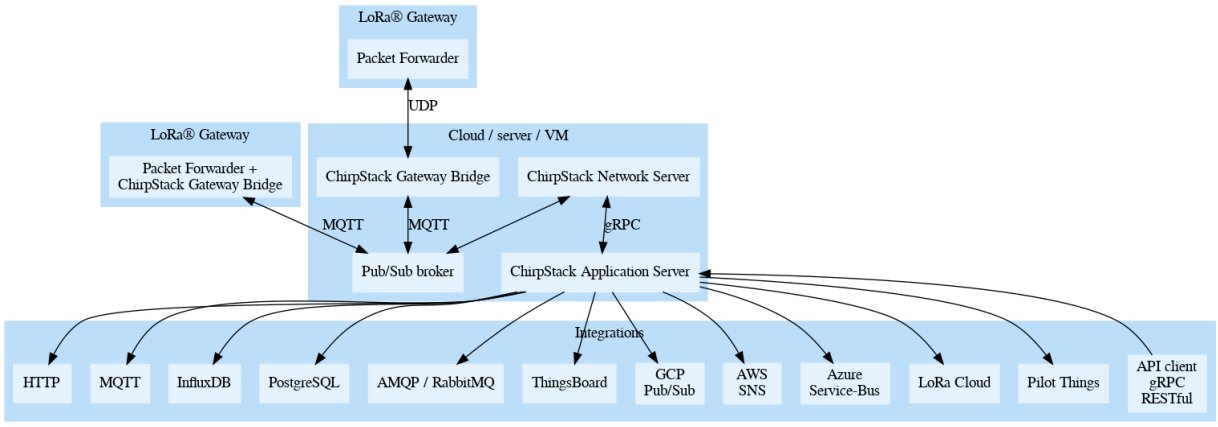


Figura 7. Arquitectura de red implementada mediante ChirpStack

2.3.2. AMAZON WEB SERVICES (AWS)

Este gigante empresarial cuenta con todo un sistema dedicado al sector IT. Ofrece una gran variedad de servicios orientados a facilitar el desarrollo de sistemas informáticos y de telecomunicaciones, y permiten integrar unos con otros de una manera sencilla e intuitiva.

Dentro del paquete AWS, se ofrecen las denominadas infraestructura como servicios (IaaS), que permiten contratar infraestructuras sobre las que desplegar aplicaciones, liberando al cliente de la carga que conlleva el mantenimiento de este tipo de servidores. Entre estos servicios ofrecidos destacan: virtualización de servidores, bases de datos, balanceo de carga, funciones Lambda (programación funcional), redes privadas virtuales (VPNs), *machine learning* y *big data,* contenedores tipo Docker, almacenamiento en bloque, etc.

Además, el uso de IaaS facilita enormemente la escalabilidad de estas, y abstraen al programador de las cuestiones más técnicas como son los cortafuegos, los *backups* o el balanceo de carga mediante la virtualización de estos. Otra de las cualidades que se destacan de estos servicios es la capacidad de pagar en función al uso de recursos de manera que se alcanza un equilibrio entre coste y recursos hardware.

Dentro del sistema de Amazon existe un bloque dedicado al sector IoT que permiten construir una arquitectura completa de dispositivos IoT, junto con su conexión con la nube de AWS [12]. En él que se enmarcan los siguientes servicios:

- ***Amazon FreeRTOS***: Se trata de un sistema operativo basado en FreeRTOS sobre el cual Amazon ha añadido las librerías necesarias para simplificar la implementación de sus servicios. Ofrece conectividad y acceso a recursos locales, conectividad segura a la nube, actualizaciones OTA, etc.
- ***IoT Greengrass***: Se implementa en los dispositivos IoT y ofrece la posibilidad de implementar lógica basada en otros servicios de AWS. Está ideado para conectar varios dispositivos dentro de un área 'local', y realizar funcionalidades entre ellos. Un ejemplo podría ser la lógica entre un sensor de humedad y un interruptor de riego.
- ***IoT Core***: Se trata del núcleo de todos estos servicios. Sirve como puerta de entrada a la nube para los dispositivos. Gestiona la conexión con los dispositivos para que sea segura. Implementa también *'Rules Engine'*, un conmutador para redirigir los datos de los dispositivos hacia otros servicios.
- ***IoT Device Manager***: Permite la Integración, monitorización y administración remota de dispositivos.
- ***IoT Device Defender***: Implementa elementos de seguridad como detección de anomalías y envío de alertas en dispositivos.

Lo mínimo necesario para establecer una conexión entre un dispositivo y la nube AWS es registrar dicho dispositivo en el panel *AWS IoT Resources*. Esto creará un *Thing Object* (representación virtual del dispositivo IoT), al que se le debe añadir un certificado X.509 para poder autenticar el dispositivo.

Además, se debe crear un programa que se ejecuta en el dispositivo y que, utilizando librerías de AWS, se encarga de establecer la conexión con la nube y el envío de los datos necesarios. Esta conexión se puede implementar con el protocolo MQTT o mediante una API REST con HTTPS 1.0 o 1.1. Para acceder a estos datos almacenados en la nube se puede hacer mediante el protocolo MQTT, MQTT over WSS o HTTPS.

2.3.2.1. AWS IOT CORE FOR LORAWAN

Recientemente Amazon ha lanzado al mercado un nuevo servicio que permite implementar un servidor de red LoRaWAN y conectarlo con *IoT Core* para integrar el resto de servicios disponibles con esta red [13]. Ofrece también un servicio denominado *AWS IoT* Rules que implementa las funciones esperadas del servidor de aplicación LoRaWAN.

El uso del bloque de servicios AWS permite integrar toda una aplicación comercial sin la necesidad de gestionar servidores, solamente configurando cada uno de los servicios y conectándolos mediante las herramientas que el propio Amazon ofrece. Esta opción facilita el despliegue de un producto desde cero y reduce enormemente los costes puesto que no es necesario que exista una inversión inicial para obtener el hardware necesario que alberque todas las funciones propias de un servidor.

En cuanto a la compatibilidad con las distintas implementaciones de *Backhaul IP*, AWS solamente ofrece soporte para el reciente protocolo propuesto por el estándar LoRaWAN: LoRa Basics Station.

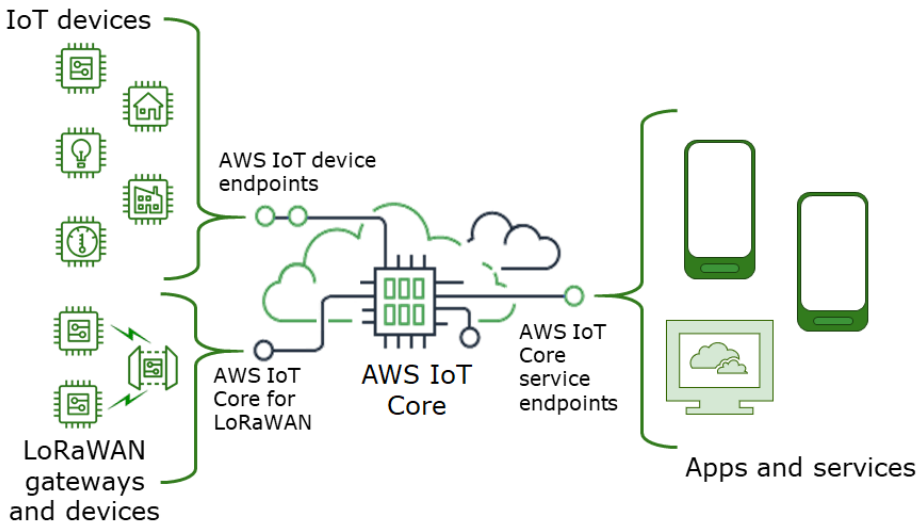


Figura 8. Esquema de componentes de un sistema implementado sobre AWS

En la figura 8 se representa un esquema básico de utilización de los servicios AWS IoT. En esta se puede observar como estos servicios actúan como intermediarios entre

la toma de datos y la aplicación final. Dentro de la nube AWS pueden existir multitud de funcionalidades que otorguen un valor añadido a los datos obtenidos desde los dispositivos IoT, aportando así mayor riqueza a la aplicación a desarrollar.

2.3.3. THE THINGS NETWORK / THE THINGS STACK

The Things Network es una Plataforma como Servicio (*PaaS)* que ofrece un servidor de red y de aplicación para implementar una red LoRaWAN [14]. Si bien cuenta con versiones de pago con sus consecuentes ventajas, ofrece también una versión gratuita sustentada por la comunidad e ideada para el desarrollo de prototipos. Proporciona una interfaz web amigable para la configuración de todos los parámetros necesarios para este despliegue y permite integrar capas superiores mediante la conexión con bases de datos externas, APIs HTTP, y otros servicios.

En contraposición con su principal ventaja, su gratuidad, carece de servicios extendidos para ofrecer mayor valor a los datos o para facilitar la implementación y despliegue de aplicaciones completas.

Sobre la compatibilidad con las pasarelas LoRaWAN mediante el *BackHaul IP*, este servidor es interoperable con los dos protocolos incluidos en el estándar, además de proponer una implementación propia y optimizada para su sistema.

	Implementación propia	**AWS IoT for LoRaWAN**	**The Things Network**
Coste	Adquisición y mantenimiento del hardware	Cuota en función del uso y características hardware	Gratuito
Mantenimiento y escalabilidad	Gestionado de forma 'manual' por la empresa responsable	Gestionado automáticamente y simplificado	Gestionado automáticamente y simplificado
Compatibilidad con Backhaul IP	Completa con el estándar LoRaWAN y extensible a otras arquitecturas de red	Limitada a LoRa Basics Station (incluida en el estándar LoRaWAN)	Completa con el estándar LoRaWAN y con su implementación propia
Funciones extendidas	Múltiples. Mediante interconexión de sistemas de manera 'manual'	Múltiples y de fácil configuración. Todo gestionado mediante la nube AWS	Mínimas
Coste de desarrollo e implementación	Alto, pues ha de ser instalado y	Sencillo. Mediante un registro y una	Sencillo. Mediante un registro y una

	configurado a bajo nivel en el servidor	serie de pasos bien documentados	serie de pasos bien documentados
Soporte	Nulo	Gestionado por Amazon	Nulo

Tabla 2. Comparativa entre implementaciones del servidor de red

2.4. CAPA DE APLICACIÓN

Existe una gran variedad de opciones para implementar una capa de aplicación que permita otorgar un valor añadido a los datos obtenidos mediante las capas inferiores. Debido a que este proyecto está orientado a ampliar las capacidades del sistema actual de la empresa Energibid, esta lógica de negocio será posiblemente implementada mediante una aplicación web desarrollada en PHP. Esta aplicación utiliza los lenguajes estandarizados para la web como son HTML, CSS y JavaScript, así como tecnologías que facilitan el desarrollo como son Bootstrap, Handlebars, Smarty; y extensiones que permiten añadir módulos de funcionamiento de una manera sencilla y rápida.

Dado que integrar este proyecto en el actual sistema es una tarea que ha de ser planificada y desarrollada por parte de la empresa, se opta por ofrecer una capa de aplicación de desarrollo propio que, aunque de manera simplificada, sea capaz de mostrar los datos obtenidos desde la red LoRaWAN de una manera amigable. Para implementar esta capa de aplicación se pueden usar tecnologías web, tratando de simular la implementación final que puede llegar a tener, aunque eso conlleva un proceso complejo y un consecuente despliegue en un servidor.

Alternativamente, esta capa puede ser desarrollada como una aplicación móvil, reduciendo los tiempos de desarrollo, así como la necesidad de que sea desplegada en un servidor. Para desarrollar una aplicación móvil existen varias opciones que se agrupan en dos bloques: aplicaciones nativas y aplicaciones multiplataforma.

La implementación mediante aplicaciones nativas ofrece un rendimiento superior al resto de opciones, además de permitir un acceso completo al hardware en el que se ejecutan. A cambio, este desarrollo se orienta a un sistema específico, ya sea Android o IOS (aunque existen otros sistemas operativos, no se plantea extender el soporte ya que su cuota de mercado es mínima). Dado que la aplicación ha de estar disponible en ambos sistemas, implementar la aplicación de forma nativa implica dos desarrollos paralelos.

Las aplicaciones multiplataforma permiten unificar su desarrollo, manteniendo la compatibilidad con múltiples sistemas. El rendimiento que ofrecen estas aplicaciones es inferior al que ofrece un desarrollo nativo, aunque existen *frameworks* que en los últimos años han mejorado enormemente, permitiendo así crear aplicaciones multiplataforma con un rendimiento similar a las nativas.

