UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO

INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Particionado de red de acceso radio en redes LoRaWAN

Departamento de Teoría de la Señal, Telemática y Comunicaciones

Autor

Martín Torres Antúnez

Director

Jorge Navarro Ortiz

Escuela Técnica Superior de Ingenierías de Informática y Telecomunicación.

—

Granada, Junio de 2022

UNIVERSIDAD DE GRANADA

Particionado de red de acceso radio en redes LoRaWAN

Departamento de Teoría de la Señal, Telemática y Comunicaciones

Autor
Martín Torres Antúnez

Director
Jorge Navarro Ortiz

Particionado de red de acceso radio en redes LoRaWAN

Martín Torres Antúnez

Palabras clave: IoT, LoRa, LoRaWAN, Resource block, Slices, RSSI, SNR.

Resumen

Hoy en día, toda tecnología relacionada con el Internet de las cosas (IoT) está en auge, ya que sirve para interconectar todos los aparatos de nuestro alrededor y proporcionarnos una vida más cómoda. Esta tecnología incrementa día tras día su fama debido al empleo de dispositivos de bajo consumo para su utilización y a su bajo coste.

Entre estas redes de bajo consumo podemos destacar el uso de LoRa y LoRaWAN como las principales de este proyecto. Estas tienen unas características principales que se centran en su baja potencia, y por ende bajo consumo energético, y su gran alcance y cobertura. Debido a este amplio rango de cobertura es de gran interés el ser capaz de dividir esta red en varios *resource block* (RB) o *slices* con los que podemos asignar porciones de red a diferentes empresas o usuarios sin que tengan que colisionar entre sí sus señales. Esta técnica se denomina particionado de red y se ha conseguido en otras tecnologías como 5G.

El objetivo y la finalidad de este proyecto ha sido crear una manera de dividir la red en estos RB. También, se ha pensado la manera de asignarlos según un criterio determinado que puede ser distintas características de la señal como el RSSI o el SNR. Se ha conseguido con éxito la comunicación entre dispositivos y la asignación de RB teniendo en cuenta el SNR. Las pruebas según RSSI no se han podido realizar debido a que la cobertura se pierde antes de que el algoritmo funcione.

Project Title: Project Subtitle

Martín Torres Antúnez

Keywords: IoT, LoRa, LoRaWAN, Resource block, Slices, RSSI, SNR.

Abstract

Nowadays, all the tecnology that it's related to IoT is one the rise, since it is used to interconnect all the devices from our surroundings and to make us live a more comfortable life. This technology increases its fame day after day due to the use of low energy consume machines and its low price.

Between these networks we can highlight LoRa and LoRaWAN as the two main subjects of this project. This two have some main characteristics that are focused on the low power, and because of that low energy consumption, and its high range and coverage. Due to this high coverage it's of a great interest to be able to divide the network into several resource blocks (RB) or slices with which we can assign portions of the network to different companies or users without their signals colliding. This technique is named network slicing and it has been achieved in other technologies such as 5G.

The objective of this Project has been to create a way to divide the network into RB. As well, it has been thought a way to assign them according to a certain standard which can be several characteristics of the signal such as the RSSI or SNR. The communications between devices and the assign of a RB according to the SNR has been achieved. The tests according to RSSI have not been able to be achieved since the coverage was lost before the algorithm could work.

Yo, **Martín Torres Antúnez**, alumno de la titulación INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 77033939Q, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Martín Torres Antúnez

Granada a 5 de Julio de 2022.

D. **Jorge Navarro Ortiz**, Profesor del Departamento Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado *Particionado de red de acceso radio en redes LoRaWAN*, ha sido realizado bajo su supervisión por **Martín Torres Antúnez**, y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 11 de julio de 2022

El director:

Jorge Navarro Ortiz

Agradecimientos

Gracias a mi familia por todo su apoyo y a mi tutor Jorge por su paciencia y todas las tutorías y ayudas recibidas.

Índice general

Índice de Figuras.

Índice de tablas.

Glosario

ABP: Activation By Personalization.
ACK: Acknowledgement.
ADR: Adaptive Data Rate.
AES: Advanced Encryption Standard.
ALOHA: Additive Links On-line Hawaii Area.
API: Application Programming Interface.
BW: Bandwith.
CR: Code Rate.
CSS: Chirp Spread Spectrum.
ETSI: European Telecommunications Standards Institute.
FSK: Frequency Shift Keying.
FTP: File Transfer Protocol.
GW: Gateway.
HTTP: Hypertext Transfer Protocol.
IEEE: Institute of Electrical and Electronics Engineers.
IoT: Internet of Things.
IP: Internet Protocol.
ISM: Industrial, Scientific and Medical.
JSON: JavaScript Object Notation.
JWT: JSON Web Token.
LoRa: Long Range.
LoRaWAN: Long Range Wide Area Network.
LPWAN: Low-Power Wide Area Network.
MAC: Medium Acess Control.
MQTT: Message Queue Telemetry Transport.
NB-IoT: NarrowBand-IoT.
OTAA: Over The Air Activation.
QoS: Quality of Service.
RB: Resource Block.
RSSI: Received Signal Strength Indicator.
SDGs: Sustainable Development Goals.
SDN: Software Defined Networking.
SF: Spreading Factor.
SNR: Signal to Noise Ratio
SSH: Secure Shell.
TCP: Transmission Control Protocol.
VLAN: Virtual Local Area Netowork.
WAN: Wide Area Network.

Capítulo 1

1. Introducción

En este capítulo se presentará una breve introducción a la materia en la cual vamos a basar este proyecto, se incluirá el contexto, los objetivos de este y la motivación que se ha seguido para la realización de este trabajo.

1.1 Contexto y motivación

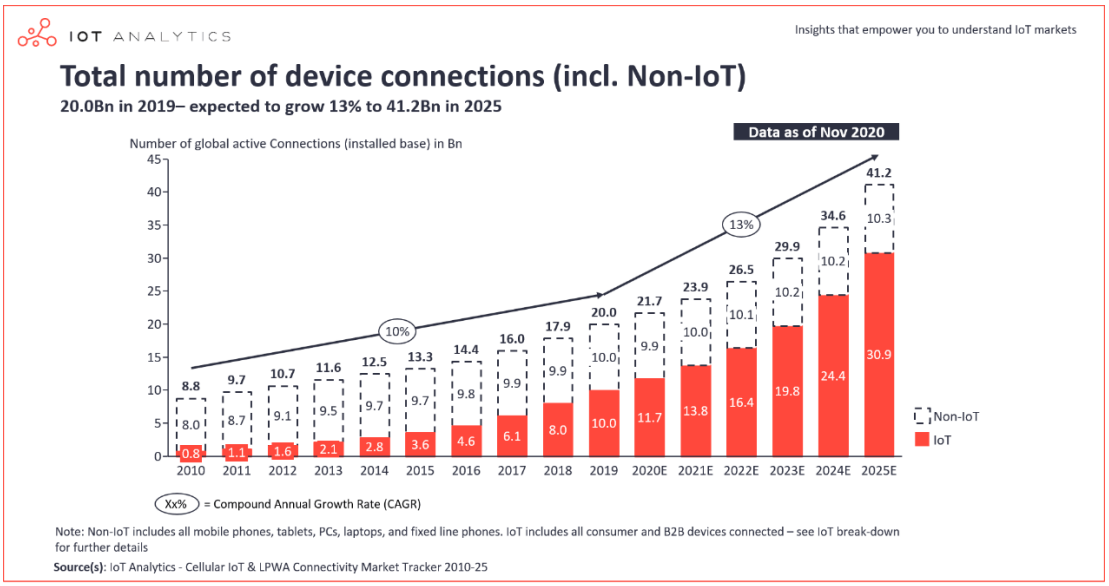
Actualmente, las comunicaciones inalámbricas son de una importancia vital ya que gracias a ellas millones de personas están interconectadas mediante internet, por lo cual todas ellas pueden disfrutar de las numerosas ventajas y beneficios que proporciona esta tecnología.

Prácticamente todos los sectores de la economía global se basan en la utilización de las tecnologías inalámbricas, como el 5G, inteligencia artificial o IoT, de manera fundamental para mejorar nuestras vidas a un ritmo sin precedentes. De hecho, en un futuro no muy lejano podríamos tener la capacidad de acelerar el progreso hasta poder cumplir con cada uno de los 17 objetivos de desarrollo sostenible propuesto por la ONU (SDGs) [1].

Sin embargo, aunque todas las tecnologías mencionadas anteriormente son de gran interés, en la que nos vamos a centrar en esta parte es en las tecnologías IoT. Realmente el concepto de IoT es un concepto muy sencillo, se trata de coger todas las cosas útiles de nuestro alrededor e interconectarlas mediante internet para que puedan interactuar entre sí y con nosotros de manera digital. Básicamente consiste en incorporar tecnología a las cosas más sencillas de nuestro alrededor para facilitarnos la vida aún más.

Gran parte del gran crecimiento de la conexión global actual se debe al crecimiento de este tipo de dispositivos. Según el último análisis datado en noviembre de 2020 [2] (Figura 1.1) podemos ver cómo a partir del año 2018 los dispositivos IoT alcanzan una gran popularidad en su manufacturación y las previsiones para los siguientes años nos indican que esta tendencia seguirá creciendo.

Figura 1.1: Número de conexiones de dispositivos IoT según IoT Analytics [2].



IoT se caracteriza por que sus funcionalidades se basan en envío de datos de tamaño reducido, bajo consumo de energía y tienen una gran rentabilidad. Las redes se dividen según el tamaño de su área de cobertura. Para este trabajo nos interesará especialmente las redes inalámbricas LPWAN (Low-Power Wide Area Network). Estas son redes que se caracterizan por tener un área de cobertura amplia (máx. 20 km) y de baja potencia. Los estándares principales de esta red inalámbrica son Sigfox, LoRaWAN y NB-IoT.

Debido a la variedad de situaciones en las que los operadores son incapaces de proporcionar cobertura en todo tipo de situaciones, ésta última tecnología nos será particularmente útil para proporcionar conectividad en este tipo de regiones en las que, de otra manera, sería muy complicado mantener una conexión entre dispositivos. Podemos usar esta tecnología, por ejemplo, para tener controlado el nivel de humedad y temperatura en una granja o para controlar animales en peligro de extinción mediante el uso de chips y sensores.

En este contexto, la motivación de este proyecto consistirá en realizar un particionado de la red, es decir, una técnica que permite trocear una infraestructura de red en diferentes segmentos independientes entre sí. Esto permite tener un operador dueño de la infraestructura que alquila parte de su red a unos operadores que proporcionan servicios a los usuarios. Este particionado de red se puede aplicar tanto a la red de acceso radio como a la red troncal, siendo un tema de especial interés en redes 5G. En este proyecto se plantea la posibilidad de realizar un particionado de la red de acceso radio para redes LoRaWAN.

1.2 Objetivos

Los principales objetivos de este proyecto son los siguientes:

- Investigación sobre las redes LPWAN y más concretamente el estándar LoRaWAN, saber cómo funciona y las diferencias entre esta tecnología y las demás tecnologías IoT.

- Particionado de la red en diferentes segmentos para permitir a un operador asignar cada trozo a un dueño distinto.

- Montar un servidor de red por el que se puedan comunicar dispositivos reales mediante python.

- Asignación de un RB concreto dentro de un *slice* en la mota según RSSI o SNR.

1.3 Estructura de la memoria

En esta sección se va a explicar brevemente la disposición que se ha seguido en este proyecto. Se describirá en unas pocas palabras cada capítulo y se explicará el motivo de su aparición en este TFG.

- **Capítulo 1, Introducción:** En este capítulo se va a describir el contexto, la motivación y los objetivos que han llevado a que sea necesario el desarrollo de este trabajo.

- **Capítulo 2, Estado del arte:** En esta parte se describirá algunos proyectos que trabajan contenidos similares a éste. Se pondrán ejemplos y lo que diferencia a este proyecto de los que ya están hechos.

- **Capítulo 3, Planificación y coste:** Aquí se explicará cuál ha sido el orden de las tareas que se han ido realizando hasta la finalización del proyecto y una breve descripción de cada una. Se añadirá un esquema de Gantt para complementar la tarea. Además, se hará una revisión de todos los recursos que han sido necesarios y los medios que tendríamos que disponer para hacer desde cero este TFG.

- **Capítulo 4, Fundamentos teóricos:** Esta sección se dedicará a la explicación de toda la teoría en la que se ha basado el proyecto, además de la necesaria para entenderlo.

- **Capítulo 5, Herramientas fundamentales:** Este capítulo se centrará en la descripción de todos los programas y tecnología usada. Se hará una descripción dedicada a las aplicaciones fundamentales que se han utilizado, arquitectura y como se comunican entre sí para que funcione el proyecto.

- **Capítulo 6, Diseño e implementación:** En esta parte se va a desarrollar el algoritmo, todos los pasos seguidos para conseguir que funcione y la implementación de este en el TFG. También se explicará el funcionamiento del mismo.

- **Capítulo 7, Resultados y simulaciones:** Aquí se presentarán los resultados de las pruebas realizadas, se podrá ver el éxito del algoritmo y una serie de simulaciones para comprobar los resultados.

- **Capítulo 8, Conclusiones y trabajos futuros:** Por último, se hará un pequeño resumen en el que se concluirá el proyecto y se propondrán trabajos futuros en los que se mejore el proyecto actual o variaciones del mismo.

Capítulo 2

2. Estado del arte

En este capítulo se informará de diversos proyectos o artículos científicos que tengan relación con el particionado de red en LoRaWAN o en otras tecnologías parecidas. Debido a que no hay muchos proyectos que realicen particionado de red LoRaWAN asignando un RB para cada dispositivo, se ha hecho un subapartado en vez de hacer un capítulo dedicado entero a la investigación de artículos parecidos.

