UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO

INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Interconexión de Miembros de un Equipo de Emergencias en Entornos sin Cobertura usando Dispositivos Móviles

Departamento de Teoría de la Señal, Telemática y Comunicaciones

Autor
Félix Delgado Ferro

Director
Jorge Navarro Ortiz

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, julio de 2020

Interconexión de Miembros de un Equipo de Emergencias en Entornos sin Cobertura usando Dispositivos Móviles

Departamento de Teoría de la Señal, Telemática y Comunicaciones

Autor
Félix Delgado Ferro

Director
Jorge Navarro Ortiz

Interconexión de Miembros de un Equipo de Emergencias en Entornos sin Cobertura usando Dispositivos Móviles

Félix Delgado Ferro

Palabras clave: Internet de las Cosas, Redes Inalámbricas, LoRa, LoRaWAN, The Things Network, Chirpstack, *Bluetooth Low Energy*, Android, Servidores, Interfaz Web.

Resumen

Hoy en día, las tecnologías referentes al Internet de las Cosas IoT incrementan sus integraciones exponencialmente en la sociedad actual. Estas tecnologías suelen emplear elementos de bajo consumo que se emplean para la creación de redes de amplio alcance a bajo coste como son las redes LPWAN.

Entre las posibles alternativas que se pueden desplegar se distinguen LoRaWAN y LoRa dadas sus características. Algunas de las características más relevantes son su bajo consumo energético y su amplio rango de cobertura. Por otro lado, se destacan tecnologías de comunicaciones de corto alcance y bajo consumo energético como puede ser BLE. Esta tecnología implica un bajo ancho de banda, baja potencia y baja complejidad. Además, ambas tecnologías emplean las bandas ISM sin licencia lo que permite una implementación más sencilla.

En este trabajo se describe una red diseñada para su funcionamiento en zonas sin cobertura empleando tecnologías diversas para las comunicaciones como BLE, LoRa y LoRaWAN. Posteriormente, se muestran la implementaciones e integraciones realizadas sobre los dispositivos que forman la red. Por último, se presenta de forma detallada las comprobaciones y pruebas realizadas sobre el sistema que permiten asegurar el correcto funcionamiento del mismo.

System to Interconnect Members of an Emergency Team in Scenarios without Coverage using Mobile Devices

Félix Delgado Ferro

Keywords: Internet of Things, Wireless Networks, LoRa, LoRaWAN, The Things Network, Chirpstack, *Bluetooth Low Energy*, Android, Servers, Web Interface.

Abstract

Nowadays, Internet of Things IoT technologies increase their integrations exponentially in the society. These technologies often use low-power elements that are used to create low-cost and wide-ranging networks such as LPWAN.

There are different alternatives that can be used for the deployment of a network. One of the main technologies is LoRaWAN and LoRa. Some of the most relevant characteristics are low energy consumption and wide coverage range. On the other hand, there are short-range communication technologies such as BLE. This technology implies low bandwidth, low power and low complexity. In addition, both technologies employ ISM bands without a license, which makes them attractive for certain scenarios.

In this project, we first describe the design of a network that works in areas without coverage. This network incorporates various communication technologies such as BLE, LoRa and LoRaWAN. Second, the implementation and integration performed on the devices that make up the network are shown and, finally, the functionalities of the system are thoroughly tested and evaluated.

Yo, **Félix Delgado Ferro**, 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 XXX, 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: Félix Delgado Ferro

Granada a 3 de Julio de 2020.

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

Informa:

Que el presente trabajo, titulado ***Interconexión de Miembros de un Equipo de Emergencias en Entornos sin Cobertura usando Dispositivos Móviles***, ha sido realizado bajo su supervisión por **Félix Delgado Ferro**, y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expide y firma el presente informe en Granada a día 7 de julio de 2020.

El director:

Jorge Navarro Ortiz

Agradecimientos

A lo largo de la carrera, he sufrido momentos de tristeza, ansiedad, depresión, sacrificios y muchas alegrías. En dichos momentos siempre he tenido cerca a personas que me han apoyado y dado los ánimos suficientes para superar las dificultades y seguir el camino. Por este motivo, me gustaría darles las gracias a todos ellos y especialmente:

A mis padres, Rafael y Noni, por creer siempre en mis posibilidades y ofrecernos todo lo que se necesitabamos en cada momento, tanto a mi como a mi hermana. Por preocuparos por nosotros, aunque os digamos que sois cansinos. Y gracias por la educación que nos inculcasteis porque la educación comienza en casa.

A mi hermana, Esther, por estar siempre en los momentos oportunos y ayudarme en los difíciles. Por animarme, aconsejarme y sorprenderse con cada trabajo, por simple que fuera, como una niña pequeña. Y por todos los momentos buenos que hemos pasado juntos.

A mi profesor y tutor, Jorge, por todo el tiempo empleado a lo largo de este proyecto como el tiempo dedicado como profesor desde tercero. Por la inmensa ayuda que he recibido a lo largo de estos últimos meses y por mantener la paciencia en todo momento.

Y a todos los amigos y amigas por compartir tantos momentos de risas y alegrías. Por la ayuda en los momentos difíciles y el apoyo incondicional que me habéis otorgado.