Dentro de las aplicaciones multiplataforma se distinguen tres tipos:

- **Generadas:** Permiten un desarrollo único para la lógica a implementar, pero requieren desarrollar código específico para el acceso a las APIs del dispositivo sobre el que se ejecutan.
- **Híbridas:** Ofrecen un desarrollo único íntegro gracias a las herramientas ofrecidas por el *framework* para el acceso a las interfaces del dispositivo.
- ***Progressive Web Apps* (PWA):** Consisten en aplicaciones web optimizadas para dispositivos móviles. Su desarrollo es igual que el de una aplicación web convencional.

Independientemente del tipo de implementación, es necesario tener en cuenta cuestiones como la monetización, o el mantenimiento de la aplicación mediante actualizaciones. Para las PWA no es necesaria ninguna instalación por lo que el acceso ha de ser controlado mediante la validación de usuarios. Por el contrario, el resto de alternativas requieren la instalación de software en el dispositivo por lo que la monetización puede ser gestionada mediante la plataforma oficial de distribución (ya sea la versión de Android o de IOS), o mediante la validación en el acceso. En cuanto a la distribución de actualizaciones, en las PWA basta con añadirlas al servidor encargado de hospedar la aplicación, mientras que en las otras opciones requieren distribuir el nuevo software mediante las plataformas oficiales.

En la tabla 3 se realiza una comparación entre las distintas opciones disponibles para desarrollar una aplicación móvil, atendiendo a las características principales esperadas en este tipo de aplicaciones.

	Nativas	**Generadas**	**Híbridas**	**PWA**
Desarrollo	Distinto para cada SO	Una parte del código es único y otro es específico del SO	Código único para todos los SO	Código único para todos los SO
Rendimiento	Máximo	Óptimo	Bueno	Aceptable
Monetización	Plataformas oficiales o validación de acceso	Plataformas oficiales o validación de acceso	Plataformas oficiales o validación de acceso	Mediante validación en el acceso
Distribución y actualización	Requieren instalación. Mediante plataformas oficiales	Requieren instalación. Mediante plataformas oficiales	Requieren instalación. Mediante plataformas oficiales	No requieren instalación. Siempre se usará la última versión.

Tabla 3. Comparativa entre tecnologías para el desarrollo de aplicaciones móviles

2.4.1. REACT NATIVE

Este *framework* para el desarrollo de aplicaciones móviles permite crear aplicaciones generadas con un rendimiento óptimo y un desarrollo único, siempre que no requiera de un acceso intensivo y avanzado al hardware del dispositivo. Está desarrollado por Facebook y se basa en React, un *framework* de desarrollo web muy utilizado en el sector.

Está basado en el lenguaje JSX, el cual se desarrolló inicialmente para React y permite incluir código para representar elementos HTML y CSS dentro del lenguaje base que utiliza para todos los aspectos: JavaScript. De esta manera se pueden definir funciones que devuelven nuevos tipos de datos, como puede ser un elemento a renderizar en la pantalla. Basa su filosofía en componentes reutilizables que pueden ser incluidos en cualquier parte del código y que encapsulan la lógica necesaria para ser renderizados, así como para gestionar su estado e interactuar con otros elementos.

Este *framework [15]* ofrece una gran variedad de componentes para implementar las funcionalidades básicas esperadas de una aplicación móvil como son botones, campos de entrada de datos, vistas para organizar la interfaz, modales, etc. Estos componentes se renderizan como elementos nativos con el fin de mantener la estética y funcionamiento esperado de cada sistema operativo: '*Material Design'* para Android o '*Human Interface Guidelines'* para IOS.

Ofrece también añadir código específico para cada sistema permitiendo acceder a características o configuraciones avanzadas. Además, cuenta con una amplia comunidad que constantemente publica componentes más complejos y específicos que permiten facilitar tareas comunes en el proceso de desarrollo de una manera sencilla.

React Native ofrece una herramienta que simplifica el proceso de desarrollo facilitando la configuración y el *testeo* de la misma en dispositivos reales. Esta herramienta se trata de Expo, quien también facilita la compilación del software para su posterior distribución.

CAPÍTULO 3:
PROPUESTA

3. PROPUESTA

A lo largo de este capítulo se describe los procedimientos para la elección de las tecnologías a aplicar en el prototipo, así como los requisitos necesarios con los que debe contar el hardware a utilizar. Se describe también el proceso de implementación de todo el sistema para permitir crear una aplicación pasando por todas sus capas, desde la lectura de datos mediante sensorización, pasando por un sistema de comunicaciones LoRaWAN, hasta presentar los datos obtenidos de manera amigable al usuario final.

El objetivo de este apartado es ofrecer una base fiable sobre la que construir un sistema más complejo de manera que este pueda ser empleado en el ámbito comercial para la mejora de los sistemas IoT actualmente en uso.

3.1. DISPOSITIVOS (NODOS FINALES)

Existen multitud de dispositivos que implementan el protocolo LoRaWAN. Estos pueden variar desde soluciones comerciales listas para su uso que requieren una configuración mínima, hasta placas de desarrollo basadas en microcontroladores con una interfaz de comunicaciones LoRa. Para este proyecto, y dado que se buscan soluciones versátiles y de bajo costo, se opta por el uso de estos microcontroladores ya que permiten ser programados y cuentan con multitud de interfaces de comunicación como puertos serie, I2C, SPI, GPIO, etc. Gracias a la arquitectura de la red LoRaWAN, esta puede estar compuesta por una gran variedad de dispositivos por lo que el uso de este tipo no limita una futura expansión a otro tipo de soluciones.

El esquema de conexiones empleado para las distintas propuestas en cuanto a dispositivos de sensorización se adjuntan en el Anexo de esta memoria.

3.1.1. ELECCIÓN DE HARDWARE

Dado que se opta por el empleo de microcontroladores, es necesario que estos implementen una interfaz de comunicaciones LoRa, pero además han de implementar la capa LoRaWAN, permitiendo así conectarse de manera directa al servidor de red, tal y como se describe la arquitectura de estas redes.

La interfaz radio es implementada por multitud de chips de comunicaciones, todos desarrollados por Semtech, y siendo el más común el SX1276/SX1278. Recientemente, han sido lanzados los chips de la familia SX1262, que ofrecen diversas mejoras, en especial una optimización en el consumo energético, haciéndolo ideal para aquellos dispositivos que se alimentan mediante baterías. Ambos chips de comunicaciones se conectan con el microcontrolador mediante I2C, por lo que la compatibilidad es sencilla.

Debido a que el objetivo del proyecto es ofrecer una solución versátil y extensible, pero también simplificar la instalación de sensores gracias al uso de baterías, se opta por escoger dos placas de desarrollo distintas, ambas desarrolladas por Heltec.

3.1.1.1. HELTEC WIFI LORA 32

La principal razón para la elección de esta placa es que utiliza un ESP32 como microcontrolador. Este cuenta con 2 núcleos por lo que permite implementar un sistema de hilos y tareas que simulan una ejecución paralela, muy útil para independizar la sensorización de variables de la transmisión de estas mediante la red LoRaWAN. Este sistema de tareas puede ser implementado mediante un Sistema Operativo de Tiempo Real (RTOS) como puede ser FreeRTOS [16]. Además, gracias a que el ESP32 lo permite, cuenta con multitud de interfaces de comunicación, permitiendo así extender su funcionamiento para aplicaciones futuras.

Su principal desventaja es que no cuenta con ningún chip que implemente el protocolo LoRaWAN, por lo que este debe ser implementado mediante software. Si bien el microcontrolador es lo suficientemente potente para esta tarea, siempre es más fiable y rápido que esta lógica sea implementada por hardware. Para permitir la comunicación con el resto de la red, existen librerías oficiales que implementan la capa LoRaWAN y la presentan mediante funciones sencillas para su uso.

3.1.1.2. HELTEC CUBECELL DEV-BOARD (HTCC-AB01)

La elección de esta placa está basada en su chip de comunicaciones LoRa, ya que se trata del SX1262, un chip mejorado y enfocado a optimizar el consumo energético. Si bien su microcontrolador es menos potente y no ofrece tanta versatilidad, ni tanta potencia, ni siquiera dos núcleos, tiene un consumo mucho menor que el ESP32. Además, no necesita tanta potencia computacional puesto que cuenta con un chip que implementa mediante hardware la capa LoRaWAN, liberando así al microcontrolador de esta tarea.

3.1.1.3. SENSORES

Para el prototipo a desarrollar, se escogen dos sensores encargados de recoger datos ambientales. Ambos cuentan con librerías que facilitan su implementación en el *framework* de Arduino por lo que no es necesario implementar estas comunicaciones a bajo nivel.

Para la lectura de la temperatura y la humedad, así como el índice de calor, se opta por el sensor DHT22, el cual cuenta con un rango de operatividad entre -40 y 80 ºC. Ofrece una precisión de +- 0,5ºC en las lecturas de temperatura y de un +- 2 %RH en las de humedad, variable en la cual mide todo el rango posible: de 0 a 100% [17]. Cabe

remarcar que no permite lecturas consecutivas en un periodo inferior a 2 segundos, pero esto no supone ninguna limitación puesto que la periodicidad a utilizar para tomar las mediciones supera ampliamente esta barrera.

Para la comunicación con la placa de desarrollo, este sensor utiliza una comunicación bidireccional mediante un solo bus, lo que permite ahorrar pines en la placa si en un futuro esta estuviese conectada a un número elevado de sensores.

Para la lectura de datos acerca de la calidad del aire, se escoge el sensor SGP30, capaz de medir la concentración de etanol e hidrógeno en el aire en un rango entre 0 y 1000 ppm (partes por millón) [18]. Gracias a estas mediciones y mediante algoritmos internos en el sensor, es capaz de calcular otras variables como son el total de compuestos orgánicos volátiles (TVOC) y el equivalente de dióxido de carbono (eCO2) [19]. Estas variables permiten detectar la calidad del aire para, por ejemplo, indicar cuando ha de ser ventilada una estancia. Este tipo de casos de uso ha cobrado gran relevancia con la reciente crisis sanitaria por lo que puede llegar a ser un aplicativo de gran valor.

Para comunicarse con la placa, el SGP30 cuenta con una interfaz I2C. Esta se encuentra disponible en la mayoría de placas de este tipo por lo que puede ser utilizada con otro hardware si fuese necesario.

Este sensor, y debido a su naturaleza basada en medir concentraciones de gases, aumenta su precisión si es calibrado mediante mediciones externas de la concentración de humedad. Es por esto que se opta por implementar ambos sensores en el mismo chip de manera que uno complemente al otro.

3.1.2. IMPLEMENTACIÓN CON ARDUINO

Con el fin de desarrollar el código necesario para estas placas de desarrollo se opta por el ecosistema de Arduino. Esto se debe a que cuenta con multitud de librerías tanto para las comunicaciones LoRa, como para implementar sensores mediante otro tipo de protocolos de comunicación, facilitando el desarrollo. Este ecosistema, además, cuenta con una gran comunidad por lo que existe una amplia documentación acerca de todos los aspectos necesarios para el desarrollo de los programas necesarios.

Para escribir código compatible con Arduino se utiliza el lenguaje C++, una versión mejorada y extendida del lenguaje C. Entre estas características extendidas del lenguaje, destaca el soporte para una Programación Orientada a Objetos (OOP), permitiendo desarrollar módulos no cohesionados que son capaces de mantener su estado (atributos) de manera interna y que se comunican con otros módulos mediante interfaces (métodos).

3.1.2.1. INTERFAZ LORA

Con el fin de simplificar este desarrollo se usan las librerías ofrecidas por el fabricante para la interfaz de comunicaciones basada en LoRa. En el caso del microcontrolador ESP32, existe la librería LMIC [20] que permite implementar el protocolo LoRaWAN en un chip SX1276/SX1278 como el instalado en este dispositivo. Esta librería se encarga de mantener el estado del dispositivo en cuanto a las fases propuestas en el estándar LoRaWAN y de proporcionar funciones para la conexión con el servidor de red, el registro y autenticación del dispositivo, el envío de mensajes o la gestión de la ventana de recepción.

Para el caso de la placa de desarrollo Cubecell, se ofrece por parte de Heltec un entorno de desarrollo para poder programar este chip [21]. En este se encuentra una librería que facilita la comunicación con el chip LoRaWAN, siendo este el encargado de las tareas de bajo nivel relacionadas con este protocolo.

Ambas librerías proporcionan un esqueleto sobre el que construir el programa encargado de la lectura de los sensores instalados, y el envío de estos datos hacia el servidor de red LoRaWAN.