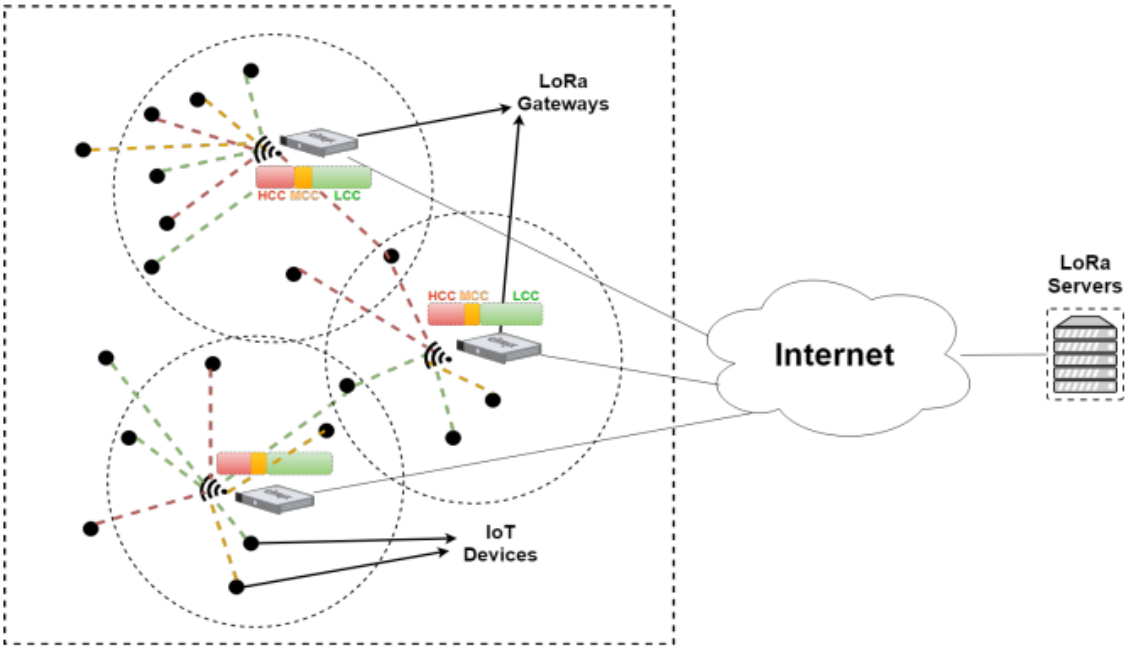
Adaptive Dynamic Network Slicing in LoRa Networks.

Este artículo está distribuido por HAL open science, el cuál es un depósito dónde autores de todos los campos académicos pueden depositar sus artículos.

En este proyecto consideran que el particionado de la red LoRaWAN consiste en dividir la red según los requerimientos QoS de cada dispositivo, en términos de retraso y fiabilidad dependiendo de la aplicación IoT que se esté ejecutando. Una partición se define como una división del hardware físico de red, más específicamente el de las LoRa GWs, donde el ancho de banda de cada GW es dividido en diferentes partes como se muestra en la figura 2.1. El principal objetivo detrás del particionado es dividir virtualmente la red mediante la reserva de recursos para cada partición en el mismo dispositivo físico. Cada partición se caracteriza por un SF prioritario y un ancho de banda asignado a ese SF. Cada dispositivo se le asigna el SF y el ancho de banda según las necesidades de los servicios que aporten. Una vez asignadas cada partición a un dispositivo, cada GW reservará dinámicamente los recursos según una estimación. [3]

Como se puede ver en la imagen, varios LoRa GWs recibirán los paquetes, pero solo una partición reenviará el paquete al servidor LoRa para evitar duplicados.

Figura 2.1: Arquitectura del particionado **Adaptive Dynamic Network Slicing in LoRa Networks.** [3]



Este artículo sería un ejemplo de todos los proyectos parecidos que hay con LoRaWAN realizando un particionado de red. Que se haya encontrado, no hay ningún proyecto que divida la red en RB, asignándole un SF y un canal a un dispositivo.

También es importante decir que, a pesar de que en LoRaWAN este tipo de tecnología no es muy común con la estrategia que estamos planteando en este TFG, en el caso de 5G es muy popular y se usa en una alta cantidad de proyectos y tecnologías para realizar multitud de servicios y aplicaciones. Esto permite dar soporte a necesidades muy diversas, tanto a nivel de usuario, profesional e industrial. La diferencia que tiene esta tecnología en comparación a la virtualización típica de la red es que el particionado se puede implementar de manera flexible y dinámica, sin tener que ser específica y compleja como era la virtualización en, por ejemplo, VLANs. Esto ha sido gracias en parte al florecimiento de la tecnología SDN.

Algunos de los usos que se le ha dado al particionado de red de 5G ha sido, por ejemplo, el integrarlo con vehículos para conseguir que intercambien datos entre sí, conducción autónoma, conducción teledirigida, seguridad y eficiencia del tráfico mejorada, streaming de video integrada en los vehículos, diagnóstico y gestión remota del vehículo [4]. Como se puede ver, solo en el enfoque de los vehículos hay muchos proyectos interesantes por lo que el objetivo de este proyecto es conseguir una utilidad parecida y superior para el particionado de red en LoRaWAN.

Capítulo 3

3. Planificación y coste

En este capítulo estimaremos el tiempo estimado que se le ha dedicado a este proyecto además del presupuesto que se habría de disponer para poder realizar este proyecto desde cero. Esto incluye todos los recursos y costes tanto económicos como humanos empleados.

3.1 Planificación temporal.

La siguiente lista es una enumeración de las tareas que se han realizado para desarrollar en su plenitud todos los objetivos propuestos en este proyecto, además de una breve descripción de cada una:

1. *Revisión de las tecnologías LoRa/LoRaWAN.*
   Consiste en el estudio de ambas tecnologías para tener una primera idea sobre lo que se va a trabajar. Además, asienta unos fundamentos teóricos necesarios para las próximas tareas.

2. *Comprensión de los logros que se quieren conseguir con este proyecto.*
   Esta tarea consiste en el entendimiento del por qué este proyecto es necesario y qué aplicaciones puede tener y los problemas que puede solventar y facilitar.

3. *Estudio del arte de otros proyectos similares de partición de red.*
   Para tener referencias sobre otros trabajos similares realizados en otro tipo de tecnologías como 5G o Cloud Computing. De esta manera, al visualizar estos artículos, se tiene una idea más clara de cómo aplicar los conceptos aprendidos a nuestra propia tecnología.

4. *Instalación del entorno de trabajo y configuración de este.*
   Instalar y configurar todos los programas para que funcionen perfectamente en nuestro lugar de trabajo, con la adecuada instalación de paquetes y con las versiones y herramientas para trabajar eficientemente.

5. *Comprensión y adaptación a los dispositivos a usar.*
   Tanto de las motas, como de los *gateways*. Entender con diferentes códigos

cómo funcionan y se comunican para consecuentemente poder aplicar esos ejemplos a nuestro código original.

6. *Esquematizar las tareas a realizar.*
   Previo a la realización del código, para ver cuáles son las funciones que van a ocupar el servidor de red y el dispositivo. También ver como dividir recursos para la transmisión y cómo transmitirlos.

7. *Creación de las bases de datos y escritura del código principal.*
   Aquí se realiza el código con el que vamos a trabajar y a realizar las pruebas del proyecto. También se crean las bases de datos con las que se trabajarán y se almacenarán los datos necesarios.

8. *Realización de pruebas, discusión y análisis de los resultados.*
   Se harán varios test de prueba asignándoles distintos RB para ver si funciona el particionado y que se comunican satisfactoriamente entre el servidor y ellas mismas. También se hará un análisis de los resultados obtenidos para ver si coinciden con los objetivos del proyecto.

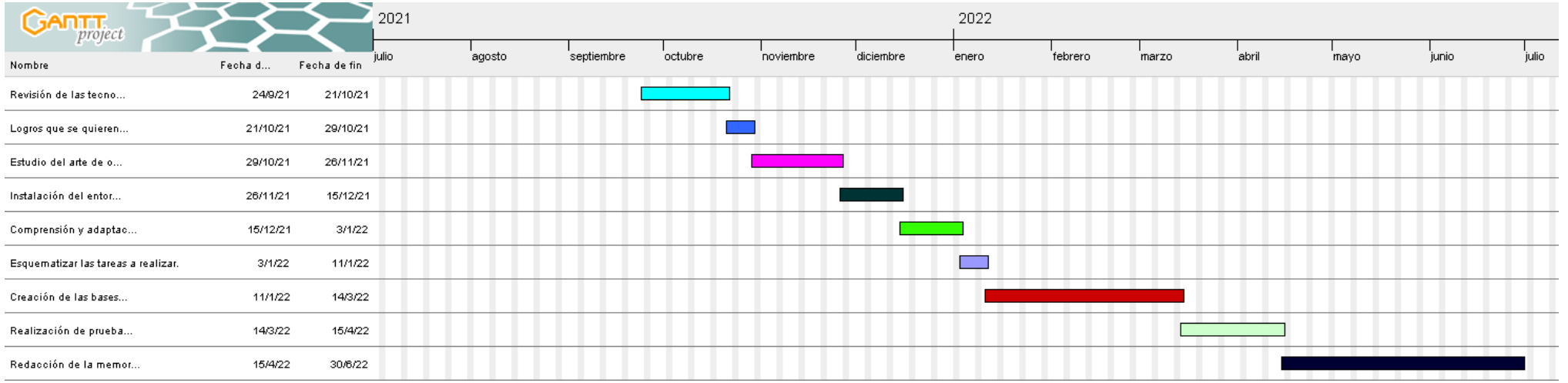
9. *Redacción de la memoria y revisión de esta.*
   Realización final de la escritura del proyecto en la que se recogerá toda la información, detalles y resultados obtenidos durante el desarrollo del TFG.

En la tabla 3.1, se ha realizado una planificación temporal inicial para la realización de todas las tareas mencionadas previamente en este punto. En la figura 3.1 se muestra un diagrama de Gantt en el que se muestra de una manera visual la división del tiempo.

Tabla 3.1: Planificación

Nombre de la tarea	Duración (días)
Revisión de las tecnologías LoRa/LoRaWAN.	20
Logros que se quieren conseguir con este proyecto.	7
Estudio del arte de otros proyectos similares de partición de red.	21
Instalación del entorno de trabajo y configuración de este.	14
Comprensión y adaptación a los dispositivos a usar.	14
Esquematizar las tareas a realizar.	7
Creación de las bases de datos y escritura del código principal.	45
Realización de pruebas con diferentes dispositivos y discusión y análisis de los resultados.	25
Redacción de la memoria y revisión de esta.	55

Figura 3.1: Diagrama de Gantt



3.2 Planificación de recursos

En este apartado se describirán todos los recursos utilizados en el desarrollo de este proyecto. Estarán divididos en recursos hardware, software y humanos.

3.2.1 Recursos hardware

Los recursos hardware se refieren a todos aquellos dispositivos, materiales y medios necesarios para el correcto funcionamiento y avance del trabajo.

- **Portátil personal:** En el que se han producido todas las simulaciones y casi la completa totalidad del proyecto, además de la escritura de la memoria. El ordenador es un modelo "Acer Nitro 5 AN515-45-R19D". Procesador "AMD Ryzen 5", memoria RAM de 16 GB y tarjeta gráfica "RTX 3060".

- **Raspberry Pi 4:** Necesaria para toda la configuración del servidor de red y su monitorización. También contiene la base de datos necesaria para la comunicación entre dispositivo y servidor. Junto a la Raspberry Pi se ha necesitado un cable de tipo C para la alimentación de energía y un cable ethernet para conectarla al router.

- **Motas:** Dispositivos encargados de enviar mensajes de tipo *request* para pedirle al servidor que se le asignen un RB que recibirán en forma de ACK. Una vez recibido tendrán que mantenerse en el canal y SF asignado para así realizar el particionado. Las motas son del modelo LOPY4/EXPANSION BOARD [5].

- **Gateway:** Es el dispositivo que habilita la comunicación entre las motas y el servidor. Generan la red inalámbrica LoRaWAN para dar cobertura a todas las motas, comunicarse con ellas para recibir o transmitir datos y notificar al servidor de toda la información recogida/transmitida correctamente. El modelo es PyGate LOPY+ESCUDO PYGATE [6].

3.2.2 Recursos Software

Los recursos software son todos aquellos programas usados para la realización y funcionamiento del TFG.

- **Atom:** Es un editor de código fuente de código abierto. La gran parte de sus paquetes son de software libre y es el entorno de programación que se ha escogido para programar con MicroPython. Es donde se ha desarrollado la lógica del proyecto y el algoritmo de particionado [7].
- **Chirpstack platform:** Una plataforma de código abierto que proporciona componentes para las redes LoRaWAN. Tiene una API que facilita la gestión de los dispositivos [8].

- **GanttProject:** Herramienta utilizada para realizar la planificación de tareas

[9].

- **Word:** Editor de texto usado para la escritura de la memoria.

- **MobaXterm:** Programa que facilita una conexión SSH y FTP entre el ordenador y la Raspberry Pi [10].

Todos los recursos software mencionados son de código abierto o de software gratuito por lo que su uso y descarga no han costado ningún tipo de coste.

3.2.3 Recursos humanos

Los recursos humanos se refieren al número de personas involucradas en el proyecto y las horas que cada una le ha dedicado al mismo. Las horas calculadas para el estudiante son una estimación sobre el tiempo dedicado a cada una de las tareas explicadas, además de todas las reuniones, investigación propia y errores que se hayan cometido durante el transcurso del proyecto. Las horas calculadas para el tutor se han estimado según el tiempo dedicado a la explicación de todas las tutorías pedidas por el alumno, escritura de correos de respuesta a dudas y revisión del proyecto para comprobar que todo es correcto. Esto se muestra en la tabla 3.2.

Las personas involucradas en el proyecto han sido dos:

- **Martín Torres Antúnez (Alumno):** Estudiante de la Universidad de Granada, cursando el grado de Ingeniería de Tecnologías de Telecomunicación.

- **Jorge Navarro Ortiz (Tutor):** Profesor Titular a la Universidad de granada del departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada.