3.1.2.1. SISTEMA DE ASENTIMIENTOS

El estándar LoRaWAN ofrece la posibilidad de solicitar un asentimiento al receptor cada vez que este reciba un nuevo mensaje. Esto es posible gracias al bit *Confirmed Data*, y este puede ser usado en ambas direcciones de la comunicación. Cuando un extremo de la red recibe una trama con ese bit activado, responderá al emisor indicando que su trama ha sido recibida mediante la activación del bit ACK.

En el caso de envíos en el sentido *uplink*, el nodo final mantiene abierta una ventana de recepción tras cada envío con el fin de recibir el asentimiento por parte del servidor si este fuera solicitado. Este sistema de asentimientos va incluido en la cabecera LoRaWAN, por lo que el *payload* de datos no se ve afectado.

Dado que el sistema a desarrollar busca la monitorización de los datos de una manera fiable, es imprescindible aplicar este sistema de asentimientos con el fin de gestionar los posibles errores en la comunicación que provoquen la perdida de mensajes. Si bien existen implementaciones complejas que optimizan los recursos en base a la información obtenida mediante asentimientos, en el caso del estándar LoRaWAN no se definen unas reglas para gestionar el envío de mensajes perdidos o corruptos, por lo que es necesario desarrollar un sistema propio basado en la recepción de estos asentimientos enviados desde el servidor de la red.

En el sistema desarrollado, cada mensaje a enviar lleva un contador que va creciendo con cada nuevo mensaje creado. Esto permite identificar cada mensaje a enviar y llevar un registro de aquellos que han sido confirmados. Para gestionar los

posibles reenvíos que el nodo final ha de hacer en caso de que algún mensaje no llegue al servidor (o el asentimiento por parte de este no llegue al nodo), se crea un buffer de transmisión en el que se almacenan los mensajes creados con la información de las lecturas de los sensores. Se crean dos 'hilos' ficticios, puesto que el programa corre sobre un solo hilo de ejecución, en los que se diferencian las tareas de generación y posterior guardado de estos mensajes en el buffer, y comprobación y reenvío de mensajes que no han sido asentidos.

Esta diferenciación se describe en la figura 9, en la cual se definen dos temporizadores, encargados de lanzar cada hilo con una frecuencia determinada. Tras la ejecución del hilo de generación de mensajes, siempre se produce un envío, ya sea de los nuevos datos generados, o de tramas anteriores que aún no han sido asentidas. Por otro lado, tras la ejecución del hilo de reenvíos, solo se produce un envío si existen tramas en el buffer de transmisión que aún no han sido asentidas. Cabe destacar que hasta que no vence el temporizador de reenvíos (TX_INTERVAL) no se comprueba si ha vencido el temporizador de generación de mensajes (APP_INTERVAL) por lo que el primero deberá ser inferior al segundo para que el funcionamiento sea correcto.

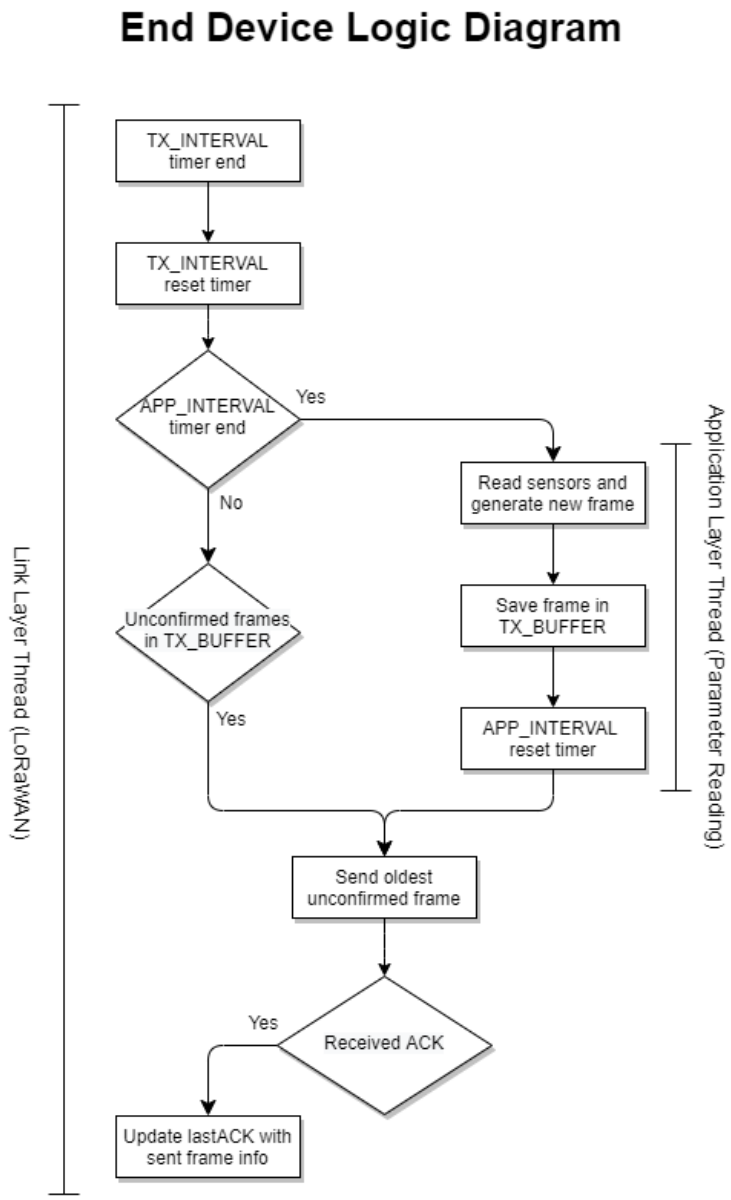


Figura 9. Diagrama de la lógica de los nodos finales

Con el fin de garantizar el orden de entrega se decide que, si existiera alguna pérdida en los mensajes enviados, se repetirá el envío de la primera trama perdida hasta que esta sea asentida. De esta manera pueden llegar a acumularse un número elevado de mensajes en el buffer a la espera de que llegue un asentimiento que permita continuar con los envíos siguientes. Esto implica que este buffer debe dimensionarse de manera adecuada puesto que, si se cortan las comunicaciones durante un periodo lo

suficientemente grande, este puede llenarse. Si esto ocurriese el sistema comenzará a pisar los mensajes más antiguos almacenados tal y como se describe en la figura 10.

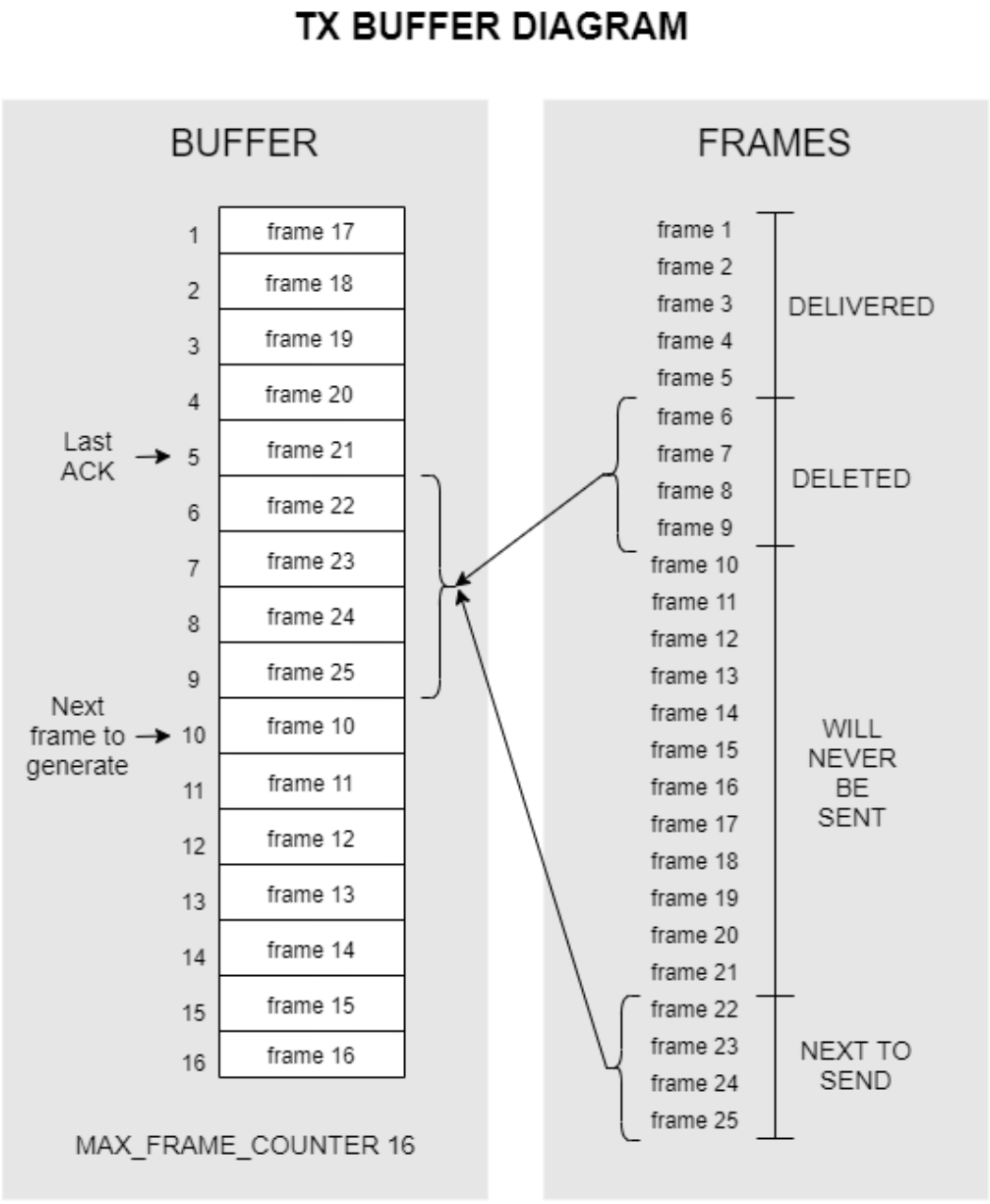


Figura 10. Ejemplo del problema del buffer lleno

Para llevar un registro de aquellos mensajes que han sido entregados, podría parecer suficiente con almacenar el contador de mensajes del último mensaje asentido: *lastACK*. Si bien esto permite diferenciar qué mensajes dentro del buffer han sido asentidos y cuáles no, si el buffer llegara a llenarse debido a un largo periodo sin comunicaciones, las tramas comenzarían a sobrescribirse y ya no bastaría con almacenar

este contador puesto que el programa perdería la noción de qué tramas son antiguas y que tramas han sobrescrito la más antigua sin confirmar, es decir, aquella que ha de enviarse. Esta situación, descrita en la figura 10, puede solucionarse almacenando además una marca de tiempo de todas las tramas del buffer, así como la marca de tiempo de la última trama confirmada. Esto permitirá, independientemente del contador almacenado como *lastACK*, enviar siempre la trama adecuada: la más antigua del buffer que esté sin confirmar.

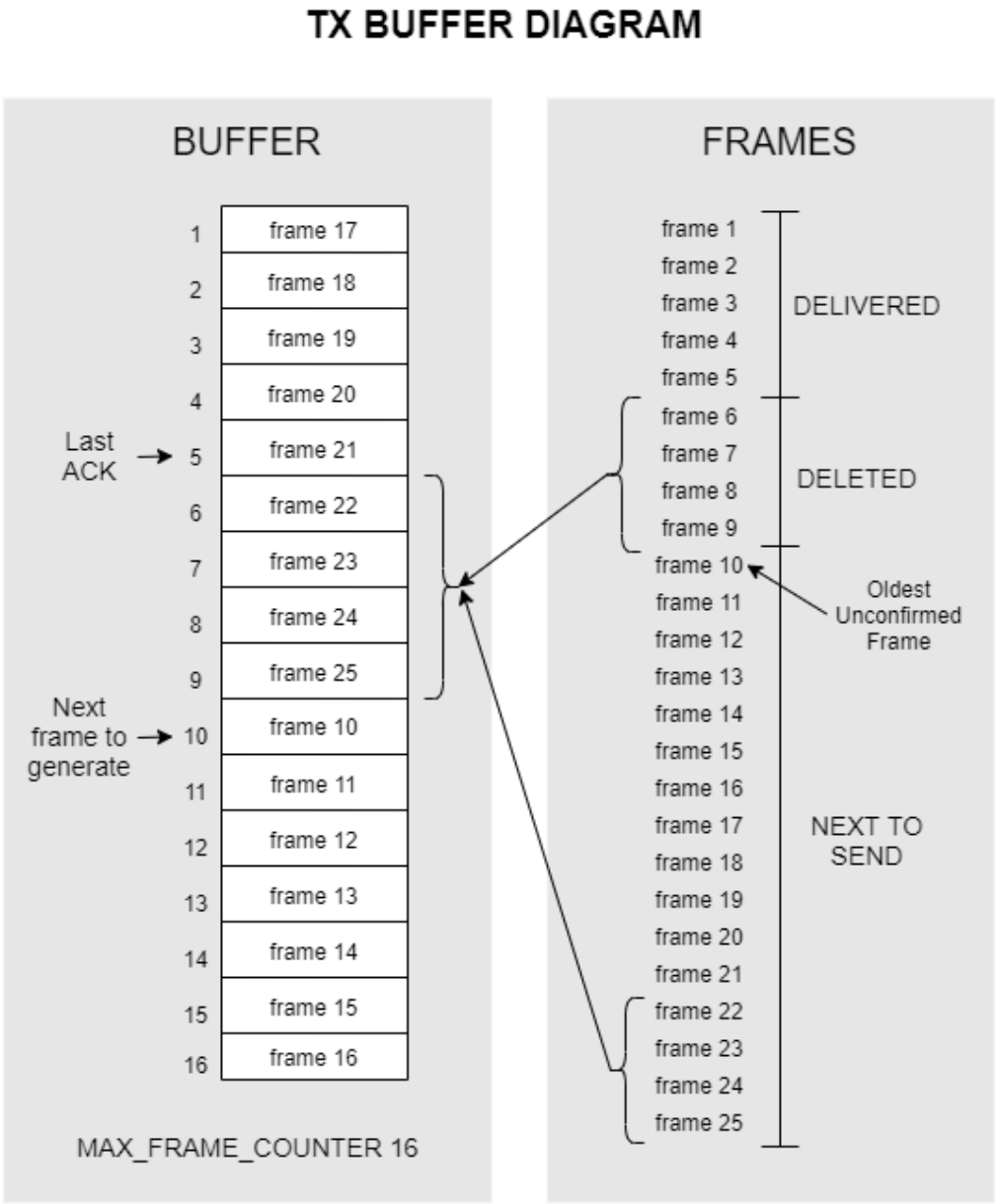


Figura 11. Funcionamiento del buffer de transmisión de los nodos finales

En la figura 11 se describe un ejemplo de funcionamiento en el que se propone el peor escenario, tal y como se ha explicado anteriormente. En este caso se cuenta con un buffer pequeño capaz de almacenar 16 tramas. La última trama entregada y por lo tanto asentida es la quinta trama generada. Tras esta, no se vuelve a entregar ninguna trama, a pesar de que el sistema continúa enviando la sexta. Cuando se genera la vigésimo segunda trama (cuyo contador será seis), esta se almacena en el espacio de memoria correspondiente a la sexta, por lo que, tras este instante, el sistema la da por perdida y envía la séptima trama puesto que es la más antigua existente en el buffer que no ha sido confirmada aún. En la figura 10 se describe el escenario en el que se han generado ya veinticinco tramas, y el sistema trata de entregar la décima generada, habiendo eliminado las cuatro anteriores que nuca serán entregadas.

Tras el desarrollo de este sistema de asentimientos es necesario evaluar su comportamiento bajo diversas circunstancias. Si bien el buffer es capaz de absorber los efectos que provoca una pérdida de comunicaciones prolongada en el tiempo, los temporizadores tanto de la capa de transmisión como de aquella encargada de generar las tramas han de estar bien configurados. En la figura 12 se presenta una gráfica en la que en función del ratio entre el temporizador de generación de mensajes (APP_INTERVAL) y el de retransmisión (TX_INTERVAL) se define la probabilidad máxima admitida de perder un mensaje. Como puede observarse, al ir aumentando este ratio el sistema es capaz de funcionar correctamente con probabilidades de perdida mayores, aunque esto provocará un mayor consumo energético ya que para la misma frecuencia de generación de tramas podrá producirse un número mayor de envíos. Esto puede llegar a ser una ventaja si se necesita que el sistema se recupere lo antes posible.

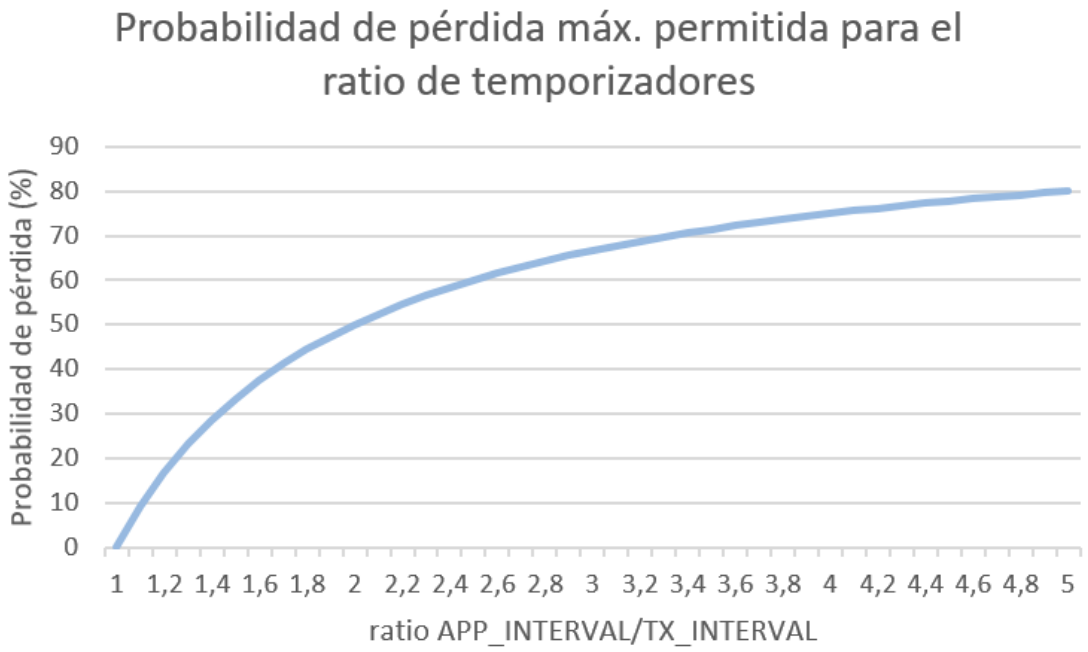


Figura 12. Probabilidad de pérdida de trama permitida en función del ratio entre temporizadores del sistema

Este razonamiento ofrece un funcionamiento estable si la probabilidad de perdida es constante, pero si la comunicación se corta por completo durante un largo periodo de tiempo, será el buffer de transmisión el que aloje todas las tramas pendientes de enviar. Es por esto que el buffer, además de absorber los efectos en la variación de esta probabilidad, deberá de ser lo suficientemente grande como para soportar un escenario sin comunicaciones en el que se enviarán todas las tramas almacenadas una vez se restablezca la comunicación.

3.1.2.1. ESQUEMA DEL PROGRAMA

Debido a que el sistema a desarrollar debe ser genérico y ofrecer soporte a multitud de variables, se opta por definir una cabecera en cada una de las tramas LoRaWAN que permita identificar el tipo de mensaje recibido por parte del servidor. Gracias a esto, la lógica encapsulada en el servidor de aplicación es capaz de decodificar la ristra de bytes recibida y entregar a las capas superiores del sistema cada variable de una forma intuitiva y simplificada, abstrayendo así la capa de enlace implementada mediante la red LoRaWAN.

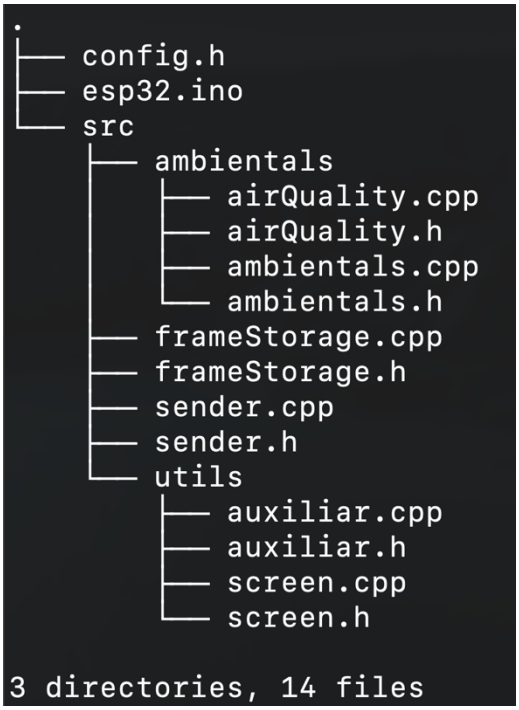


Figura 13. Esquema de clases del nodo final de la red

Con el fin de unificar ambos desarrollos, se implementa un esquema por clases definido en la figura 13 en el que se diferencian:

- **Fichero de extensión *ino*:** Requerido por el framework de Arduino donde se definen las funciones principales *setup()* y *loop()*. Es en este fichero donde

se incluye la lógica asociada a la interfaz LoRaWAN, por lo que varía en función de la placa a utilizar.

- **Clase *Sender***: Encargada de formar cada una de las tramas a enviar a partir de las lecturas realizadas a los distintos sensores con los que cuenta el dispositivo, además de gestionar los temporizadores y realizar las comprobaciones de asentimientos.
- **Clase *FrameStorage*:** Gestiona el buffer de trasmisión en el que se almacenan las tramas generadas. Además, contiene la lógica y los datos necesarios para determinar qué tramas han sido ya asentidas y cuál es la siguiente trama a enviar.
- **Clase *AmbientalSensors*:** Implementa la comunicación con el sensor de temperatura y humedad DHT22, ofreciendo una interfaz sencilla para la lectura de este por parte de otras clases. También almacena y gestiona el estado de este en caso de que hubiera un error estableciendo las comunicaciones.
- **Clase *AirQuality*:** De igual manera que con el otro sensor, esta clase implementa y encapsula la lógica relacionada con el sensor SGP30 y además de gestionar su estado, ofrece métodos para su calibración y lectura.
- **Clase *Auxiliar*:** Implementa funciones auxiliares que facilitan el manejo de datos y la depuración del programa.
- **Clase *Screen*:** Implementa y encapsula funciones que permiten imprimir información en la pantalla OLED integrada en el chip si existiera. En este caso se usa para la placa de desarrollo Heltec Wifi LoRa 32.

Con el fin de obtener un producto base sobre el que construir más funcionalidades en el futuro, se opta por dividir cada bloque de funcionamiento en clases independientes encargadas de gestionar el estado y, en el caso de la recolección de datos, las comunicaciones con los elementos conectados como son, en este caso, los sensores. Gracias a esto, un cambio en el hardware de lectura de temperatura tan solo supondría modificar la clase *AmbientalSensors,* facilitando así el futuro desarrollo y eliminando posibles errores futuros.

En cuanto a la lógica de envíos, se ha optado por separar en dos clases cada bloque de funcionamiento. Esto permite mantener encapsulada toda la lógica del buffer de transmisión y que mediante las interfaces que ofrece sea capaz de comunicarse con la clase principal de envíos, que es la encargada de la generación de tramas y de la gestión de temporizadores y de estados.

Además de las clases antes mencionadas, se incluye un fichero *config.h* en el que se declaran los parámetros de configuración necesarios. En estos se incluyen desde los pines a los que se conecta cada elemento hasta los temporizadores de envíos, pasando por el tamaño de cada trama, el tamaño del buffer, el identificador de trama, etc.