Tabla 3.2: Recursos humanos

Nombre	Horas dedicadas
Martín Torres Antúnez	500
Jorge Navarro Ortiz	40

3.2.4 Presupuesto del proyecto

Teniendo en cuenta todos los aspectos anteriores vamos a calcular en este punto el presupuesto del proyecto considerando que tuviésemos que adquirir todos los recursos necesarios desde cero. Se han realizado tres tablas para calcular el presupuesto para cada recurso y una tabla con la estimación del presupuesto final con la suma de estos tres.

Teniendo en cuenta que la vida útil media de un portátil es de 36 meses y en este caso se ha utilizado durante 9 meses, el coste será de 9/36 el precio del ordenador.

Tabla 3.3: Presupuesto recursos hardware

Recurso	Unidades	Precio por Unidad
Portátil personal	1	299,75€
Raspberry Pi	1	75€
Motas + Antenas + Expansion Board	1	60€
Gateway + Antenas + Expansion Board	1	100€

Por lo que el presupuesto para los recursos hardware sería en total 534,75€

Tabla 3.4: Presupuesto recursos software

Recurso	Coste
Atom	0€
Chirpstack platform	0€
Gantt Project	0€
Word	0€
MobaXterm	0€

Por lo que el presupuesto para los recursos software es de 0€ ya que todo el software se ha intentado buscar gratuito y de código abierto.

Tabla 3.5: Presupuesto para recursos humanos

Persona	Horas invertidas	Precio por hora
Martín Torres Antúnez	500	25€
Jorge Navarro Ortiz	40	50€

En este caso se han estimado el precio por hora según lo que ganaría un ingeniero júnior, que serían 25€ por hora. El presupuesto total de este apartado sería de 14.500€.

Tabla 3.6: Presupuesto final

Tipo de recursos	Presupuesto
Hardware	534,75€
Software	0€
Humano	14.500€

Por lo que el presupuesto estimado para este proyecto, en el caso de que se empezase desde cero, sería de un total de 15.034,75€.

Capítulo 4

4. Fundamentos teóricos

En este capítulo explicaremos los conceptos teóricos necesarios para comprender en su totalidad todas las partes de este proyecto. Entre los términos a explicar entrarán LoRa y LoRaWAN, las bandas ISM y las limitaciones y características de ambos.

4.1 LoRa y LoRaWAN

Lora es una técnica de modulación de propagación del espectro, proveniente de la tecnología Chirp Spread Spectrum (CSS) [11]. Fue desarrollada por una empresa francesa llamada Cicleo, pero finalmente fue adquirida por Semtech [12]. LoRa es una plataforma de largo alcance y área amplia y baja potencia que se ha convertido en una de las plataformas inalámbricas insignes de IoT. Los dispositivos LoRa y redes como la de LoRaWAN permiten solventar algunos de los desafíos más complejos del planeta como: gestión de la energía, reducción de los recursos naturales, prevención de desastres, etc…

LoRa y LoRaWAN definen juntos una LPWAN (Low Power, Wide Area Network), diseñada para conectar de manera inalámbrica "cosas" en redes regionales, nacionales o incluso mundiales. Además, cumple con los requerimientos esenciales de las tecnologías IoT, comunicación bidireccional, seguridad *end-to-end*, movilidad y localización de servicios. Lo que diferencia a una red de este tipo con una red WAN inalámbrica es la baja potencia y el bajo *bit rate* de las redes LPWAN.

Una ventaja de LoRa es la baja complejidad del diseño del receptor debido a la equivalencia de los *offsets* en el tiempo y la frecuencia entre el transmisor y el receptor. La señal de los datos es modulada en una señal *chirp* de manera que su frecuencia puede incrementarse o reducirse con el tiempo. El *chirp rate* define el ancho de banda espectral (BW) de la señal LoRa. Asumiendo un ancho de banda fijo, el *data rate* puede variar dependiendo del SF (Spreading Factor) empleado. El SF puede variar entre 7 y 12 y es el número de bits en bruto por símbolo. Por lo que, el *data rate* R_b, se calcula de la siguiente manera: [13]

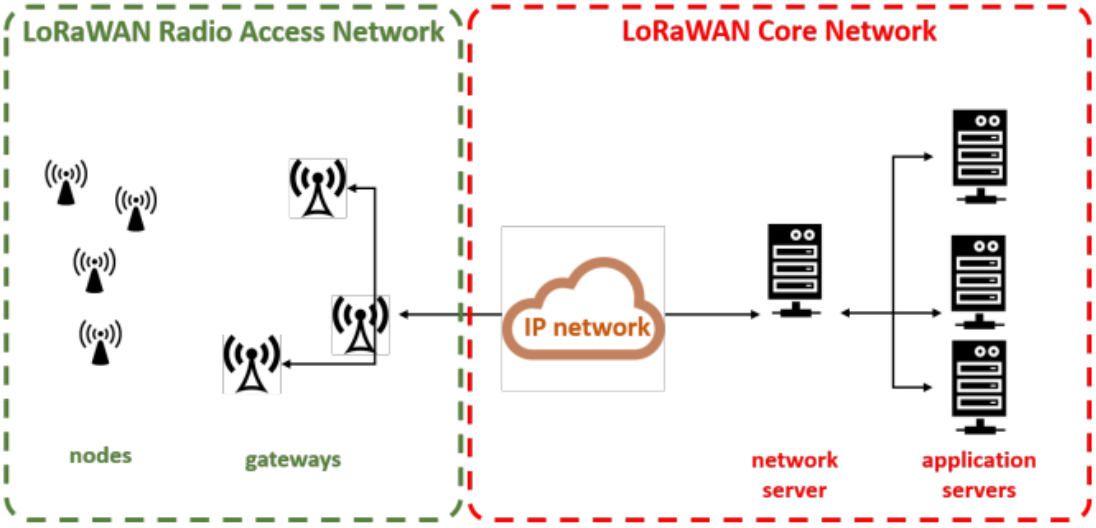
$$R_b = SF \times \left(\frac{BW}{2^{SF}}\right) \times \left(\frac{4}{4+CR}\right), \qquad (4.1)$$

donde el primer término es el SF, el segundo término se refiere a los símbolos por

segundo (R_s), y el tercer término depende del Code Rate (CR). Por lo que, observando la ecuación 4.1 podemos ver cómo conforme el SF incrementa, el data rate se reduce.

El protocolo y la arquitectura de la red tienen una gran influencia en la capacidad, la seguridad y la variedad de aplicaciones que podrán ser usadas por la misma [14]. Una red típica LoRaWAN consistiría en una topología de estrella donde uno o más *gateways* transmiten tráfico de bajada y de subida entre una variedad de dispositivos finales y un servidor central de red (figura 4.1).

Figura 4.1: Arquitectura de red típica LoRaWAN [13].



El servidor central de red gestiona el tráfico entre cada dispositivo final y su aplicación asociada. La comunicación asociada entre dispositivos finales y *gateways* consiste en conexiones de un solo salto LoRa o conexiones FSK. Todas las comunicaciones son bidireccionales.

La comunicación entre dispositivos finales y *gateways* tiene lugar usando diferentes canales y SF. El canal de transmisión es escogido en cada comunicación por el dispositivo final de una manera pseudoaleatoria y el SF es seleccionado teniendo en consideración el rango y la duración del mensaje y la comunicación mediante un esquema Adaptive Data Rate (ADR). Las transmisiones simultáneas que se produzcan en el mismo canal, pero con diferentes SFs nunca colisionarán debido a la ortogonalidad existente entre los SFs. Por lo que se puede usar esta cualidad para mejorar la capacidad de la red. La unión de un canal y un SF se denomina *resource block*, RB. Cuando se habla de máscara de *resource block* se hace referencia al conjunto de pares canal/SF que están disponibles para su uso.

4.1.1 Clases de LoRaWAN

Los dispositivos finales tienen multitud de utilidades y necesitan de diferentes requerimientos. Para optimizar todos los perfiles de dispositivos finales, LoRaWAN utiliza diferentes clases de dispositivos. Estas clases sacrifican latencia en las comunicaciones *downlink* a cambio de más ahorro en la batería.

El ahorro de la batería en los dispositivos LoRaWAN es realmente importante ya que, estos dispositivos son asíncronos y transmiten cuando tienen datos que enviar. Los protocolos de este tipo suelen ser referidos como método ALOHA. En una red que esté sincronizada se tiene que "despertar" a los dispositivos para comprobar los mensajes. Este acto es el principal factor que consume energía en los dispositivos por lo que dependiendo de nuestro sistema nos convendrá tener una u otra clase dependiendo de la energía que estemos dispuestos a usar.

Habiendo explicado el ahorro de la batería en nodos LoRaWAN se va a proceder a explicar las diferentes clases que usa [15]:

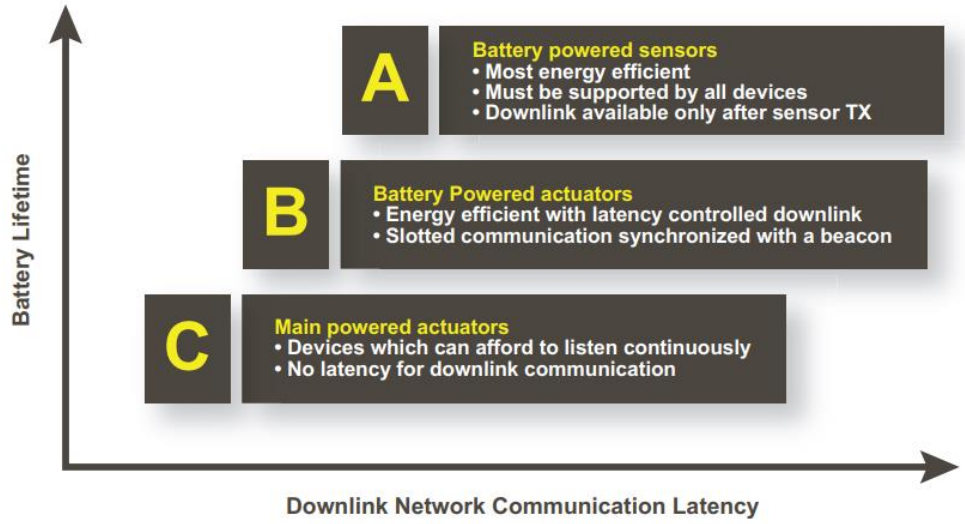
- **Clase A:** Para dispositivos finales bidireccionales. Los dispositivos finales de clase A habilitan comunicaciones bidireccionales por medio de las cuales cada transmisión uplink de cada dispositivo final es seguida de dos ventanas receptoras downlink. La ranura de transmisión programada por el dispositivo final está basada en lo que su propia comunicación necesita con pequeñas variaciones basadas en un fundamento aleatorio de tiempo (un protocolo tipo ALOHA). Esta clase suele ser la más escogida debido a su baja consumición de energía. Está creada para aplicaciones que solo requieren de comunicación downlink desde el servidor justo después de que el dispositivo final haya enviado una transmisión uplink. El resto de las comunicaciones downlink desde el servidor a cualquier otro momento tendrán que esperar hasta la próxima comunicación uplink programada.

- **Clase B:** Para dispositivos finales bidireccionales con ranuras programadas. Esta clase añade a la clase A ventanas de recepción aleatoria. Los dispositivos de la clase B abren ventanas de recepción extra en tiempos planificados. A fin de que el dispositivo abra estas ventanas receptoras en los tiempos programados, reciben una señalización de tiempo sincronizado desde el gateway. Esto permite al servidor saber cuándo el dispositivo final está escuchando.

- **Clase C:** Para dispositivos finales con el máximo de ranuras receptoras. Los dispositivos de la clase C tiene casi continuamente abiertas ventanas receptoras, y sólo se cierran cuando se está transmitiendo.

Para resumir y tener un esquema aclarativo sobre los tipos de clases de LoRaWAN y sus usos se ha insertado la figura 4.2 como ayuda.

Figura 4.2: Esquema clases de LoRaWAN [14].



4.1.2 Seguridad LoRaWAN

Es extremadamente importante para cualquier red LPWAN incorporar la debida seguridad. LoRaWAN utiliza dos capas de seguridad: Una para la red y otra para la aplicación. La seguridad de la red asegura la autenticidad del nodo en la red mientras que la capa de seguridad de la aplicación asegura que la red del operador no tiene acceso a los datos del usuario final de la aplicación. La codificación AES es usada con el intercambio de llaves utilizando el identificador IEEE EUI64.

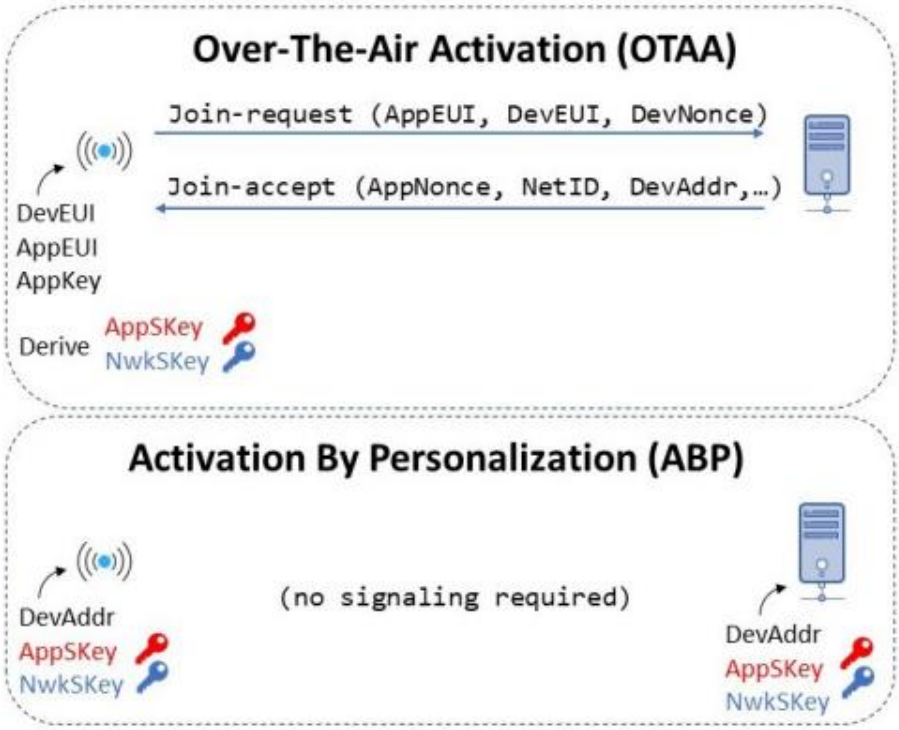
Una parte importante de la seguridad de LoRaWAN es la activación del dispositivo final. Para que el dispositivo pueda enviar y recibir mensajes debe estar activado. Hay dos métodos de activación disponibles [16]:

- **Over-The-Air-Activation (OTAA):** Este método es el más recomendable y seguro a la hora de activar un dispositivo final. Los dispositivos realizan un procedimiento para entrar a la red, durante el que una dirección de dispositivo dinámica es asignada y las *security keys* son negociadas con el dispositivo.