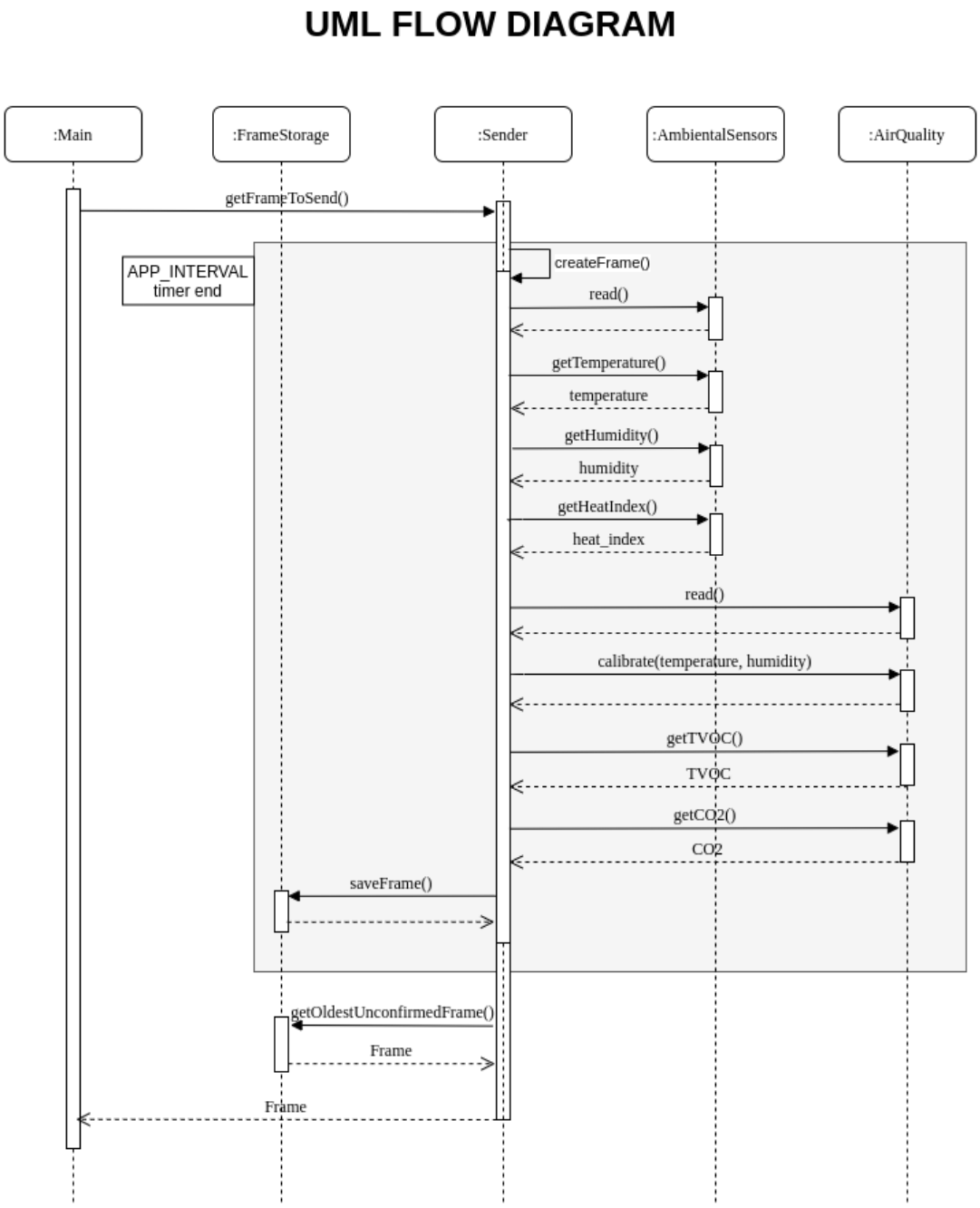


Figura 14. Diagrama de flujo UML de la generación de una trama

En la figura 14 se representa el diagrama de flujo seguido tras el vencimiento del temporizador de retransmisión (TX_INTERVAL). Si además hubiera vencido el temporizador de generación de mensaje (APP_INTERVAL), se realizarían las lecturas y la trama generada sería almacenada para su posterior envío. Además, la clase *Sender* comprueba si existen tramas pendientes en el buffer (ya sea porque han de ser reenviadas, o porque ha sido añadida una nueva trama), y pide a la clase *FrameStorage* la trama a enviar. Como se ha mencionado anteriormente, esta clase contiene la lógica necesaria para la gestión de este buffer, así como la comunicación con la clase *Sender*.

3.2. GATEWAY

El gateway es uno de los elementos que más afecta al funcionamiento de la red. De este depende la zona de cobertura, así como la capacidad de realizar diversas comunicaciones simultaneas. Esto es debido a que cualquier mensaje, ya sea *uplink* o *downlink* debe ser recibido y reenviado por este elemento de la red.

Es por esto que la pasarela ha de contar con una interfaz radio que permita soportar la carga que conlleva la recepción y envío de los mensajes LoRa que envíen todos los dispositivos de su zona de cobertura. Esta interfaz radio puede variar en cuanto al número de canales simultáneos, los factores de dispersión soportados, la potencia de transmisión, etc. La elección del chip LoRa a utilizar deberá ser acorde a la carga de trabajo que deba soportar, así como de la zona de cobertura a cubrir; logrando así un equilibrio entre coste y prestaciones.

La función principal de este elemento de la red es el reenvío de mensajes LoRa por lo que otro aspecto a tener en cuenta a la hora de escoger el tipo de pasarela es la compatibilidad que tenga en el otro extremo de la comunicación: la conexión con el servidor de red mediante el *Backhaul IP*. Como ya se ha mencionado, existen multitud de implementaciones de esta interfaz de comunicaciones, y la compatibilidad juega un papel muy importante dado que limita la elección del servidor de red a utilizar.

Para evitar que el sistema dependa de un único proveedor de servidor de red, se opta por utilizar una implementación estandarizada. En la tabla 1 se presenta una comparación entre las dos alternativas propuestas en el estándar LoRaWAN. Es claramente observable que la versión antigua presenta carencias en el ámbito de la seguridad ya que sus comunicaciones no van cifradas y es fácilmente suplantable. Estos problemas son inadmisibles para ofrecer una solución comercial por lo que, sumado a que el nuevo estándar ofrece funcionalidades añadidas que facilitan su mantenimiento, se opta por emplear el protocolo LoRa Basics Station para implementar el *Backhaul IP*.

3.2.1. ELECCIÓN DE HARDWARE

Dado que este desarrollo busca sentar unas bases para su posterior explotación, se busca la máxima versatilidad y sencillez tanto de uso como de instalación para estos sistemas. Es por esto que, dado que deben contar con una conexión a Internet, es importante qué opciones ofrece el hardware para realizar esta conexión. Si bien la mayoría de soluciones permiten conexión mediante cable Ethernet y WiFi, se opta por buscar un producto que además permita la comunicación mediante redes móviles como puede ser el 3G. Existen multitud de adaptadores que permiten este tipo de compatibilidades mediante un módem USB, pero a la larga estos dan problemas, especialmente en condiciones de calor o humedad, por lo que para evitarlos se decide que esta interfaz con redes WWAN ha de venir integrada en el hardware.

Finalmente, y teniendo lo anteriormente expuesto en cuenta, se decide el uso de una solución comercial con todas estas características incluidas, buscando la máxima fiabilidad. Se opta por la pasarela *Wirnet iFemtoCell-evolution* dado que cumple con los requisitos definidos sin elevar excesivamente su coste. Cuenta con hasta 19 canales de recepción y 1 de transmisión en la interfaz LoRa por lo que es capaz de dar servicio a múltiples dispositivos, y ofrece la posibilidad de conectarse a internet mediante un módem 4G integrado [22].

En cuanto a su software, cuenta con el sistema operativo KerOS basado en Linux, por lo que soporta multitud de software para extender sus capacidades. Dado que está orientado como dispositivo '*plug and play',* trae ya instalado el software propietario que facilita enormemente su configuración.

3.2.2. CONFIGURACIÓN

Este dispositivo trae instalado un software que permite definir una serie de parámetros en fichero de configuración con sintaxis TOML, basada en pares clave-valor y similar al formato JSON o YAML. Además, proporciona una aplicación para línea de comandos (CLI) que permite modificar esta configuración mediante parámetros, simplificando aún más esta tarea.

En busca de la máxima simplicidad ofrece varios métodos para su configuración, así como para conectarse al dispositivo. A continuación, se describen los pasos seguidos para configurarlo a través de una terminal, puesto que es el método que mayor simplicidad ofrece si esta tarea debiera ser ejecutada de manera remota.

Para comenzar a configurar esta pasarela, el primer paso es conectarse a ella mediante SSH. Por defecto trae asociado un nombre de host, por lo que basta con estar conectados a la misma red que el dispositivo y usar este *hostname* para acceder. Mientras que el usuario es fijo (*root)*, por razones de seguridad la clave está basada en el número de serie del dispositivo. Una vez accedemos, es necesario configurar tanto la interfaz LoRa (denominada LoRa Daemon) como la interfaz IP (denominada LoRa Forwarder).

3.2.2.1. LORA DAEMON

Para configurar la interfaz LoRa, accedemos al fichero `/etc/default/lorad`. En este, se debe indicar la ruta al fichero de configuración en la variable CONFIGURATION_FILE, así como habilitar esta interfaz asignando el valor "no" a la variable DISABLE_LORAD [23].

Este fichero de configuración del que se ha indicado ya la ruta, se definen los parámetros básicos para la interfaz radio: desde la región para seleccionar la banda de frecuencias, hasta la configuración de potencias de cada uno de los factores de

dispersión disponibles en el *gateway*. Existen ficheros de ejemplo para cada una de las regiones soportadas en el directorio `/etc/lorad/fevo.`

3.2.2.2. LORA FORWARDER

Tras esto, es necesario configurar la interfaz asociada al *Backhaul IP*. Para ello se sigue un esquema similar al anterior caso: En el fichero `/etc/default/lorafwd` se define la variable DISABLE_LORAFWD a "no" y se indica la ruta al fichero de configuración en la variable CONFIGURATION_FILE [23].

En el fichero de configuración indicado tan solo es necesario definir tres variables: la dirección del servidor de red, el identificador del servicio al que debe enviar cada trama en el sentido *uplink* y el mismo identificador para el sentido *downlink*. Dentro de este fichero existen multitud de opciones de configuración para funcionalidades añadidas como puede ser el almacenamiento temporal de mensajes a la espera de que puedan ser enviados, la configuración de una API HTTP para su configuración o el empleo de filtros para descartar mensajes LoRaWAN siguiendo distintos criterios.

Existe una alternativa para realizar esta última configuración y esta es el uso del comando *klk_apps_config* [24]. Este permite modificar los parámetros antes mencionados en forma de argumentos de este comando. Una configuración básica como la definida anteriormente se puede realizar mediante:

```
klk_apps_config --activate-cpf --lns-server [lorawan_server]
--lns-dport [puerto_downlink] --lns-uport [puerto_uplink]
```

Este comando ya realiza la activación del servicio *Backhaul IP,* así como de la interfaz lora; pero se recomienda reiniciar el dispositivo tras realizar esta configuración.

Finalmente, es necesario conocer el EUI de esta pasarela para poder registrarla en el servidor de red LoRaWAN. Este identificador puede ser también modificado en la configuración de lorafwd. El definido por defecto se encuentra definido en el fichero:

```
/var/run/lora/gateway-id.toml
```

3.3. SERVIDOR DE RED

El elemento más importante de la red LoRaWAN es el servidor de red. Esto se debe a que gestiona todos los elementos pertenecientes a esta de manera centralizada. Debido a esto, el servidor es un punto crítico del que depende el correcto funcionamiento de la red, por lo que ha de estar bien configurado, mantenido y seguro.

Cualquier implementación de un servidor de red LoRaWAN ha de cumplir con lo recogido en el estándar para asegurar el correcto funcionamiento de la red, pero muchas de las opciones para esta elección permiten integrar otros servicios con el fin de mejorar las características y funcionalidades que nuestra aplicación ofrece. Estas mejoras pueden suponer un valor añadido de cara al usuario, una mejora en el proceso de mantenimiento y configuración de la red, o permitir un desarrollo más ágil y sencillo de la misma.

3.3.1. ELECCIÓN DE SERVIDOR

Para llevar a cabo la elección del servidor de red a utilizar, ha de cumplirse un requisito indispensable: que sea compatible con el *Backhaul IP* que permite la comunicación con las pasarelas. Teniendo esto en cuenta, y debido a que nuestra elección para este protocolo ha sido *LoRa Basics Station*, una gran variedad de opciones ha sido descartadas ya que tan solo ofrecían compatibilidad con la antigua implementación del estándar o con protocolos fuera de este.

Otra cuestión a tener en cuenta es el tipo de implementación para este servidor. Esta puede basarse en la instalación de un software sobre un servidor ya existente y gestionado por el propietario de esta red LoRaWAN, o puede tratarse de una Plataforma como Servicio (*PaaS)* en la que, a cambio de una cuota, se ofrece toda la infraestructura necesaria para este despliegue, así como el software necesario para su puesta en marcha de una manera sencilla y amigable.

En la tabla 2 se realiza una comparación de las características principales para la elección entre tres candidatos para la implementación del servidor de red. Aunque la opción de realizar una implementación propia mediante el software *open-source* ChirpStack ofrece la capacidad de extender sus funcionalidades y de alterar la estructura de la red según convenga, queda descartado ya que supone realizar un proceso de instalación y configuración más complejo que sus alternativas. También, y dado que se busca la máxima fiabilidad para la globalidad del sistema, se opta por una solución que no requiera una constante atención si surgieran actualizaciones necesarias del software, o errores que deban ser corregidos. Finalmente, otra de las razones para descartarlo es el desembolso inicial que conlleva adquirir un hardware sobre el que instalar este servidor de red, así como los costes que conllevaría su ampliación si la red escalase en tamaño.

Debido a estas razones, se opta por elegir una alternativa que proporcione toda la infraestructura necesaria para realizar las tareas esperadas por un servidor de red. El uso de los servicios ofrecidos por *Amazon Web Services* (AWS) ofrece una implementación sencilla del servidor LoRaWAN, así como multitud de funcionalidades añadidas que permiten el análisis, almacenamiento y gestión de los datos. Estas funcionalidades añadidas se encuentran atomizadas en los diferentes servicios ofrecidos por la plataforma, y pueden ser contratados de manera individual mediante una cuota, permitiendo ajustar el presupuesto a las necesidades de cada momento. Además,

debido a la sencillez para integrar estos servicios ofrecen la capacidad de optimizar los tiempos de desarrollo y despliegue de nuevas funcionalidades.

AWS supone una solución completa y ajustada a las necesidades de este sistema, pero su servicio para ofrecer un servidor LoRaWAN se encuentra aún en fase de pruebas por lo que para este desarrollo se opta por una alternativa estable y madura. Cabe remarcar que, aunque el servicio de Amazon quede temporalmente descartado puede ser una solución interesante en el largo plazo, por lo que se el sistema completo ha sido planteado para que sea compatible con esta.

La alternativa a AWS es *The Things Network*, una implementación de servidor de red sustentada por la comunidad que, aunque no ofrece funcionalidades añadidas más allá de lo esperado de un servidor de red, permite facilitar las tareas de configuración y gestión de equipos. Esta plataforma supone una solución ideal para el desarrollo y pruebas de prototipos gracias a su amplia documentación, su compatibilidad con múltiples protocolos para el *Backhaul IP* y su coste, ya que se puede usar de manera gratuita.

3.3.2. CONFIGURACIÓN DEL SISTEMA

A continuación, se describen los pasos necesarios para realizar la configuración del servidor de red LoRaWAN en la plataforma *The Things Network*. Para comenzar es necesario entrar con una cuenta ya registrada y acceder a la consola desde la cual se gestiona todo lo referido a la red. Tras esto es necesario escoger un *cluster* sobre el que montar la aplicación. Este *cluster* depende de la región en la que se instalarán los distintos elementos de la red por lo que se opta por el europeo.

Una vez en la consola es necesario dar de alta una nueva aplicación, que implementará la lógica del servidor de aplicación definido en el estándar LoRaWAN. Asociado a la aplicación, se escoge un servidor *handler* encargado de la integración de la red con las capas superiores. La elección de este *handler* no limita ninguna funcionalidad por lo que se escoge el más cercano, para disminuir los retardos en la comunicación.

Una vez creada la aplicación, es necesario implementar la lógica para la decodificación de los mensajes recibidos siguiendo el criterio del identificador previamente definido en la sección 3.1.2.1 (Esquema del programa). Para esto, en la pestaña '*Payload Formats'* se pueden definir una serie de funciones encargadas tanto de la decodificación, como de la validación y conversión de datos en el sentido *uplink*, así como la codificación de los parámetros enviados en el sentido *downlink*. En este caso solo es necesario definir la decodificación de los mensajes *uplink* en función del identificador de trama. El código utilizado para esto se incluye en la figura 15.

```
function decodeUplink(input) {
  var bytes = input.bytes;
  var frame_id = bytes[0];
  var payload = {
    frame_id: frame_id,
  };

  switch (frame_id) {
    case 1:
      // Frame Counter [1, 2]
      payload.frame_counter = (bytes[2] << 8) | bytes[1];

      // Ambientals [3, 8]
      payload.temperature = ((bytes[4] << 8) | bytes[3]) / 100; //comes x100
      payload.humidity = ((bytes[6] << 8) | bytes[5]) / 100; //comes x100
      payload.heat_index = ((bytes[8] << 8) | bytes[7]) / 100; //comes x100

      // Air Quality [9, 12]
      payload.tvoc = (bytes[10] << 8) | bytes[9];
      payload.co2 = (bytes[12] << 8) | bytes[11];

      //Battery [13, 15]
      payload.battery = bytes[13]; // %
      payload.bat_voltaje = (bytes[15] << 8) | bytes[14]; //[mV]
      break;

    default:
      payload.frame_id = 0;
      payload.error = 'TTN: Unknown frame_id';
  }

  return {
    data: {
      bytes: input.bytes,
      payload: payload
    },
    warnings: [],
    errors: []
  };
}
```

Figura 15. Función ejecutada por el servidor de red para decodificar el payload upstream

También es necesario configurar como se comunicará el servidor de red con las capas superiores del sistema. Para eso, en la pestaña '*Integrations'* se definen múltiples opciones para la integración con otros servicios del sector IoT como pueden ser *Ubidots, ThingsSpeak* o *OpenSensors*.

En este caso, y dado que se busca la máxima compatibilidad se opta por el uso de una API HTTP para realizar esta comunicación. TTN ofrece herramientas para realizar un POST a un servidor capaz de procesar los datos enviados cada vez que llega un dato desde los dispositivos. Esta solución es ideal para asegurar la inmediatez de los datos, pero requiere la existencia de un servidor a la escucha, así como implementar la lógica que procese los datos en el formato definido por TTN.

Para este proyecto se opta por usar una herramienta de integración que ofrece una API HTTP sobre la que realizar consultas GET acerca de los datos almacenados. Este tipo de integración es válida para el desarrollo de prototipos, pero no es recomendable para su uso en producción, ya que se sustenta en un servicio externo llamado *Swagger [25]* y los datos almacenados se eliminan tras siete días.

La API ofrecida mediante *Swagger* acepta tres tipos de consultas:

- **/api/v2/devices:** Devuelve una lista con los dispositivos de los que hay datos almacenados.
- **/api/v2/query:** Devuelve todos los mensajes almacenados de la última hora. Acepta un parámetro *'last'* que permite modificar este rango de tiempo. Este se define como un número y una unidad de tal forma que '2h' indica las últimas dos horas y '4d' se refiere a los últimos cuatro días.
- **/api/v2/query/{device-id}:** Similar al caso anterior, pero para obtener los datos correspondientes a uno de los nodos finales de la red. Se accede a ellos mediante el identificador de dispositivo asignado al registrar el nodo en la plataforma TTN.

Para garantizar la seguridad del acceso a estos datos, es necesario añadir una cabecera *'Authorization' a* cada una de las consultas donde se incluya la clave de acceso asociada a la aplicación TTN.

3.3.3. CONFIGURACIÓN DE EQUIPOS

Para que cada uno de los equipos que componen la red puedan comunicarse con el servidor TTN, es necesario darlos de alta de manera que puedan autenticarse. Por un lado, los nodos finales de la red deben estar asociados a una aplicación ya existente. Para ello, y desde la pestaña de la aplicación antes creada, se accede al apartado de dispositivos.

Para llevar a cabo este registro es necesario definir un identificador de dispositivo, así como un EUI que identifique al dispositivo inequívocamente. Tras esto, y dado que se usa autenticación del tipo OTAA, tan solo es necesario definir el EUI de aplicación, el EUI de dispositivo y la clave de aplicación obtenidos desde la web de TTN en el programa que se carga en cada dispositivo para que este se valide y pueda comunicarse con el servidor de red [26].

Por otro lado, es necesario también registrar cada una de las pasarelas que reenvían los mensajes dentro de la red, aunque estas no corresponden a ninguna de las aplicaciones registradas ya que reenvían cada mensaje independientemente de su contenido. Visto desde la arquitectura del estándar LoRaWAN, estos *gateways* se comunican con el servidor de red, mientras que los nodos finales lo hacen con el servidor de aplicación.

Para registrar un *gateway* es necesario definir un identificador en función del EUI del equipo. Este puede ser uno predefinido, o puede modificarse en el equipo, pero ha de coincidir con el definido en el servidor TTN. Es necesario definir también el protocolo a utilizar en la comunicación *Backhaul IP, Lora Basics Station;* así como el plan regional de frecuencias, en este caso el marcado por el estándar y validado por las autoridades: EU868. Finalmente, ha de indicarse a qué *cluster* ha de conectarse el *gateway.* Este

cluster determina la dirección del servidor LNS definido en la configuración de la pasarela [27].

3.4. DESARROLLO DE LA APLICACIÓN MÓVIL

El objetivo de la aplicación móvil desarrollada es ofrecer una interfaz amigable y sencilla que permita visionar los datos obtenidos desde el sistema anteriormente planteado mediante la red LoRaWAN. No pretende ser una solución comercial por lo que cuenta con un funcionamiento sencillo y no ofrece valor añadido a los datos.

En la tabla 3 se realiza una comparación entre las diferentes tecnologías disponibles para llevar a cabo el desarrollo de una aplicación móvil. Si bien algunas de las opciones comparten características, existe un factor diferenciador clave: permitir unificar el desarrollo de esta, independientemente del sistema operativo sobre el que deba ejecutarse. Si bien no realizar una implementación nativa supone una desventaja en el ámbito del rendimiento, la aplicación a desarrollar no requiere gran capacidad de cálculo por lo que las tecnologías multiplataforma suplen estas necesidades con creces.

Dentro de las aplicaciones multiplataforma se opta por el uso del *framework* React Native. Esto es debido a que el único requerimiento relacionado con el hardware sobre el que se debe ejecutar es la capacidad de conectarse a Internet, y este *framework* no requiere ninguna configuración ni código específico para esto. Cabe mencionar que se trata de un *framework* sobre el que se tienen conocimientos previos, por lo que se simplifica el desarrollo y permite explotar las ventajas que ofrece.

Dado que los datos son accesibles mediante una API HTTP, el acceso a estos consta de una simple consulta. Dado que esta API acepta un parámetro para solicitar datos, se opta por añadir también esta funcionalidad a la aplicación ya que ofrece la posibilidad de seleccionar desde cuando se desea visualizar los datos.

La complejidad en este desarrollo radica en cómo mostrar estos datos ya que, aunque en este prototipo se envían siempre los mismos datos en cada mensaje, el objetivo es que el sistema desarrollado soporte varios tipos de mensajes con multitud de datos recogidos. Para esto, en cada mensaje recibido por parte de la aplicación móvil se analizan las variables que lo componen y se van extrayendo aquellas que han de ser mostradas. El objetivo de implementarlo de esta manera es que no sea necesaria una actualización de la aplicación si se añadiesen nuevos tipos de mensajes con nuevas variables, o si los tipos de mensajes ya existentes fueran modificados.

En cuanto a las características y funcionalidad de la aplicación desarrollada, esta permite seleccionar un rango de fechas para las cuales se desean visualizar los datos obtenidos mediante el sistema LoRaWAN. Tras la elección de este rango, y gracias a las funcionalidades que ofrece la API HTTP encargada de entregar estos datos, la aplicación es capaz de obtener los mensajes correspondientes al rango temporal seleccionado y

mostrarlos de una manera amigable y funcional. Si bien en la primera iteración del desarrollo de la aplicación solamente se mostraba la información correspondiente a cada trama, se optó por implementar una nueva funcionalidad que permita realizar un análisis detallado de los datos obtenidos mediante los distintos sensores que componen el sistema. Esta nueva funcionalidad es el renderizado de cada uno de los parámetros leídos en gráficos, los cuales se agrupan por dispositivo y permiten obtener información útil de una manera sencilla e intuitiva.