- **Activation By Personalization (ABP):** Requiere programar las direcciones además de las *security keys* del dispositivo. ABP es menos seguro que OTAA y también tiene la desventaja de que los dispositivos no pueden cambiar de proveedor de red sin cambiar las *security keys* manualmente en el dispositivo.

En la figura 4.3 se puede ver claramente las principales diferencias mencionadas.

Figura 4.3: Esquema OTAA y ABP [13].



4.1.3 Capacidad de la red

Para crear una red de gran alcance con arquitectura en estrella como la presentada en la figura 4.1, el *Gateway* debe tener una muy alta capacidad o el potencial de recibir mensajes de una gran cantidad de nodos. Una alta capacidad de red es alcanzada en LoRaWAN usando un *adaptive data rate* (ADR) y utilizando un multi canal, multi-modem transmisor-receptor en el *gateway*, de forma que los mensajes simultáneos en los diferentes canales pueden ser recibidos. Los factores críticos que afectan a la capacidad de red son los números de canales concurrentes, el *data rate*, la longitud del *payload*, y como de frecuente los nodos transmiten.

Teniendo en cuenta que LoRa está basada en tecnología CSS, las señales son prácticamente ortogonales entre ellas cuando se utilizan diferentes *Spreading Factors.* Conforme los SF cambian, el *data rate* efectivo cambia en consecuencia. Los *gateways* toman ventaja de esta propiedad siendo capaces de recibir múltiples *data rates* distintos en el mismo canal al mismo tiempo. Si un nodo tiene un buen enlace y está cerca de un *gateway*, no hay ninguna razón para usar siempre el *data rate* más bajo y usar el espectro disponible más tiempo del necesario. Al elevar el *data rate*, el tiempo que dura la transmisión es más bajo, abriendo un abanico de posibilidades debido al espacio que esto crea para la transmisión de otros nodos.

El ADR también optimiza el ahorro de la batería de un nodo. Para que un ADR funcione, se deben tener enlaces simétricos *uplink y downlink* con suficiente capacidad *downlink*. Estas características hacen que una red LoRaWAN tenga la posibilidad de ser muy escalable y de alta capacidad. Una red sencilla puede ser implementada con una

mínima cantidad de esfuerzo y de infraestructura y, conforme se va necesitando una red mayor, se pueden añadir más *gateways*, multiplicando la capacidad hasta seis u ocho veces.

4.1.4 Limitaciones de la capa MAC de LoRa

LoRaWAN emplea un protocolo ligero MAC basado en P-ALOHA para programar transmisiones *uplink* desde los dispositivos finales. Es un simple protocolo de acceso al medio en el que, todo el tiempo en el que el dispositivo final tiene datos que transmitir, un paquete es enviado sin ninguna coordinación. Por lo que, si dos o más nodos transmiten datos simultáneamente, podrá ocurrir una colisión. Cuanto más alto sea el número de dispositivos finales accediendo a la red, mayor será el número de colisiones. Esto hace que se incremente la tasa de error por paquete lo que resulta en una degradación del desempeño y una reducción de la capacidad de red.

Uno de los problemas que afectan significativamente el rendimiento de una red LoRaWAN es el acceso al medio [17]. Por ello es importante recalcar la importancia de la coordinación en una infraestructura compartida que está ejecutando diferentes aplicaciones con diferentes requisitos usando un acceso al medio de tipo ALOHA [18].

4.2 Bandas ISM

Las bandas de radio ISM son porciones del espectro de radio que están reservadas internacionalmente para propósitos industriales, científicos y médicos, excluyendo aplicaciones en telecomunicaciones [19]. Las emisiones más potentes de los dispositivos que utilizan estas bandas pueden crear interferencias electromagnéticas, por ello, se ha ido incrementando a un ritmo muy rápido el uso de estas bandas para sistemas inalámbricos de baja potencia, como por ejemplo las redes LPWAN y LoRaWAN.

En Europa se definen unas bandas concretas para la red LoRaWAN. Operan en las bandas ISM EU863-869, que definen diez canales de los cuales ocho son *multi data rate* desde 250bps hasta 5.5 kbps, uno para *high data rate* LoRa a 11 kbps y un último canal FSK a 50 kbps. Los ocho canales *multi data rate* son los que nos interesarán para este proyecto y tienen un ancho de banda de 125 Khz cada uno. La máxima potencia permitida por ETSI en Europa es +14 dBM. Para respetar las regulaciones de la ETSI, los dispositivos LoRaWAN están limitados a un ciclo de trabajo menor del 1% [20].

Capítulo 5

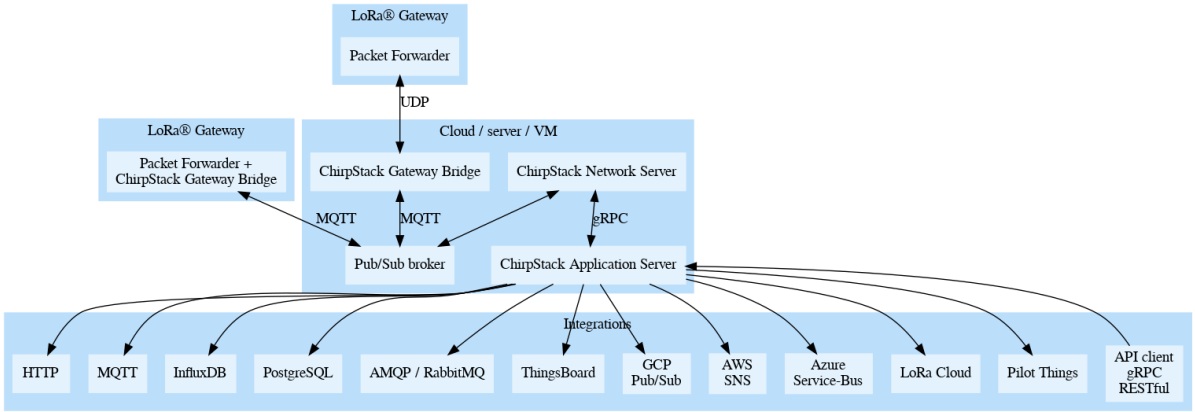
5. Herramientas fundamentales

En este capítulo vamos a hablar sobre la plataforma de Chirpstack y MQTT, comentaremos todas sus características, capacidades y las aplicaciones que nos interesan para este proyecto.

5.1 Arquitectura de Chirpstack

La arquitectura de Chirpstack es una arquitectura modular, es decir, una arquitectura que separa las funcionalidades de cada programa para que sean independientes y que, de esta manera, se puedan separar en módulos intercambiables ya que cada uno de ellos contiene todo lo necesario para realizar la funcionalidad deseada. Gracias a esta arquitectura se pueden integrar a infraestructuras ya existentes sin ningún problema. Una posible implementación de la arquitectura mencionada se muestra en la figura 5.1

Figura 5.1: Ejemplo arquitectura modular Chirpstack [8].



Dentro de los componentes de Chirpstack, los más importantes dentro de su arquitectura son:

- **Dispositivos LoraWAN:** Son aquellos que envían la información al *gateway* LoRa. Pueden ser de multitud de tipos, desde sensores de aire y temperatura hasta sensores de luz, geolocalización, etc… En este TFG estos dispositivos

consisten en las motas utilizadas.

- ***Gateway* LoRa:** Es el que contiene el *software* responsable de recibir y enviar paquetes, este *software* es también llamado *Packet Forwarder*. Suele escuchar en varios canales a la vez y reenvía los datos recibidos de los dispositivos LoRaWAN al servidor de red de Chirpstack.

- ***Gateway Bridge* de Chirpstack:** Se localiza entre el *gateway* y el bróker MQTT. Cambia el formato usado por el *Packet forwarder* al formato usado por los demás elementos de Chirpstack.

- **Servidor de red de Chirpstack:** En realidad este servidor no es más que un servidor de red de LoRaWAN que gestiona el estado de la red. Controla las activaciones de los dispositivos y puede responder a peticiones *join-requests* cuando algún dispositivo quiere entrar a la red. En el caso de que alguna información le llegue doble al servidor, debido a bucles en los *gateways*, lo identifica y lo interpreta como un mensaje que envía al servidor de aplicación de Chirpstack. En el caso de que un servidor de aplicación necesite enviar datos a un dispositivo, el servidor de red será el encargado de mantener estos datos en espera hasta que sea capaz de enviarlos a las GW.

- **Servidor de aplicación de Chirpstack:** En este caso también se trata de un servidor de aplicación de LoRaWAN. Proporciona una interfaz web y APIs para la gestión de usuarios, organizaciones, aplicaciones, *gateways* y dispositivos.

5.1.1 Servidor de red de Chirpstack

Anteriormente se ha mencionado que este servidor red no es más que un servidor de red LoRaWAN, esto hace que las activaciones de los dispositivos también se hagan mediante los dos tipos de activaciones: OTAA y ABP. En el caso de OTAA el servidor de red llamará a la función *Join Server*, que habrá sido configurada previamente en el archivo de configuración, y gestionará la autenticación del dispositivo y la generación de claves. En el caso de ABP el servidor de red puede activar previamente los dispositivos mediante su API, que después serán gestionados de igual manera que con OTAA.

El ADR es un elemento importante en nuestra red ya que nos permite optimizar el espectro y aumentar la capacidad de nuestra red. La activación de este elemento la controla el dispositivo, solo cuando éste envíe un mensaje *uplink* con el campo ADR establecido en “true” el servidor de red ajustará el *data rate* y la potencia de transmisión del dispositivo.

El servidor de red cuenta con diferentes funciones que permiten realizar distintas acciones. Entre ellas destacan *DevStatusReq*, que permite solicitar al dispositivo el estatus de la batería, y *DeviceTimeReq*, que permite la sincronización del reloj interno del dispositivo con la red. Este último es particularmente útil para aplicaciones finales que necesiten una manera de medir el tiempo de forma precisa.

También es posible asignar un perfil a un *gateway*. De esta manera el servidor de red de Chirpstack actualizará automáticamente las configuraciones nuevas que se hayan implementado de manera que siempre esté a la par que la red. Esta actualización se activa siempre que se reciban las estadísticas del *gateway*, dentro de las que aparece la versión actual del GW. Dentro del perfil del GW se pueden configurar a su vez los canales *uplink* además de que, si es posible, se pueden añadir canales extra.

5.1.2 Servidor de aplicación de Chirpstack

El servidor de aplicación de Chirpstack se encarga de la encriptación y desencriptación de los *payloads* de las aplicaciones. También conserva las claves de los dispositivos y gestiona los mensajes *join-accept* en caso de que la activación del dispositivo se de mediante OTAA. Tiene varias maneras de enviar y recibir y los *payloads* de los dispositivos, como MQTT o HTTP. A pesar de estar encargado de la encriptación y desencriptación también es posible ver los contenidos codificados que se reciben para inspeccionar los detalles de estos. Ejemplos de contenidos encriptados y desencriptados se pueden ver en las figuras 5.2 y 5.3. La desencriptación pasa de base64 a hexadecimal.

Este servidor tiene la capacidad de enviar *pings* sucesivos a los distintos *gateways* para así comprobar la cobertura de la red y exponerlo en un mapa descriptivo que se puede comprobar siempre que se quiera en la interfaz web proporcionada.

Figura 5.2: Paquete encriptado

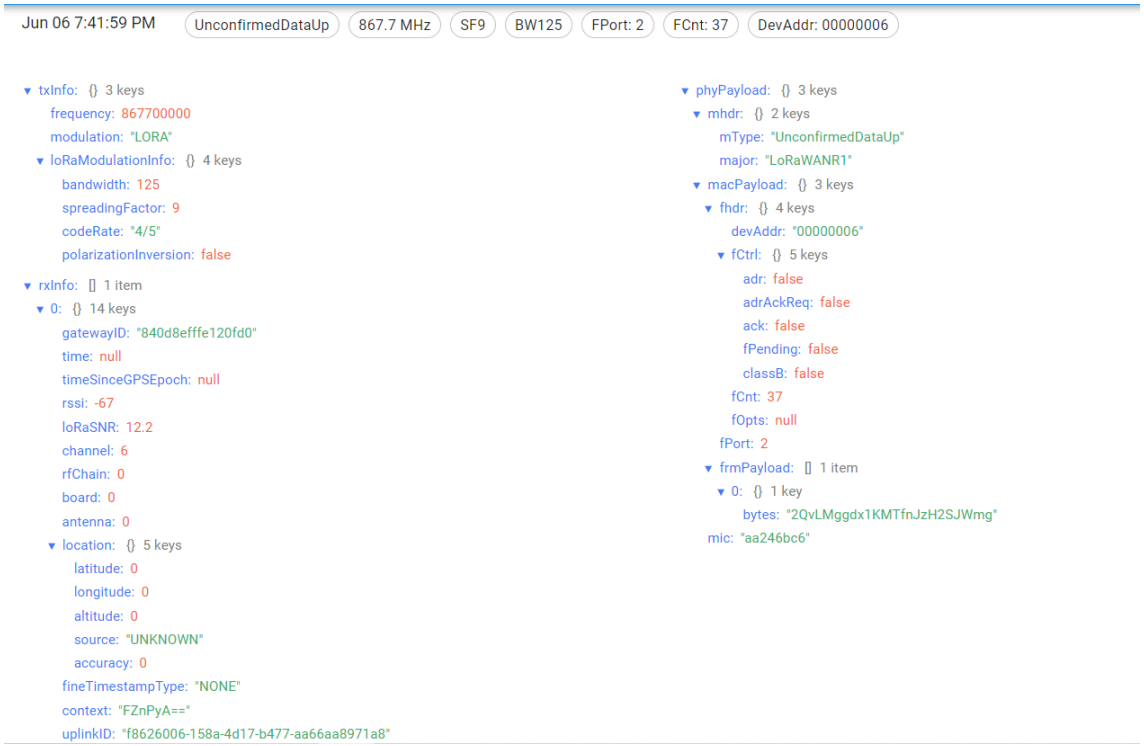


Figura 5.3: Paquete desencriptado.