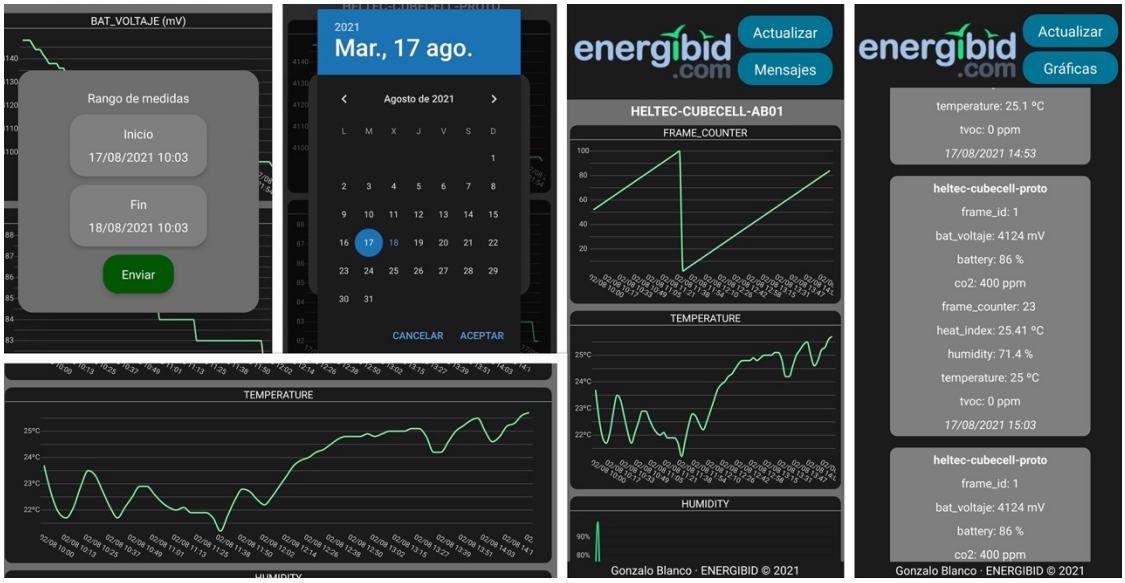


Figura 16. Capturas de la aplicación móvil desarrollada

En la figura 16 se presentan una serie de capturas correspondientes a la aplicación en las que se describe cómo se realiza la selección del rango de tiempo deseado para visualizar los datos (mediante un calendario integrado en el ecosistema Android/iOS), cómo se visualizan los distintos mensajes recibidos por el servidor de red desde los nodos finales, y cómo se visualizan estos datos de manera gráfica con el fin de permitir y facilitar el análisis de los datos obtenidos.

CAPÍTULO 4:

PRUEBAS Y DISCUSIÓN DE RESULTADOS

4. PRUEBAS Y DISCUSIÓN RESULTADOS

Tras el proceso de diseño e implementación del sistema de telemedida, se realiza una fase de pruebas en la que se simulan condiciones adversas para la comunicación del sistema, así como situaciones inesperadas que pudieran darse una vez se instalase. La primera prueba consiste en bloquear las comunicaciones desactivando la pasarela LoRaWAN, escenario que puede darse si se cortara la conexión de esta a Internet o si dejara de recibir alimentación eléctrica. Frente a esta situación, y como ya se ha mencionado, el sistema responde de la manera esperada y es capaz de almacenar los datos no enviados y reenviarlos en cuanto se recuperan las comunicaciones.

También se realizan pruebas de consumo dado que los nodos finales se alimentan mediante baterías. Estas pruebas se realizan con baterías de 1800 mAh de capacidad y con envíos configurados cada 10 minutos, simulando una situación real. Las baterías son cargadas si el nodo es alimentado mediante una fuente de alimentación o un cable USB, tardando en alcanzar su máxima capacidad no más de 3 horas. En cuanto a la duración de la batería esta puede llegar a durar más de dos semanas, descargándose en torno a un 1% cada 4 o 5 horas.

En cuanto a la estabilidad del sistema, los puntos más débiles son los nodos finales. Esto se debe a que, a pesar de que el software está preparado para ello, si se produce algún error grave, no existe la posibilidad de conectarse al nodo de manera remota puesto que no cuenta con conexión a Internet y, siguiendo las restricciones de las comunicaciones LoRaWAN habría que esperar a que este envíe datos para poder comunicarse con él.

Este último punto es la mayor desventaja respecto a otros sistemas similares a los cuales se pretender ofrecer una alternativa. Al no tener disponible una conexión a Internet, se reduce la versatilidad de las comunicaciones y la capacidad de control sobre los dispositivos instalados en campo. A cambio, se ofrece mayor flexibilidad y libertad a la hora de la instalación dado que la cobertura es menos restrictiva en comunicaciones LoRa. El uso de esta tecnología además reduce el coste del hardware, así como el consumo eléctrico, haciéndola ideal para este tipo de sistemas alimentados mediante baterías.

Otra de las diferencias con las alternativas presentadas es que no es necesario gestionar el servidor de red del sistema desarrollado, mientras que en otros proyectos se lleva a cabo la implementación de estos servidores capaces de gestionar y almacenar la recepción de los datos. Gracias al uso de estas *PaaS* se facilita enormemente este desarrollo, y permite ampliar el sistema con más funcionalidades de manera sencilla y económica.

Cabe destacar que, en cuanto al sistema de sensorización desarrollado, los elementos hardware encargados de la recogida de datos ambientales no poseen gran precisión ni son completamente fiables debido a su costo reducido. Si se desplegase este sistema en un entorno de producción, habría que considerar modificar estos elementos

por unos más fiables y precisos. Dado que el código de los nodos finales es modular, implementar la lectura de los nuevos sensores no debe traer mayor complejidad.

CAPÍTULO 5:

CONCLUSIONES Y LÍNEAS FUTURAS

5. CONCLUSIONES Y LÍNEAS FUTURAS

A lo largo de este trabajo se han descrito las diferentes tecnologías que han permitido desarrollar un sistema de telemedida y captación de datos. Se ha hecho énfasis en la base de este sistema, que es la posibilidad de establecer enlaces radio de largo alcance y de gran resistencia frente a las interferencias. Gracias al estándar LoRaWAN, y a la tecnología LoRa, en la que se basa, ha sido posible implementar una red de comunicaciones completa, que además ofrezca funciones adicionales para su gestión y mantenimiento, así como garantizar la seguridad y la fiabilidad en la entrega de datos.

Además de permitir comunicaciones inalámbricas, se ha empleado un gran esfuerzo en desarrollar el software necesario para que los nodos finales de esta red sean capaces de comunicarse de la manera esperada, así como de leer los sensores que permiten recoger los datos. Añadido a esto, el diseño y desarrollo de este programa ha pretendido ofrecer fiabilidad, robustez y un funcionamiento sistematizado, permitiendo añadir otros tipos de sensores o capacidades en un futuro. Entre estas capacidades, podría resultar sencillo implementar la interacción con dispositivos más complejos mediante protocolos de comunicación industrial como puede ser MODBUS.

En cuanto a la red desarrollada, y tal y como se ha descrito a lo largo de esta memoria, basa todo su funcionamiento en el estándar propuesto, por lo que es compatible con otros tipos de implementaciones del protocolo LoRaWAN. En consecuencia, también permite comunicarse con soluciones comerciales más completas siempre que estas sean compatibles con este estándar.

De cara a utilizar el sistema desarrollado a lo largo de este proyecto en un ámbito comercial, será necesario ampliar las variables medidas, con el fin de que la instalación de este ofrezca un servicio más completo. Además, y gracias al desarrollo de una interfaz API HTTP, el sistema será capaz de entregar los datos leídos a multitud de aplicaciones de una manera sencilla y unificada. El objetivo final es poder realizar un análisis completo de los datos obtenidos para ofrecer información adicional al cliente de manera que se le añade un valor añadido a todo el servicio ofrecido.

Para que esto sea posible, y el sistema pueda escalar de una manera eficiente y controlada, se ha propuesto emplear los productos ofrecidos por Amazon Web Services, quien además de permitir realizar esta comunicación entre la red LoRaWAN y las capas superiores, tiene la capacidad de implementar funciones adicionales mediante las IaaS y PaaS (Infraestructuras y Plataformas como Servicio). Entre estas, se presenta como futura posibilidad el empleo de los datos obtenidos para realizar análisis mediante técnicas de Big Data y Machine Learning, ofreciendo así un mayor grado de análisis y valor de los resultados.

Siguiendo esta línea, y si el uso de este tipo de servicios arroja buenos resultados, sería posible migrar la actual plataforma encargada de realizar este análisis de la

empresa Energibid a los servicios ofrecidos por AWS, permitiendo así ahorrar costes, facilitar tareas básicas, y ofrecer un mayor rendimiento y escalabilidad para la misma.

6. PRESUPUESTO

Para llevar a cabo el desarrollo de este proyecto se han adquirido los distintos equipos mencionados a lo largo de esta memoria, de los cuales se desglosa su importe a continuación:

- Pasarela Kerlink iFemtoCell-evolution: 393,25€.
- Placa de desarrollo Heltec CubeCell AB01: 22 €.
- Placa de desarrollo Heltec WiFi LoRa 32 V2: 30€.
- 2 baterías a 3.7V con capacidad de 1800mAh: 2 x 13€.
- 2 sensores de temperatura y humedad DHT22: 2 x 7€.
- 2 sensores de calidad del aire SGP-30: 2 x 17€.
- 2 Protoboards: 2 x 3€.
- 2 resistencias de 10k Ohm.
- Cableado para Protoboard.

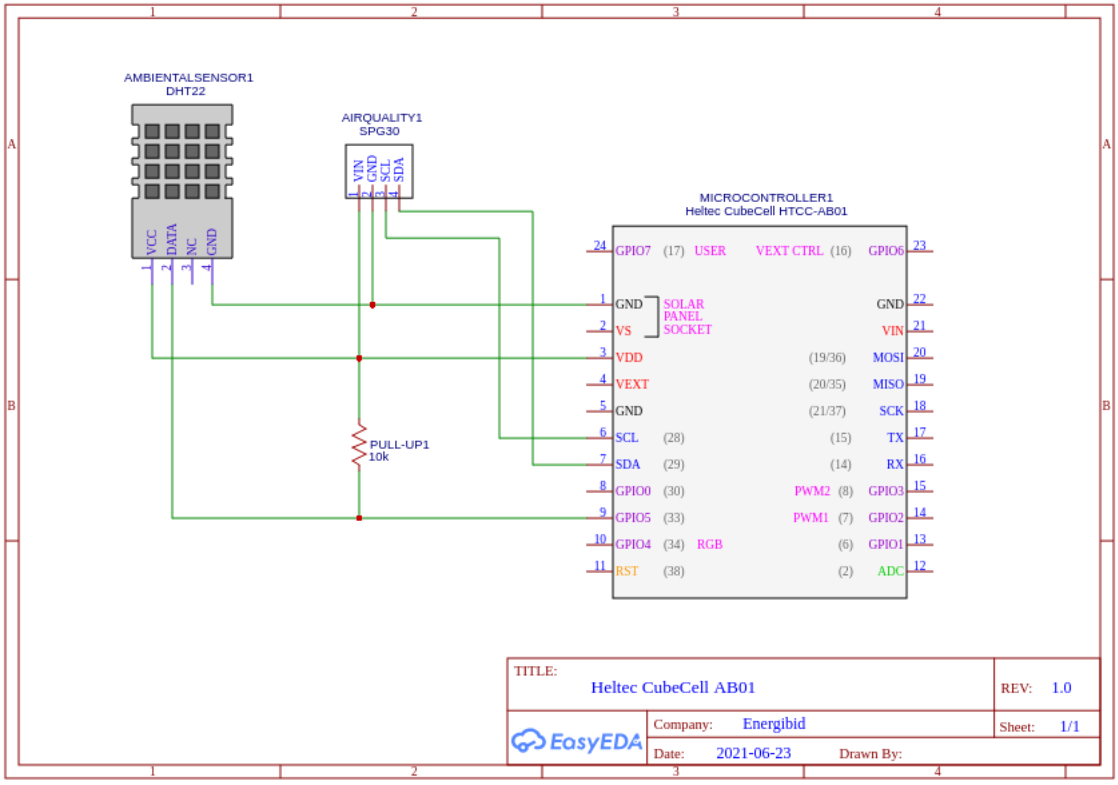
Se estima que el coste total asociado al equipo utilizado asciende a 525,25€.