5.2 MQTT

MQTT es un protocolo de mensajería cliente servidor de publicación/suscripción [21]. Múltiples clientes se conectan a un bróker y se suscriben a tópicos en los que están interesados, también se conectan al bróker para publicar mensajes en esos mismos

tópicos. Es un protocolo abierto y simple, diseñado para ser fácil de usar. Esto lo hace perfecto para usarse en las tecnologías que incumben a las IoT, donde el ancho de banda es bajo y protocolos complejos no tienen cabida. Funciona sobre TCP/IP o sobre cualquier otro que provea conexiones bidireccionales que no tengan pérdidas y los paquetes lleguen ordenados. Algunas de sus características más importantes son:

- El uso de un patrón de mensajería de publicación/suscripción que provee una distribución *one-to-many* y desvinculación de las aplicaciones.

- El transporte de los mensajes no depende del contenido del *payload*.

- Tiene tres tipos de QoS dependiendo de los requisitos demandados:

    1. "Como mucho una vez", es decir, que el mensaje no llegará por duplicado, pero la pérdida de mensajes puede ocurrir si se pierde el mensaje enviado. Esto es útil para dispositivos que actualizan periódicamente su estado y no importa perder una de sus actualizaciones.

    2. "Por lo menos una vez", es decir, que los mensajes están garantizados que lleguen, pero pueden llegar duplicados.

    3. "Exactamente una vez", es decir, que los mensajes están garantizados que lleguen sin haber posibilidad de duplicados. Esto es útil para aplicaciones que no puedan permitirse pérdida ni redundancia de paquetes.

- No sobrecarga la red de tráfico.

- Notifica cuando una ocurre una desconexión no prevista.

5.2.1 Mosquitto

Mosquitto es un bróker de mensajería de código abierto que implementa el protocolo MQTT [22]. Es ligero e ideal para usar en aplicaciones que requieran de poca potencia. Básicamente Mosquitto es un puente que se conecta a otros servidores de mensajería que implementen el protocolo MQTT. Debido a este puente es posible pasar mensajes MQTT desde un origen a un destino. El funcionamiento de Mosquitto se basa en dispositivos conectándose a un servidor de mensajería, suscribiéndose a un tópico y publicando en él. En el caso de este proyecto distribuiremos mensajes mediante JSON.

5.3 PostgreSQL

PostgreSQL es una base de datos de código abierto que usa y extiende el lenguaje SQL combinado con multitud de características que hace que el almacenamiento seguro de los datos más complejos sea posible [23]. En esta base de datos puedes definir tus propios tipos de datos, construir funciones y programar en diferentes lenguajes sin tener

que volver a compilar la propia base de datos. Algunas de sus características más destacadas son:

- Tipos de datos: Entre los que se encuentran primitivos, estructurados, geométricos, JSON, XML…
- Integridad de los datos: *Primary/Foreign keys*, UNIQUE, NOT NULL, etc... que permiten que los datos sean únicos y no estén vacíos.
- Concurrencia y eficiencia: Permite planificador de colas, control de la concurrencia…
- Fiabilidad: Recuperación frente a desastres
- Seguridad: Autenticación, sistema de control de acceso robusto…
- Extensibilidad: Fácilmente customizables y posibilidad de conectar con otras bases de datos SQL
- Internacionalización, búsqueda de texto: Soporte para caracteres de otros países y posibilidad de buscar textos completos.

5.4 JWT

Es un estándar abierto que define una manera de transmitir información entre entidades de manera segura, transmitiéndola como objetos JSON [24]. La información compartida se puede confiar y verificar debido a que está firmada digitalmente. Los JWTs pueden ser firmados mediante un secreto o mediante claves públicas/privadas.

La estructura de un token JWT consiste en tres partes, que son [25]:

- **Cabecera:** Que a su vez consiste en dos partes, el tipo del token, es decir JWT, y el algoritmo para firmarlo. Un ejemplo de la cabecera sería la de la figura 5.5.
- **Payload:** Que contiene las declaraciones del token. Estas declaraciones contienen la información del usuario o la entidad que lo usa e información adicional. Un ejemplo de *payload* sería el de la figura 5.4.
- **Firma:** Para ello se tiene que coger la cabecera y el *payload* codificados, un secreto, el algoritmo especificado en la cabecera y firmarlo. Para crear el secreto se puede hacer mediante el siguiente comando:

  openssl rand -base64 32

  La firma se usa para verificar la integridad del mensaje y si se usa en combinación con una clave privada también se comprueba la identidad del remitente.

El resultado son tres *strings* separados por puntos concatenados entre sí.

Figura 5.5: Cabecera JWT

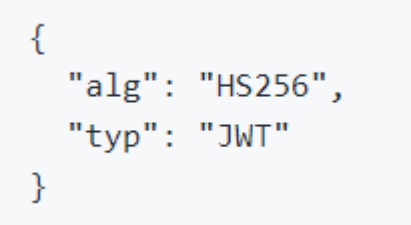


Figura 5.4: Payload JWT

Los JWTs funcionan de manera que cuando un usuario quiere acceder a un recurso protegido, el agente del usuario deberá enviar un JWT, normalmente mediante una cabecera de autenticación. El servidor comprobará si el JWT es válido mirando el JWT contenido en la cabecera de autenticación, si es correcto dejará acceder a la información restringida, o en este caso a una API.

Capítulo 6

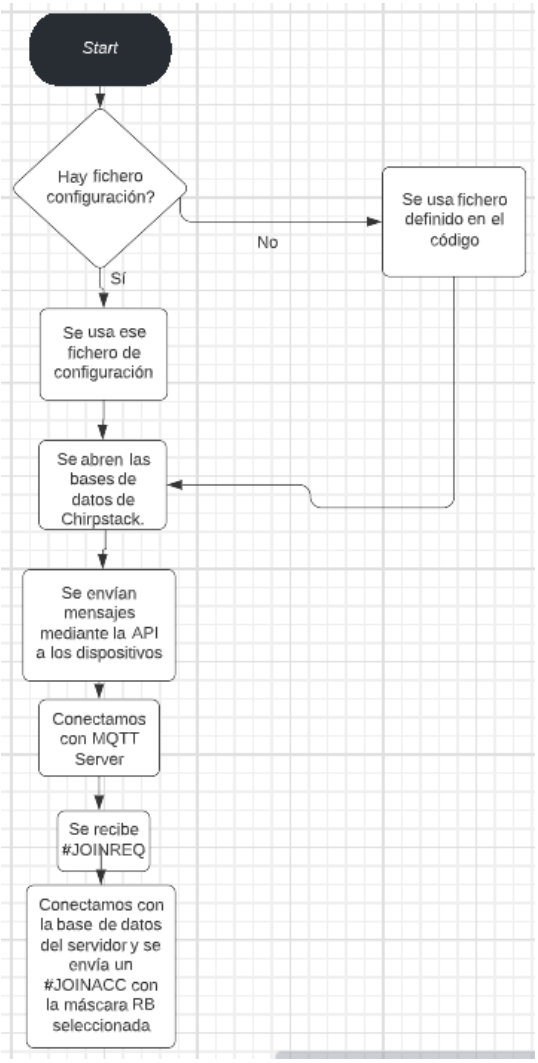
6. Diseño e implementación

En este capítulo se va a explicar el código diseñado para este proyecto, se enseñarán los algoritmos propuestos y cómo se han implementado.

6.1 Servidor

En este subapartado se va a describir las principales funciones del código usado para el servidor y las partes en las que está dividido el archivo donde se encuentra código. Se explicará brevemente el fichero de configuración creado, las bases de datos usadas, la API de Chirpstack y las acciones realizadas por MQTT. Se puede ver un esquema general en el diagrama de flujo de la figura 6.1.

Figura 6.1: Diagrama de flujo del código del servidor



6.1.1 Fichero de configuración

Esta parte del fichero corresponde a la parte de configuración del servidor de red y de aplicación, se crean diversas clases en las que se definen los parámetros para la configuración de la base de datos PSQL, el servidor de aplicación y servidor de red. Por último, se crean funciones para inicializar estos parámetros con los valores requeridos para el funcionamiento correcto del sistema. Sin embargo, esta parte del código solo es útil siempre y cuando no se use un archivo de configuración externo para configurar todo lo previamente explicado.

6.1.2 Bases de datos

En este apartado simplemente se definen dos funciones para abrir y cerrar la base de datos PSQL. En la función de abrir la base de datos se usan los datos proporcionados por el archivo de configuración y para la función de cerrar la base de datos simplemente se cierra la conexión.

6.1.3 Base de datos de Chirpstack

Aquí se usa la función creada en el apartado previo para abrir la base de datos que se debe haber creado previamente con la lista de dispositivos y las características buscadas, en nuestro caso el DEVEUI.

6.1.4 API de Chirpstack

En la API tenemos las funciones para enviar mensajes a un dispositivo en concreto o a todos los dispositivos que tenemos en la lista. Para ello también se ve que hay que decodificar el mensaje en concreto ya que está en base64. El cuerpo del mensaje se concatena en un *string* de un *byte*.

6.1.5 Base de datos del servidor

En este caso se usará una única función de este apartado. Será la correspondiente función que escoja la máscara del *resource block,* de la base de datos en la que tenemos guardada todos los dispositivos y *slices* posibles. Se escogerá el *slice* que corresponda con el dispositivo elegido mediante su DEVEUI.

6.1.6 MQTT

Aquí definimos las acciones que realizará MQTT, publicando y suscribiéndose a tópicos. La primera función consiste precisamente en suscribirse al servidor. También se define la función necesaria para realizar las acciones necesarias para cuando un mensaje *PUBLISH* se recibe del servidor, entre estas acciones se encuentra la codificación y decodificación de los datos y que, si se encuentra un mensaje que contenga un *#JOINREQ* procedente de algún dispositivo, cree un *#JOINACC* en el que enviemos la máscara del *resource block* mediante la función descrita en el apartado anterior, este mensaje es codificado y enviado mediante la función creada en la parte de chirpstack API. Todo esto se puede observar en la figura 6.2.

Figura 6.2: Función MQTT

```
def on_message(client, userdata, msg):
  payload = msg.payload
  payload = payload.decode('utf-8')

  try:
    json_data = json.loads(payload)
    data = json_data['data']
    device = json_data['devEUI'].upper()
    decodedData = base64.b64decode(str(data))
    bEncodedData = base64.b64encode(decodedData)
    encodedData = bEncodedData.decode('ascii')
  except Exception as e:
    print ("Error: " + str(e))

  strDecodedData = decodedData.decode('utf-8')
  print('[MQTT] Message received from device ' + device + ': \"' + data + '\" (decoded \"' + strDecodedData + '\")')

  if int(serverConfigOptions.debug) > 1:
    print('[DEBUG] Encoded data: ' + encodedData)

  if strDecodedData.find("#JOINREQ#") == 0:
    print("[INFO] Received join request from device 0x" + device)
    RBMask = getResourceBlockMaskFromDevice(device))
    joinAcceptMessage = "#JOINACC# " + " " + str(RBMask)

    print("[INFO] Sending join accept to device 0x" + device + ": \"" + joinAcceptMessage + "\"")
    bEncodedDataJoinAccept = base64.b64encode(joinAcceptMessage.encode())
    encodedDataJoinAccept = bEncodedDataJoinAccept.decode('ascii')
    sendMessageLoRaWAN(device, encodedDataJoinAccept)
```

6.2 Dispositivo

En este subapartado se va a describir las principales funciones del código usado para el dispositivo y el algoritmo utilizado para calcular el SF y el canal según un SNR o RSSI.

En primer lugar, se tendrá que escoger el método de activación del dispositivo que queremos escoger, es decir ABP u OTAA. Dependiendo de cuál se escoja se tendrán que hacer algunos pasos previos, en este caso al haber escogido ABP se ha tenido que activar previamente el dispositivo y configurar en la interfaz web de chirpstack los elementos necesarios para su funcionamiento, los cuáles se verán en el anexo I. Seguidamente aparecen las direcciones y claves del dispositivo, necesarias para la comunicación con el servidor y autenticación. Con la función *showBoard()* se permite ver la información del dispositivo, su DEVEUI, el nombre de la placa y la Wi-Fi MAC. También se inicializa LoRaWAN según un tiempo aleatorio en donde se autentica mediante ABP usando los parámetros que se han descrito anteriormente.

6.2.1 ReceiveJoinAccept()

Figura 6.3: Enviar *request.*

```
def receiveJoinAccept():
  # Waiting for JoinAccept message
  JoinAcceptReceived = False
  # JoinRequest has to be transmitted
  timeToRetransmitJoinReq = True
  while not JoinAcceptReceived:

    if (timeToRetransmitJoinReq):
      s.setblocking(True)
      s.send("#JOINREQ#")
      print("[INFO] #JOINREQ# sent!")
      s.setblocking(False)
      timeToRetransmitJoinReq = False
      timeJoinReq1 = utime.ticks_ms()
```

Figura 6.4: Recibir ACK

```
if strDecodedData.find("#JOINACC#") == 0:
  print ("[INFO] #JOINACC# received")
  JoinAcceptReceived = True
  strDecodedDataSplit = strDecodedData.split()
  rb_mask = int(strDecodedDataSplit[1])
  rb_mask = bin(rb_mask)
  length = len(rb_mask)
  rb_mask = invertir_cadena_manual(rb_mask)
```

Figura 6.5: Obtención de la máscara, selección del *slice* y cálculo del canal y SF.

```
loraStats = lora.stats()
print("RSSI: " + str(loraStats[1]))
print("SNR: " + str(loraStats[2]))
j = 0
pos = [None] * 48
for i in range(2,length):
  if (rb_mask[i] == "1"):
      pos[j] = i
      j = j + 1
 pos_rb = pos[random.randint(0,(j-1))]
# A partir de aquí se decide el canal y el SF de ma
#rssi_vec = [-123, -126, -129, -132, -134'5, -137]
bRSSI = False
repeat = False
while (repeat == False):
    pos_rb = pos[uos.urandom(1)[0] % (j-1)]
    print("RB: " + str(pos_rb))
    sf = ((abs(pos_rb - 1)) % 6) + 7
    print("SF: " + str(sf))
    ch = (int((abs(pos_rb - 1)) / 6)) - 1
    print("CH: " + str(ch))
    selectedfq = frequencyForChannel(ch)
    selectedDR = convertSFtoDR(sf)
```

Esta función es la de más interés para este proyecto pues es donde se asignan los *slices* a los dispositivos y se aplican de manera que se dividen los recursos radio en *resource*

blocks. En primer lugar, el programa envía una solicitud *#JOINREQ* (Figura 6.1) hasta que el servidor responda con un *#JOINACC*. Mientras que no llegue, el programa seguirá enviando solicitudes *#JOINREQ* de forma indefinida. Una vez que llegue el ACK por parte del servidor se decodificará el mensaje que contiene la máscara del RB y se guardará en un vector, habiendo pasado previamente de entero a binario el número proveniente del *resource block*, esto se puede ver en la figura 6.2.

A continuación, viene el algoritmo propuesto para la elección de uno de los slices posibles según la máscara del RB, figura 6.3.

Algoritmo:

Como ha sido mencionado en la sección 4.2, en Europa hay ocho canales disponibles de 125 kHz por los que son posibles transmitir con LoRa. Cada uno de estos canales esta a su vez dividido en 6 *Spreading Factors* que se encuentran desde el SF7, que es el menos robusto, pero el más rápido, hasta el 12, el cual es el más robusto, pero menos rápido. Esto hace que en total haya 48 *resource blocks* disponibles para escoger, y en los que podemos dividir la red entre diferentes proveedores. Además, debido a la propiedad de la ortogonalidad de estos SF, aunque haya dos dispositivos transmitiendo en el mismo canal, siempre y cuando no tengan el mismo SF, no colisionarán entre ellos, lo que hace que pueda haber 48 dispositivos transmitiendo simultáneamente.

A la hora de realizar la función para escoger un *resource block* en concreto se ha tenido en cuenta el RSSI del dispositivo, es decir el indicador de fuerza de la señal recibida, y también el SNR, es decir la relación señal-ruido. Dentro de la función se puede escoger si se quiere medir según el SNR o el RSSI. Si el dispositivo tiene un RSSI muy pequeño se coja un SF más robusto, como por ejemplo el SF11 o el SF12, y si tiene gran fuerza la señal recibida que se escoja un SF que sea más rápido, como los SF7 y SF8. Los valores del RSSI que se han cogido para escoger un SF u otro se presentan en la tabla 6.1. Lo mismo pasa con el SNR según la tabla 6.2:

Tabla 6.1: Impacto de los SF en el RSSI [26].

Spreading Factor	RSSI fijada a 125 kHz de ancho de banda
SF7	-123 dbm
SF8	-126 dbm
SF9	-129 dbm
SF10	-132 dbm
SF11	-134.5 dbm
SF12	-137 dbm

Tabla 6.2: Impacto de los SF en el SNR [27].

Spreading Factor	SNR
SF7	-7.5 dB
SF8	-10 dB
SF9	-12.5 dB

SF10	-15 dB
SF11	-17.5 dB
SF12	-20 dB

Según la tabla 6.1 si un dispositivo enviase un *#JOINREQ* y su RSSI fuese -127 dbm, se podrían usar *spreading factors* desde el 8 hasta el 12 para suplir las necesidades de robustez para esa fuerza de la señal. En el caso del SNR, si fuese -11 solo se podrían usar SF de 8 hacia arriba.

Una vez escogido el slice a usar según el RSSI o el SNR hay que pasar del "1" escogido de la máscara del RB al canal y SF correspondientes de ese "1".

La máscara del RB será un numero de 6 bytes, una vez llegue al dispositivo se pasará de representación decimal a binario, con lo cual nos quedará una línea de hasta 48 unos y ceros. Un ejemplo de este número sería el siguiente:

$$\text{RB\_MASK} = 24685436 \xrightarrow{\text{Dec-Bin}} 1\ 0111\ 1000\ 1010\ 1011\ 0111\ 1100$$

Al tener el número ya convertido invertimos el vector para que las posiciones de los "1" concuerden con las posiciones correctas del vector. Seguidamente tenemos que guardar todas las posiciones de los "1" en un vector para poder escoger aleatoriamente uno de los posibles RB entre ellos. El vector resultante quedaría de la siguiente manera:

RB_Posicion = [2,3,4,5,6,8,9,11,13,15,19,20,21,22,24]

Con el vector ya creado podemos elegir de manera aleatoria una de esas posiciones y calcular el SF y el canal mediante la siguiente fórmula:

$$SF = \left((|RB\_Posicion_i - 1|)\ \%\ 6\right) + 7$$

$$Canal = \left((|RB\_Posicion_i - 1|)\ \div 6\right) - 1$$

Con lo que, si se escoge la posición 21, el SF sería 9 y el canal sería 2 ya que solo se coge la parte entera del número que resulta.

Por último se establecen los parámetros mediante la función *setTransmissionParametres()*.

6.3 Archivo de configuración y Scripts

Aquí se hablará del fichero usado junto al código del servidor para configurar el dispositivo, los scripts usados para hacer más cómodo el uso de la base de datos Postgresql y por último hablaremos del token JWT necesario para la autenticación.

6.3.1 Archivo de configuración

La arquitectura del archivo de configuración es la que se puede observar en la figura 6.6

Figura 6.6: Archivo de configuración

```
[APP-SERVER]
psqlserver = 127.0.0.1
psqluser = chirpstack_as
psqlpassword = chirpstack_as
psqlport = 54321
psqldb = chirpstack_as
appserver = 127.0.0.1
appserveruser = admin
appserverpassword = admin
appserverapiport = 8080
appserverjwttoken = eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOiJhcyIsImV4cCI6MTY1NDk3
appserverappid = 1
fport = 1
appservermqttserver = 127.0.0.1
appservermqttport = 1883
appservermqttuser = MQTTUSER
appservermqttpassword = password

[SERVER]
psqlserver = 127.0.0.1
psqluser = minichannelsuser
psqlpassword = dbpassword
psqlport = 5432
psqldb = minichannelsdb
expirationtimeinminutes = 60
caraenabled = 1
debug = 1


```

Como se puede ver hay dos partes en el archivo, la parte del servidor de aplicación y la parte del servidor de red. En la parte del servidor de aplicación tenemos la dirección del servidor PSQL, MQTT y del propio servidor de aplicación que será la dirección *loopback* ya que se encuentra en el propio host. También se tiene las contraseñas y usuarios necesarios para entrar a los perfiles de la base de datos, de MQTT y del servidor. Por último, tendremos el token JWT. Para la parte del servidor de red tendremos prácticamente lo mismo con la diferencia de que el perfil de la base de datos es distinto y el puerto es otro.

6.3.2 Scripts

Los scripts se han escrito en *shell*, *shell* es un lenguaje intérprete de comandos que es usado por *sh.* Los scripts han sido escritos de esta manera por comodidad y para poder automatizar las operaciones realizadas sin tener que repetir los comandos continuamente.

Los scripts que se han creado han sido los siguientes:

- **slice.sh:** Crea una nueva base de datos llamada *minichannelsdb* con usuario *minichannelsuser*, a su vez crea dos tablas dentro de esta base de datos, la tabla *slices*, que contiene un id para contar cada slice introducido y la máscara del RB asociada, y la tabla *devices_to_slice*, que contiene el DEVEUI del dispositivo y el id del slice asignado al mismo, procedente de la anterior tabla. Además previamente a la creación de las tablas también borramos cualquier base de datos que previamente estuviese creada con el nombre *minichannelsdb.*

- **insertar_into_*:** Un script para insertar valores en las dos tablas que hemos creado.

- **show_*:** Enseña los valores de las tablas *slices* y *devices_to_slice*.

Capítulo 7.

7. Resultados y simulaciones

En este capítulo comprobaremos que todo lo diseñado funciona como estaba planificado, se realizarán distintas pruebas para alcanzar este cometido y que tenga más validez.

7.1 Conexión y comunicación de dispositivos

En primer lugar, vamos a ejecutar de manera corriente nuestro proyecto para ver si se realiza la conexión con el servidor, funciona la comunicación entre dispositivos y se adjudica un RB a la mota correspondiente. Para ello ejecutaremos los comandos dispuestos en el anexo B. Una vez que hayamos seguido todos los pasos dispuestos en el manual, tendremos nuestro proyecto listo para ejecutar.

El siguiente paso sería abrir Atom y ver las comunicaciones que se van produciendo entre el GW y la mota. Por último, ver el resultado de éstas en el servidor de red. El resultado de los mensajes intercambiados debería ser como los de la figura 7.1 en la mota y 7.2 en el GW:

Figura 7.1: Comunicaciones en la mota

```
[INFO] Detected board: FiPy
[INFO] Wi-Fi MAC:      807D3AC3069C
[INFO] LORAWAN DevEUI: 70B3D549986901D5
[INFO] Random time for joining = 5.000000
[INFO] Fixed time between LoRaWAN frames = 10.000000
[INFO] Ramdom time between LoRaWAN frames = 10.000000
[INFO] Duration of CARA period = 60.000000
[INFO] Avoid border effect = 0
[INFO] Border effect guard time = 1.000000
[INFO] Nodes with version 2 counted = 1
[INFO] Time waiting before joining = 4.7
[INFO] Not joined yet...
[INFO] --- Joined Sucessfully ---
[INFO] #JOINREQ# sent!
[INFO] #JOINACC# received
[INFO] CARA enabled
RSSI: -29
SNR: 7.0
RB: 32
SF: 8
CH: 4
[INFO] Waiting for next transmission (t=253.813)...
[INFO] Message sent at 00:04:18.816957 on 0 Hz with DR 4 (air time 0.093 s)
[INFO] Waiting for next transmission (t=263.907)...
```

Figura 7.2: Comunicaciones en el GW

```
>>> [2420700745] LORAPF_INFO_:[main] report
##### 2022-06-23 11:18:29 GMT #####
### [UPSTREAM] ###
# RF packets received by concentrator: 3
# CRC_OK: 66.67%, CRC_FAIL: 33.33%, NO_CRC: 0.00%
# RF packets forwarded: 2 (44 bytes)
# PUSH_DATA datagrams sent: 3 (523 bytes)
# PUSH_DATA acknowledged: 100.00%
### [DOWNSTREAM] ###
# PULL_DATA sent: 6 (100.00% acknowledged)
# PULL_RESP(onse) datagrams received: 1 (220 bytes)
# RF packets sent to concentrator: 1 (39 bytes)
# TX errors: 0
# TX rejected (collision packet): 0.00% (req:1, rej:0)
# TX rejected (collision beacon): 0.00% (req:1, rej:0)
# TX rejected (too late): 0.00% (req:1, rej:0)
# TX rejected (too early): 0.00% (req:1, rej:0)
### [JIT] ###
[jit] queue is empty
### [GPS] ###
# GPS sync is disabled
##### END #####
[2420705714] LORAPF_INFO_:[up  ] received pkt from mote: 00000006 (fcnt=7/7), RSSI -52.0
[2420706052] LORAPF_INFO_:jitqueue: Current concentrator tv_sec=68 time_us=68018380, pkt_type=0, p
 = 0/0 s
[2420721268] LORAPF_INFO_:[up  ] received pkt from mote: 00000006 (fcnt=0/0), RSSI -57.0
[2420721605] LORAPF_INFO_:jitqueue: Current concentrator tv_sec=83 time_us=83571596, pkt_type=0, p
 = 0/0 s
[2420730803] LORAPF_INFO_:[main] report
##### 2022-06-23 11:18:59 GMT #####
### [UPSTREAM] ###
# RF packets received by concentrator: 3
```

Como se puede observar en la figura 7.1 la mota envía un *#JoinReq* que es contestado inmediatamente con un *#JoinAcc.* A continuación, se puede ver como salen las diferentes estadísticas, el RSSI que es -29, el SNR que es 7, el RB, el SF y el canal. En la figura 7.2 se ve como se reciben los datos y toda la información relacionada con el envío de paquetes.

Una vez que se ha visto que la comunicación se ha producido correctamente se va a comprobar que se haya adjudicado correctamente mediante la comprobación de las tramas en el servidor de red. Para ello se tiene que ir a *Applications → chirpstack-app (nombre de nuestra aplicación) → Devices → Device Data.* Y se comprobará el SF y el canal de la trama correspondiente. En este caso las tramas serían las de la figura 7.3:

Figura 7.3: Comprobación SF y canal

Como se puede ver, el SF y el canal cambian después del *txack*, también se puede observar que la mota está transmitiendo en el SF y el canal asignado, SF=8 y canal=4, durante un tiempo indefinido hasta que se desconecte. Esto significa que el algoritmo ha funcionado y que se ejecuta con éxito.