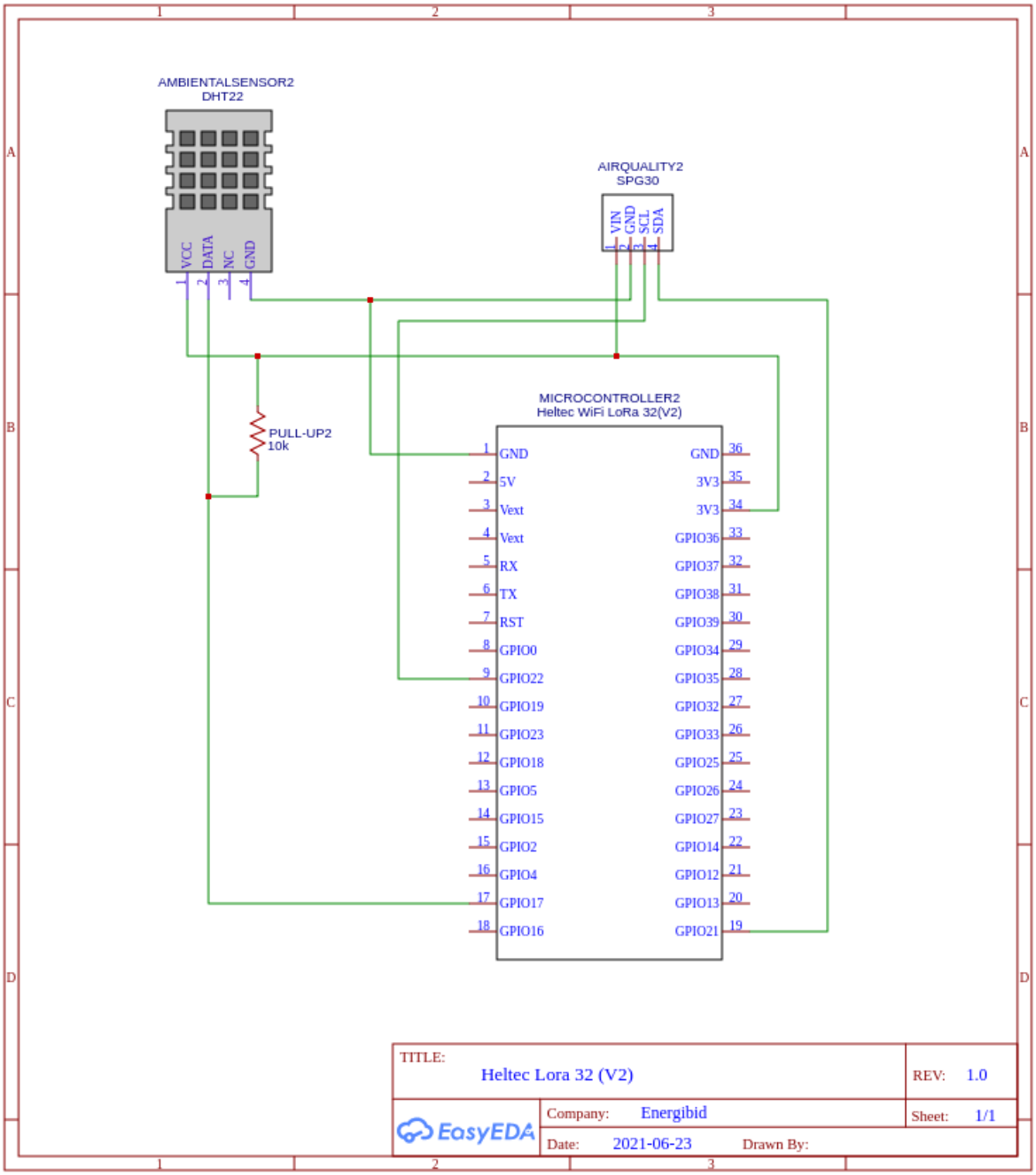
A este equipamiento hay que añadirle el coste asociado al servicio del servidor de red LoRaWAN. Para el desarrollo de este prototipo se ha empleado The Things Network, un servicio gratuito sustentado por la comunidad. De cara a un despliegue comercial se plantea el uso de los servicios ofrecidos por Amazon Web Services, los cuales se facturan en función del coste computacional que conlleva este sistema. Es por esto que es posible predecir el coste que supondrá un despliegue teniendo en cuenta el número de equipos a desplegar, así como la periodicidad con la que se enviarán cada una de las medidas.

Gracias al servicio ofrecido por Amazon que permite estimar el precio [28], se calcula que para un despliegue de 25 dispositivos y un periodo de 10 minutos entre cada envío de datos el coste mensual asociado será cercano a los 0,5 USD. Este importe varía de forma lineal con el número de dispositivos desplegados por lo que cada dispositivo instalado conlleva un coste estimado de 0,2 USD.

En cuanto a las horas de trabajo e investigación necesarias para desarrollar el sistema propuesto, se estima que se han empleado 230 horas de trabajo entre las cuales se incluyen: Las horas dedicadas a la investigación de la tecnología LoRa, así como el estándar LoRaWAN y las alternativas disponibles para su despliegue; las horas dedicadas al desarrollo del software de los nodos finales y la aplicación móvil; las horas dedicadas a la integración y configuración de los elementos que forman la red LoRaWAN; y el tiempo empleado en la realización de pruebas unitarias de los distintos elementos de los que se compone el proyecto, así como del sistema en su conjunto.

7. ANEXO


AMBIENTALSENSOR1
DHT22
VCC
DATA
NC
GND
AIRQUALITY1
SPG30
VIN
GND
SCL
SDA
PULL-UP1
10k
MICROCONTROLLER1
Heltec CubeCell HTCC-AB01
GPIO7 (17) USER
VEXT CTRL (16) GPIO6
GND
SOLAR PANEL SOCKET
VS
VIN
VDD
(19/36) MOSI
VEXT
(20/35) MISO
GND
(21/37) SCK
SCL (28)
(15) TX
SDA (29)
(14) RX
GPIO0 (30)
PWM2 (8) GPIO3
GPIO5 (33)
PWM1 (7) GPIO2
GPIO4 (34) RGB
(6) GPIO1
RST (38)
(2) ADC
TITLE:
Heltec CubeCell AB01
REV: 1.0
EasyEDA
Company: Energibid
Sheet: 1/1
Date: 2021-06-23
Drawn By:


AMBIENTALSENSOR2
DHT22
VCC
DATA
NC
GND
AIRQUALITY2
SPG30
VIN
GND
SCL
SDA
PULL-UP2
10k
MICROCONTROLLER2
Heltec WiFi LoRa 32(V2)
GND
5V
Vext
Vext
RX
TX
RST
GPIO0
GPIO22
GPIO19
GPIO23
GPIO18
GPIO5
GPIO15
GPIO2
GPIO4
GPIO17
GPIO16
GND
3V3
3V3
GPIO36
GPIO37
GPIO38
GPIO39
GPIO34
GPIO35
GPIO32
GPIO33
GPIO25
GPIO26
GPIO27
GPIO14
GPIO12
GPIO13
GPIO21
TITLE:
Heltec Lora 32 (V2)
REV: 1.0
EasyEDA
Company: Energibid
Sheet: 1/1
Date: 2021-06-23
Drawn By:

INDICE DE FIGURAS:

INDICE DE TABLAS:

[1] Daniel Ibaseta, et al. *An IOT platform for indoor air quality monitoring using the Web of Things*. (2019). Retrieved 27 May 2021, from https://digibuo.uniovi.es/dspace/bitstream/handle/10651/53764/AIR19005FU1.pdf

[2] *LoRaWAN. What is it?* LoRa Alliance Documentation (2015). Retrieved 20 April 2021, from https://lora-alliance.org/wp-content/uploads/2020/11/what-is-lorawan.pdf

[3] Rafia Mumtaz et al. (2021). *Internet of Things (IoT) Based Indoor Air Quality Sensing and Predictive Analytic—A COVID-19 Perspective*. Retrieved 4 June 2021, from https://www.mdpi.com/2079-9292/10/2/184.

[4] V. Barot and V. Kapadia, "Air Quality Monitoring Systems using IoT: A Review," *2020 International Conference on Computational Performance Evaluation (ComPE)*, 2020, pp. 226-231, doi: 10.1109/ComPE49325.2020.9200053. Retrieved 6 June 2021, from https://ieeexplore-ieee-org.ponton.uva.es/document/9200053.

[5] P. Edward, M. El-Aasser, M. Ashour and T. Elshabrawy, "Interleaved Chirp Spreading LoRa as a Parallel Network to Enhance LoRa Capacity," in *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3864-3874, 1 March1, 2021, doi: 10.1109/JIOT.2020.3027100. Retrieved 20 May 2021, from https://ieeexplore-ieee-org.ponton.uva.es/document/9207749.

[6] D. Eridani, E. D. Widianto, R. D. O. Augustinus and A. A. Faizal, "Monitoring System in Lora Network Architecture using Smart Gateway in Simple LoRa Protocol," 2019 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI), 2019, pp. 200-204, doi: 10.1109/ISRITI48646.2019.9034612. Retrieved 25 May 2021, from https://ieeexplore-ieee-org.ponton.uva.es/document/9034612

[7] *P. A. Barro, M. Zennaro and E. Pietrosemoli, "TLTN – The local things network: on the design of a LoRaWAN gateway with autonomous servers for disconnected communities," 2019 Wireless Days (WD), 2019, pp. 1-4, doi: 10.1109/WD.2019.8734239*. Retrieved 23 May 2021, from https://ieeexplore-ieee-org.ponton.uva.es/document/8734239.

[8] *A. Gladisch, S. Rietschel, T. Mundt, J. Bauer, J. Goltz and S. Wiedenmann, "Securely Connecting IoT Devices with LoRaWAN," 2018 Second World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4), 2018, pp. 220-229, doi: 10.1109/WorldS4.2018.8611576*. Retrieved 23 May 2021, from https://ieeexplore-ieee-org.ponton.uva.es/document/8611576.

[9] *Semtech UDP Packet Forwarder*. The Things Network Documentation. (2021). Retrieved 13 May 2021, from https://www.thethingsnetwork.org/docs/gateways/packet-forwarder/semtech-udp/.

[10] *LoRa Basics™ Station*. Semtech Documentation. (2021). Retrieved 10 May 2021, from https://doc.sm.tc/station/#.

[11] *ChirpStack architecture*. ChirpStack Documentation. (2021). Retrieved 6 May 2021, from https://www.chirpstack.io/project/architecture/.

[12] *What is AWS IoT*. Amazon Web Services Documentation. (2021). Retrieved 28 April 2021, from https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html.

[13] *AWS IoT Core for LoRaWAN*. Amazon Web Services Documentation. (2021). Retrieved 28 April 2021, from https://docs.aws.amazon.com/iot/latest/developerguide/connect-iot-lorawan.html.

[14] *Learn | The Things Network*. The Things Network Documentation. (2021). Retrieved 20 April 2021, from https://www.thethingsnetwork.org/docs/.

[15] *React Native. Learn once, write anywhere*. React Native. (2021). Retrieved 3 May 2021, from https://reactnative.dev/.

[16] José Daniel Rodríguez Munca. *Dispositivo LoRa de comunicación a largo alcance y bajo consume energético para aplicaciones del ámbito del desarrollo*. Universidad Politécnica de Madrid (2017). Retrieved 2 May 2021, from http://oa.upm.es/44890/1/TFM_JOSE_DANIEL_RODRIGUEZ_MUNCA.pdf.

[17] *Digital relative relative humidity & temperature sensor AM2302/DHT22*. Adafriut datasheet. Retrieved 30 April 2021, from https://cdn-shop.adafruit.com/datasheets/Digital+humidity+and+temperature+sensor+AM2302.pdf.

[18] *Datasheet SGP30. Indoor Air Quality Sensor for TVOC and CO2eq Measurements*. Sensirion (2020). Retrieved 30 April 2021, from https://www.sensirion.com/fileadmin/user_upload/customers/sensirion/Dokumente/9_Gas_Sensors/Datasheets/Sensirion_Gas_Sensors_Datasheet_SGP30.pdf.

[19] *Adafruit SGP30 TVOC/eCO2 Gas Sensor*. Adafruit Library (2018). Retrieved 30 April 2021, from https://www.mouser.es/pdfdocs/adafruit-sgp30-gas-tvoc-eco2-mox-sensor.pdf.

[20] *Arduino-LMIC library ("MCCI LoRaWAN LMIC Library").* MCCI Corporation. Retrieved 20 April 2021, from https://github.com/mcci-catena/arduino-lmic.

[21] *Heltec Cubecell Series Arduino Development Environment.* Heltec Automation. Retrieved 21 April 2021, from https://github.com/HelTecAutomation/CubeCell-Arduino.

[22] Wirnet iFemtoCell-evolution Documentation. Kerlink (2021). Retrieved 17 April 2021, from official vendor site.

[23] *Keros 4.3.3: CPF configuration.* Kerlink (2021). Retrieved 28 April 2021, from https://wikikerlink.fr/wirnet-productline/doku.php?id=wiki:lora:keros_4.3.3:cpf_configuration.

[24] *Keros application configuration.* Kerlink (2021). Retrieved 28 April 2021, from https://wikikerlink.fr/wirnet-productline/doku.php?id=wiki:keros_custo:keros_applications_configuration.

[25] *Swagger, API Documentation & Design Tools for Teams*. Swagger. (2021). Retrieved 12 May 2021, from https://swagger.io/.

[26] *Adding Devices*. The Things Network Documentation. (2021). Retrieved 13 May 2021, from https://www.thethingsnetwork.org/docs/devices-and-gateways/adding-devices/.

[27] *Adding Gateways*. The Things Network Documentation. (2021). Retrieved 13 May 2021, from https://www.thethingsnetwork.org/docs/devices-and-gateways/adding-gateways/.

[28] *AWS Pricing Calculator, Configure AWS IoT Core*. Amazon Web Services (2021). Retrieved 26 May 2021, from https://calculator.aws/#/createCalculator/IoTCore