7.2 Algoritmo según SNR

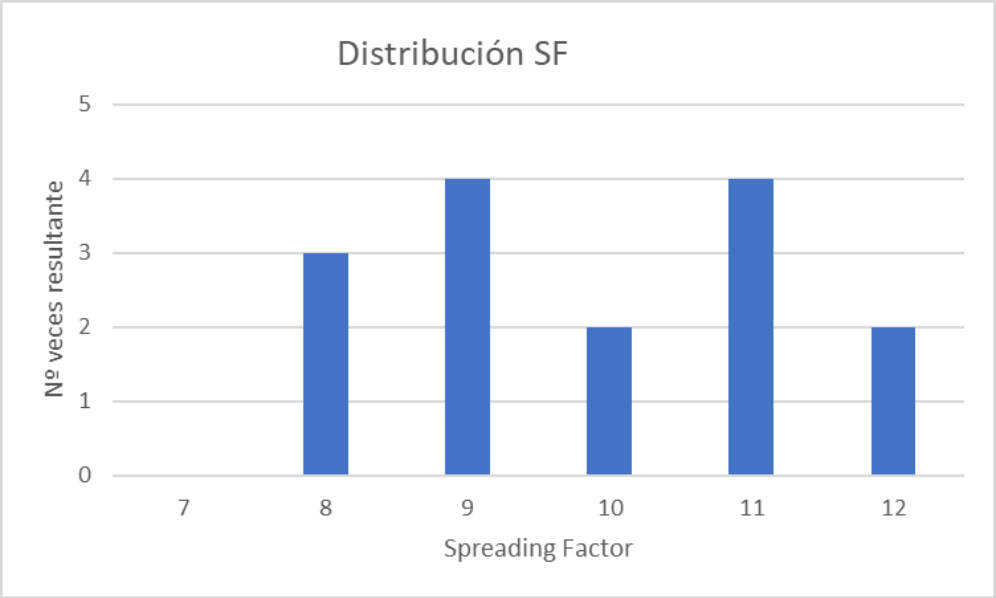
En este apartado se va a demostrar que el algoritmo, explicado en la sección 6.2.1, para que se asignen *resource blocks* dependiendo de la relación señal ruido funciona. Para ello se han hecho pruebas consistentes del GW en un 6º piso y la mota siendo llevada por distintos sitios de la ciudad en un rango menor de 1 km alrededor del GW. Durante las pruebas no se ha conseguido tener una cobertura menos a -132 dBm, por lo que hemos pasado el algoritmo para que funcione con SNR en vez de RSSI. Hemos conseguido valores de SNR alrededor de -11 dB, esto significa que deberíamos tener un SF de 8 o mayor según la tabla 6.2. Esto se demuestra en las figuras 7.4 y 7.5, en la figura 7.5 tenemos un SNR de 13’5 por lo que el SF tiene que ser mayor o igual que 9:

Figura 7.4: Demostración algoritmo SNR 1

Figura 7.5: Demostración algoritmo SNR 2

Como se puede observar, en la figura 7.4 se ha cumplido con el algoritmo descrito en la sección 6.2.1 ya que el SF no ha sido 7 u 8, según la tabla 6.2 solo podría tomar los valores de 9 a 12. En el caso de la figura 7.5, también se comprueba que la prueba ha sido exitosa debido a que el SF vuelve a ser 9. Esto vuelve a ser correcto debido a que según la tabla 6.2, al tener un SNR de -11 dB solo se podría tener SFs mayores que 8. Una vez visto que funciona según corresponde, se va a comprobar la uniformidad de los SF según un SNR concreto. Se han realizado alrededor de 15 simulaciones con un SNR de entre -10 y -12’5 dB, en la figura 7.6 se va a mostrar los SF resultantes de las diferentes simulaciones y se va a observar una distribución de los SF obtenidos en cada prueba.

Figura 7.6: Distribución SF



En conclusión, para un número reducido de pruebas se ha visto que la distribución de los *spreading factors* es similar y que en ningún momento para un SNR menor que -10 se ha asignado al dispositivo un SF de 7.

Capítulo 8.

8. Conclusiones y trabajos futuros

En este último capítulo se hablará sobre todas las conclusiones sacadas del trabajo, logros conseguidos, proyectos futuros que puedan surgir a raíz de este y el interés de estos.

8.1 Conclusiones

En este proyecto se ha diseñado un algoritmo de particionado de la red para una red LoRaWAN, se ha realizado para que distintos usuarios o empresas puedan usarla en sus comunicaciones sin que estas colisionen entre sí debido a la propiedad de ortogonalidad de los canales junto a sus SF. Además de ello, el algoritmo se ha propuesto para que asigne los canales y SF según el RSSI o SNR de los nodos LoRaWAN y que así no se elijan de manera arbitraria.

En primer lugar, se estudió la tecnología LoRa y LoRaWAN y se observaron ejemplos de otras tecnologías en las que el particionado de red se había conseguido. A raíz de esto se vio la necesidad de realizar este proyecto para LoRaWAN, puesto que solventa problemas que las demás tecnologías no pueden lidiar tan fácilmente, como que haya un operador de infraestructura que alquile sus recursos a otros operadores de servicios, cada uno con su trozo. Cada trozo tendrá los recursos necesarios (según se negocie con el operador) y será independiente de las demás particiones. El siguiente paso fue la instalación, montaje y configuración de una red LoRaWAN sobre la que montar nuestro algoritmo. Una vez teniendo todo configurado todo el sistema se pasó al estudio del algoritmo a realizar y a la visualización de cómo funciona la red LoRaWAN ya montada. Por último, se realizó el proyecto pensado y se añadió la modificación para que eligiese los RB según el RSSI o SNR de cada dispositivo.

Teniendo todo el trabajo hecho se pasó a la simulación de pruebas para comprobar que todo estaba montado correctamente y que funcionaba según el comportamiento deseado. Se observó que las pruebas según el SNR funcionaban y que la distribución de SF es uniforme. Viendo el resultado de todas las simulaciones podemos ver que la red es operativa y funciona de la manera pensada.

8.2 Trabajos futuros

A pesar de tener un trabajo funcional, no es un proyecto perfecto y hay cabida para ciertas mejoras o distintas soluciones que se pueden implementar dependiendo del funcionamiento requerido:

- Implementar un cambio en el código para aumentar el número de características por el que escoger un RB, aparte de RSSI y SNR.

- Análisis del rendimiento y del coste energético del sistema. Probar con las diferentes clases de LoRaWAN según las necesidades del consumidor.

- Implementar una manera por la que el usuario pueda escoger el *slice* según la QoS que requiera.

I. Anexo manual de instalación.

En este anexo explicaremos detalladamente todos los pasos necesarios para instalar y configurar todas las herramientas necesarias para realizar el proyecto planteado.

I.1 Configuración de la Raspberry pi

En primer lugar, hay que conectar la Raspberry Pi al router y asignarle una IP fija. Esto no es un paso obligatorio, pero ayuda a que no haya problemas en los pasos posteriores. Una vez hecho esto hay que instalar el OS en la Raspberry Pi. Para ello hay que entrar en la página oficial de Raspberry e instalar la versión Debian Bullseye 11. Seguidamente pasaremos a instalar *Docker engine* en la Raspberry Pi, para ello simplemente ejecutaremos el script de conveniencia que nos proporciona la web oficial. Los comandos que ejecutar son los siguientes:

$ curl -fsSL https://test.docker.com -o test-docker.sh
$ sudo sh test-docker.sh

Si queremos ver qué se está ejecutando durante el funcionamiento del script podemos añadir la opción *DRY_RUN=1* al principio del segundo comando.

I.2 Herramientas y programas necesarios

Para poder programar dispositivos LoRaWAN y realizar programas con esta tecnología es necesario disponer de diversos programas que nos habilitarán esta tarea. Lo primero que se deberá descargar será Atom, el editor de código. Debido a un problema de versiones se deberá descargar la versión 1.60, es decir, la última versión disponible. Este programa se descargará desde la web oficial de Atom [7]. El siguiente paso será descargarse el entorno de ejecución para JavaScript Node.js. Esta herramienta se descargará desde su página oficial [28] en la ventana de versiones anteriores ya que se necesitará la versión 14.16.1. En el caso de que haya versiones de node.js más recientes habrá que desinstalarlas primero. Por último, hace falta pymakr, un plugin desarrollado por Pycom para facilitar la programación y compilación del código cuando se usa micropython. Este plugin no se debe descargar desde el instalador de paquetes de Atom puesto que éste instalará la última versión, que no es compatible con alguna de las herramientas que se usan y dará problemas. Se tendrá que descargar desde la consola mediante el siguiente comando.

apm install pymakr@2.1.13

Apm es un comando procedente de la instalación de Atom, pero se puede comprobar si está instalado mediante el comando *apm help install* que debería dar detalles sobre el mismo. Como se puede ver en el comando de instalación de pymakr, la versión que deberá ser descargada es la 2.1.13 por problemas de compatibilidad. Una vez realizados todos estos pasos se tendrá configurado Atom al completo.

La descarga de Python también será necesaria. Desde la página oficial de Python [29] se puede descargar la última versión que es la recomendada para este proyecto. Por último, hace falta descargar *Visual Studio Code* con los accesorios de *desktop development* y soporte a Python y *node-gyp.*

I.3 Configuración Pygate

El primer paso será actualizar la versión del *firmware* del *gateway* al 1.20.2.r6. Esto se hará con la herramienta Firmware Upgrade Tool [30]. Hay diferentes tipos de *firmwares,* nosotros escogeremos el *pybytes* o el *pygate,* cualquiera de los dos es correcto. Una vez teniendo la versión correcta nos tendremos que descargar de GitHub el main.py y el global_config.json [31]. Lo único que tendremos que cambiar de estos archivos son la id y clave de nuestra Wi-Fi en el main.py y la EUI del *gateway* y la dirección IP del servidor de red en el global_conf.json. En el caso del main.py lo que hace es crear un enlace con el *gateway* y ejecutar el reenvío de paquetes de LoRa. Podremos ver que el *gateway* está funcionando cuando se encienda el LED verde. Para este punto las comunicaciones entre nodos LoRa llegarán al *gateway* y recibirán tanto mensajes *downlink* como *uplink* mediante el mismo.

I.4 Configuración Postgresql

Para la configuración de Postgresql simplemente tendremos que ejecutar el script slice.sh, el cual creará la base de datos y las tablas necesarias, después introduciremos los valores de las motas y de sus respectivos identificadores de RB mediante el script *insertar_into_devices_to_slice* y por último añadiremos la máscara de RB y el id de la máscara mediante el script *insertar_into_slices.*

I.5 Configuración Chirpstack

En primer lugar, hay que copiar el repositorio que contiene un punto de inicio en el que basarnos para crear nuestro servidor. Para ello hay que ejecutar el siguiente comando:

git clone https://github.com/brocaar/chirpstack-docker.git

Ya que no queremos que todo el mundo pueda ver nuestros mensajes MQTT tendremos que añadir un usuario y contraseña, para ello abriremos el archivo *docker-compose.yml* y añadiremos la siguiente línea al final de la parte de la configuración de

mosquitto. Usaremos los siguientes comandos

cd chirpstack-docker
sudo nano docker-compose.yml
./configuration/eclipse-mosquitto/mqtt_passwd:/mosquitto/config/mqtt_passwd

Este último comando es la línea que tenemos que añadir al final de la parte de configuración de *mosquitto,* dentro de *volumes.* El siguiente paso será instalar *mosquitto* y desactivarlo. Esto se realizará mediante las próximas líneas:

sudo apt-get install mosquitto
sudo systemctl stop mosquitto
sudo systemctl disable mosquitto

Ahora generaremos el fichero donde vamos a guardar nuestro usuario y contraseña:

mosquitto_passwd -c configuration/eclipse-mosquitto/mqtt_passwd

E introducimos el usuario y contraseña que en este caso será *MQTTUSER* y *password.* Por último, tenemos que incluir este archivo en el fichero de configuración de *mosquitto.* Para ello abrimos el archivo *mosquitto.conf,* deshabilitamos el usuario anónimo y añadimos el *path* hacia nuestro fichero recién creado:

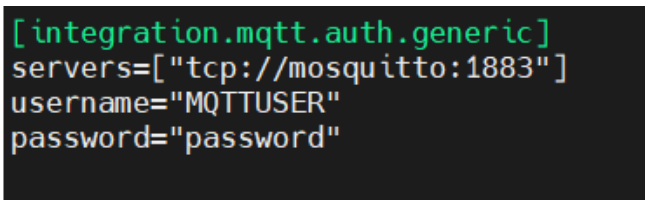
sudo nano configuration/eclipse-mosquitto/mosquitto.conf
allow_anonymous false
password_file /mosquitto/config/mqtt_passwd

Ahora que ya hemos configurado esta parte toca cambiarla también en el *gateway-bridge.* En primer lugar modificaremos el archivo *chirpstack-gateway-bridge.toml:*

sudo nano configuration/chirpstack-gateway-bridge/chirpstack-gateway-bridge

Cambiamos el archivo modificando el usuario y la contraseña hasta que quede como en la figura I.1:

Figura I.1: Configuración del *gateway-bridge*



Tendremos que hacer lo mismo para el fichero de configuración del servidor de red, ejemplo en la figura I.2:

sudo nano configuration/chirpstack-network-server/chirpstack-network-server.toml

Figura I.2: Configuración del servidor de red para *mosquitto*

```
[network_server.gateway.backend.mqtt]
server="tcp://mosquitto:1883"
username="MQTTUSER"
password="password"
```

Por último, tendremos que hacer lo siguiente para el servidor de aplicación, ejemplo en la figura I.3:

sudo nano configuration/chirpstack-application-server/chirpstack-application-server.toml

Figura I.3: Configuración del servidor de aplicación para *mosquitto*

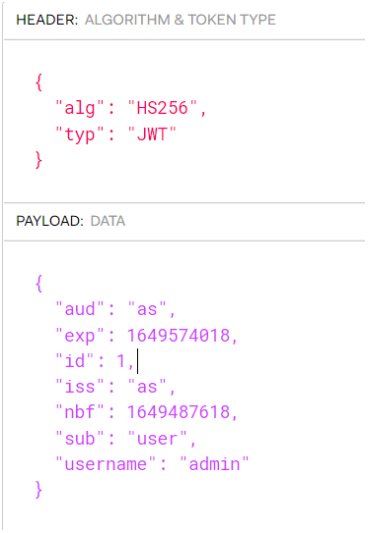
```
[application_server.integration.mqtt]
server="tcp://mosquitto:1883"
username="MQTTUSER"
password="password"
```

El siguiente paso será para poder usar la REST API de Chirpstack, lo que hará falta será generar un JWT token. Primero habrá que generar un secreto codificado en base64, lo crearemos mediante el comando:

openssl rand -base64 32

Y lo incluiremos en la parte *jwt_secret* del archivo de configuración del servidor aplicación, *chirpstack-application-server.toml*. Después generaremos un JWT token en la página oficial [24]. La cabecera y el *payload* del token serán los de la figura I.4, en la parte de *verify signature* introduciremos el secreto generado:

Figura I.4: Cabecera y *payload* para el JWT token.



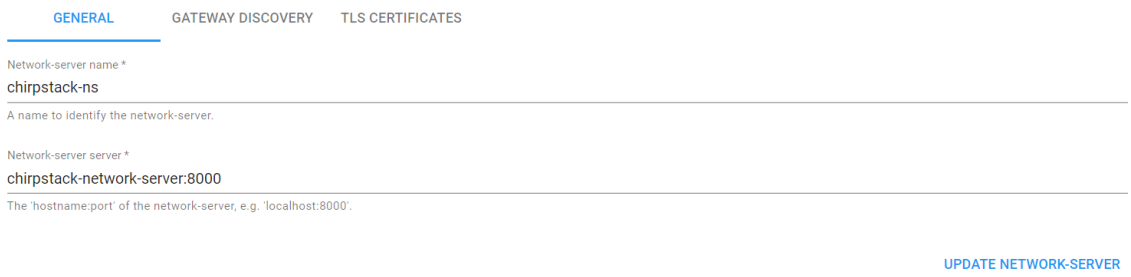
En el caso de que este token no funcionase hay otro método para crear el token. Habría que conectarse a la API de chirpstack, *http://<IP address>:8080/api*, bajamos hasta el apartado *GET /api/internal/login* e introducimos *admin* en el campo del *email* y en el apartado de la contraseña introducimos la contraseña elegida.

Teniendo todo configurado podemos encender el servidor con el siguiente comando.

sudo Docker-compose up -d

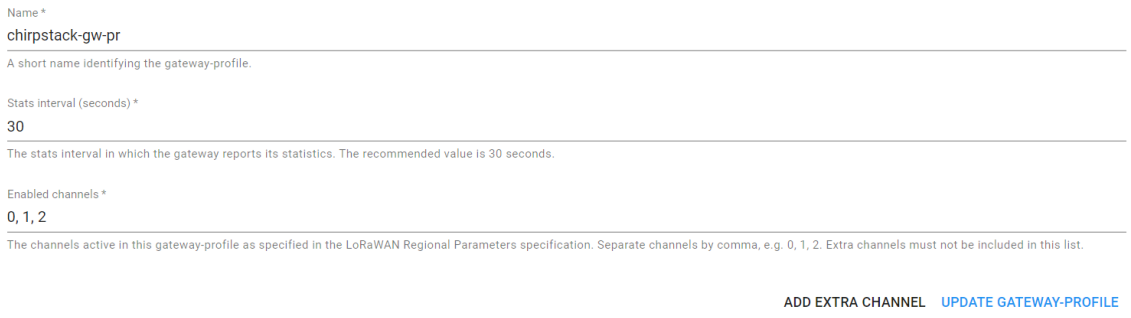
Entramos en la plataforma de Chiprstack en *http://<IP address>:8080* con usuario *admin* y contraseña *admin.* Ahora configuraremos la plataforma para que el *gateway* y los nodos puedan conectarse. En primer lugar, añadiremos un servidor de red, entramos en *Network servers → Add*, y configuramos hasta que quede como en la I.5.

Figura I.5: Servidor de red Chirpstack



El siguiente paso será crear un *gateway profile*, entramos en *gateways profile → Create*, parámetros en figura I.6.

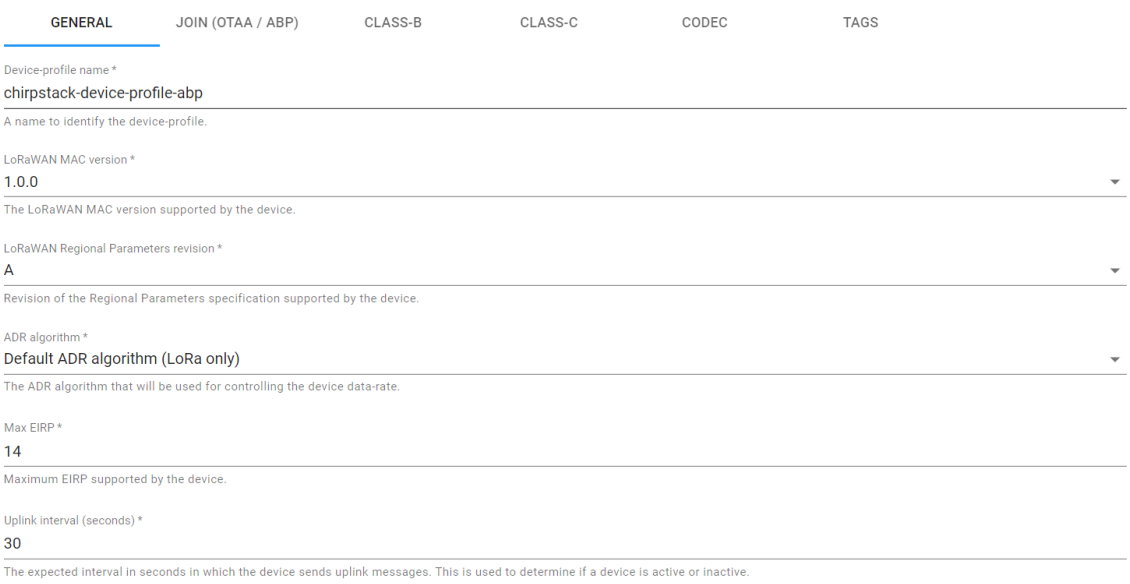
Figura I.6: *Gateway profile* Chirpstack



Vamos ahora a nuestra organización y creamos un *service profile*, *Service profile → Create*, en este caso solo marcamos la casilla de *Add gateway meta data* y el resto lo dejamos a 0 y sin marcar. Ahora crearemos un *gateway*, *Gateways → Create*, donde el *Gateway ID* es el *Gateway EUI*. Consecuentemente creamos una aplicación, *Applications → Create*. Después de este paso añadiremos un perfil del dispositivo, *Device profiles → Create*, lo configuraremos como en la figura I.7. En el caso de que queramos usar OTAA nos vamos a la pestaña *JOIN(OTAA/ABP)* y marcamos la casilla *Device Supports OTAA*. Por último, añadiremos un dispositivo dentro de la aplicación que acabamos de crear,

Applications → Entramos en la aplicación creada → *Create*, aquí deshabilitamos la casilla de *Disable frame counter validation*, en la pestaña *Activation* tenemos que introducir la *Device Adress*, *Network Session Key* y *Application Session Key*, que son las que tendremos que tener en el código de nuestro dispositivo.

Figura I.7: Perfil del dispositivo Chirpstack



II. Anexo manual de usuario.

En este anexo se va a explicar cómo, una vez teniendo el proyecto configurado, poner todo el sistema en marcha.

En primer lugar, tendremos que encender la Raspberry Pi y conectarnos a ella mediante MobaXterm, putty o cualquier programa que permite conectarnos mediante SSH. Una vez dentro tendremos que meternos en el directorio de Chirpstack e introducir el siguiente comando.

cd chirpstack-docker
sudo docker-compose up -d

Si todo ha funcionado correctamente deberíamos poder entrar a la plataforma de Chirpstack en *http://<IP address>:8080*. Una vez teniendo encendida la plataforma de Chirpstack podemos crear un JWT token en la REST API de la plataforma como se explica en el manual de instalación. Este token tenemos que copiarlo en introducirlo en el archivo de configuración, *config.ini*, en el apartado *appserverjwttoken.* Para ello tendremos que ir al directorio de la Raspberry Pi *lorawan-ran-slicing → network-slice-server/* y modificar el fichero → *config.ini*. Terminado este paso podemos ejecutar el servidor de red mediante el siguiente comando.

sudo python3 ./server.py

Una vez se haya ejecutado se quedará en espera hasta que reciba un *#JoinReq*. Después de este paso tendremos que conectar el *gateway* y la mota para que empiece a enviar *#JoinReq* y el sistema empiece a funcionar. En el caso de que devuelva un *#JoinACC* sabremos que está funcionando, aun así, lo podemos comprobar en el servidor de red en el apartado *Applications → app → Devices → Device Data* para ver los mensajes que se están intercambiando y el RB que se ha utilizado.

Bibliografía

[1] *Objetivos de desarrollo sostenibles (SDGs).* Disponible online: https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/ (último acceso 24/06/2022).

[2] IoT Analytics, State of the IoT 2020: 12 billion IoT connections, surpassing non-IoT for the first time. Disponible online: https://iot-analytics.com/state-of-the-iot-2020-12-billion-iot-connections-surpassing-non-iot-for-the-first-time/ (último acceso 24/06/2022).

[3] Dawaliby, S.; Bradai, A.; Pousset, Y. "Adaptive dynamic network slicing in LoRa networks". *Future Generation Computer Systems.* 2018, 98, 697-707.

[4] Campolo, C.; Molinaro, A.; Iera, A.; Menichella, F.; "5G Network Slicing for Vehicle-to-Everything Services". *IEEE Wireless Communications*. 2017. 24. 38-45.

[5] Pycom, *Lopy 4.* Disponible online: https://pycom.io/product/lopy4/ (último acceso 24/06/2022).

[6] Pycom, *Pygate.* Disponible online: https://pycom.io/product/pygate/ (último acceso 24/06/2022).

[7] *Página de descarga de Atom.* Disponible online: https://atom.io/ (último acceso 24/06/2022).

[8] *Página oficial de Chirpstack.* Disponible online: https://www.chirpstack.io/ (último acceso 24/06/2022).

[9] *Página de descarga de GanttProject.* Disponible online: https://www.ganttproject.biz/ (último acceso 24/06/2022).

[10] *Página de descarga de MobaXterm.* Disponible online: https://mobaxterm.mobatek.net/ (último acceso 24/06/2022).

[11] Semtech. *What is LoRa.* Disponible online: https://www.semtech.com/lora/what-is-lora (último acceso 24/06/2022).

[12] Laurens Slats, *A Brief History of LoRa: Three inventors share their personal story at thing conference*, 8 Jan 2020. Disponible online: https://blog.semtech.com/a-brief-history-of-lora-three-inventors-share-their-personal-story-at-the-things-conference (último acceso 24/06/2022).

[13] Chinchilla-Romero, N.; Navarro-Ortiz, J.; Muñoz, P.; Ameigeiras P. “Collision Avoidance Resource Allocation for LoRaWAN”. *Sensors*. 2021; 21(4):1218.

[14] What is LoRaWAN. A technical overview of LoRa and LoRaWAN. Disponible online: https://lora-alliance.org/wp-content/uploads/2020/11/what-is-lorawan.pdf (último acceso 24/06/2022).

[15] S. Cheong, J. Bergs, C. Hawinkel and J. Famaey, "Comparison of LoRaWAN classes and their power consumption," *IEEE Symposium on Communications and Vehicular Technology*, 2017, pp. 1-6.

[16] Noura, H.; Hatoum, T.; Salman, O.; Yaacoub, J.; Chehab, A. “LoRaWAN security survey: Issues, threats and possible mitigation techniques” *Internet of Things*. 2020. 12. 100303.

[17] Sundaram, J.P.S.; Du, W.; Zhao, Z. “A Survey on LoRa Networking: Research Problems, Current Solutions and Open Issues”. *IEEE Commun. Surv. Tutorials* 2019, 22, 371–388.

[18] Adelantado, F.; Vilajosana, X.; Tuset-Peiro, P.; Martinez, B.; Melia-Segui, J.; Watteyne, T. “Understanding the limits of LoRaWAN”. *IEEE Commun. Mag*. 2017, 55, 34–40.

[19] *Reglamento ITU.* Disponible online: https://life.itu.int/radioclub/rr/art1.pdf (último acceso 24/06/2022).

[20] *Regulaciones ETSI.* Disponible online: https://www.etsi.org/ (último acceso 24/06/2022).

[21] *Estándar MQTT 5.0. Oasis Standard.* 7/03/2019. Disponible online: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf (último accceso 24/06/2022).

[22] *Página oficial de Mosquitto.* Disponible online: https://mosquitto.org/ (último accceso 24/06/2022).

[23] *Página oficial de POSTGRESQL.* Disponible online: https://www.postgresql.org/about/ (último accceso 24/06/2022)

[24] *Página oficial de JWT*. Disponible online: https://jwt.io/introduction (último accceso 24/06/2022).

[25] Jones, M.; Bradley, J.; Sakimura, N. "JSON Web Token (JWT)", RFC 7519, 2015, Disponible online: https://www.rfc-editor.org/info/rfc7519 (último accceso 24/06/2022).

[26] *Impacto SF en el RSSI.* Disponible online: https://www.thethingsnetwork.org/docs/lorawan/spreading-factors/ (último

acceso 24/06/2022).

[27] Farhad, A.; Kim, D.; Pyun, J. “Resource Allocation to Massive Internet of Things in LoRaWANs”. *Sensors.* 2020. 20(9):20.

[28] *Página oficial de Node.js.* Disponible online: https://nodejs.org/en/ (último acceso 24/06/2022).

[29] *Página oficial de Python.* Disponible online: https://www.python.org/ (último acceso 24/06/2022).

[30] *Página de descarga de Firmware Update Tool.* Disponible online: https://docs.pycom.io/updatefirmware/device/ (último acceso 24/06/2022).

[31] *Repositorio GitHub personal.* Disponible online: https://github.com/mtorresantunez/LoRaWAN-Pygate.git (último acceso 24/06/2022).