Muchas gracias a todos.

Índice general

Índice de figuras

Índice de tablas

Glosario

ABP Activation By Personalization.

ACK Acknowledgement.

AP Access Point.

AppEUI Application Identifier.

AppKey Application Key.

AppSKey Application Session Key.

ATT Protocolo de Atributo.

B2B Business-to-business.

BLE Bluetooth Low Energy.

BR/EDR Bluetooth Basic Rate/ Enhanced Data Rate.

BSS Basic Service Set.

BW Ancho de Banda.

CCK Complementary Code Keying.

CEPT Conferencia Europea de Administraciones de Correos y Telecomunicaciones.

CFP Período libre de contención.

CP Período con contención.

CSMA/CA Carrier Sense Multiple Access with Collision Avoidance.

CSS Chirp Spread Spectrum.

CTS Clear to Send.

DCF Distributed Coordination Function.

DevAddr Device Address.

DevEUI Device Ifentifier.

DHCP Dynamic Host Configuration Protocol.

DIFS DCF Inter Frame Space.

DQPSK Differential Quadrature Phase Shift Keying.

DSSS Direct Sequence Spread Sprectrum.

ECDH Ellimptic Curve Diffie-Hellman.

EEUU Estado Unidos.

EIFS Extended Inter Frame Space.

ESS Extended Service Set.

ETH Instituto Federal de Tecnología de Zurich.

ETSI European Telecommunications Standards Institute.

FHSS Frequency Hopping Spread Spectrum.

FSK Frequency Shift Keying.

FTP File Transfer Protocol.

GAP Perfil Genérico de Acceso.

GATT Perfil Genérico de Atributo.

GFSK Gaussian Frecuency Shift Keying.

GPS Global Positioning System.

HCI Interfaz de Control de Host.

HTML HyperText Markup Language.

IDE Integrated Development Environment.

IEEE Insitute of Electrical and Electronics Engineers.

IoT Internet of Things.

IP Internet Protocol.

ISM Industrial, Scientific and Medical.

ITU International Telecommunication Union.

JWT JSON Web Token.

L2CAP Logical Link Control and Adaptation Protocol.

LAN Local Area Network.

LL Capa de Enlace.

LoRa Long Range.

LoRaWAN LoRa Wide Area Network.

LPWAN Low-Power Wide Area Network.

MAC Medium Access Control.

MAC Media Access Control Address.

MANET Mobile Ad hoc Network.

MIMO Multiple Input Multiple Output.

MQTT Message Queing Telemetry Transport.

MSDU MAC service data unit.

NB-IoT NarrowBand-IoT.

NTP Network Time Protocol.

NwkSKey Network Session Key.

OFDM Ortogonal Frecuency Division Multiplexing.

OSI Open Systems Interconnection.

OSM Open Street Map.

OTAA Over-The-Air Activation.

PC Point Coordinator.

PCF Point Coordination Function.

PDU Protocol Data Unit.

PHY Capa Física.

PIFS PCF Inter Frame Space.

PLCP Physical Layer Convergence Protocol.

PMD Physical Medium Dependant.

REGA Guardia Aérea Suiza de Rescate.

RSSI Received Signal Strength Indicator.

RTS Request to Send.

S.O.S Send Out Succour.

SF Spreading Factor.

SIFS Short Inter Frame Space.

SIG Bluetooth Special Interest Group, Inc..

SMP Protocolo de Gestión de Seguridad.

SNR Signal to Noise.

SSH Secure SHell.

TCP Transmission Control Protocol.

TLS Transport Layer Security.

TTN The Things Network.

UDP User Datagram Protocol.

UUID Universally Unique Identifier.

VCS Virtual Carrier Sensing.

VHT Very High Throughput.

Wi-Fi Wireless-Fidelity.

WLAN Wireless Local Area Networks.

WPAN Wireless Personal Area Networks.

Capítulo 1

Introducción

1.1. Contexto y Motivación

En la actualidad, las comunicaciones inalámbricas son parte de la sociedad del siglo XXI. Esto se debe a que las tecnologías siguen avanzando a pasos agigantados y, por tanto, no son las mismas que se conocían hace pocos años. Estos avances nos han permitido comunicarnos e interactuar entre nosotros y con el entorno.

Hoy en día, se habla de las tecnologías IoT. Este concepto fue mencionado por primera vez por Kevin Ashton en 1999 [1], pero fue Gartner [2] quien definió el concepto como: "*The Internet of Things (IoT) is the network of physical objects that contain embedded technology to communicate and sense or interact with their internal states or the external environment.*"

El crecimiento de la conexión global se debe principalmente a los dispositivos IoT, tanto en el lado del consumidor (por ejemplo, *smart homes*) como en el lado de la empresa / Business-to-business (B2B) (por ejemplo, maquinaria conectada). Para contextualizar esta evolución, los dispositivos dedicados al IoT se han disparado, llegando en enero de 2018 a la interconexión de 16 mil millones de dispositivos. Estas estadísticas podemos verlas en la figura 1.1.

Además, se espera que la cantidad de dispositivos IoT que estén activos aumente a 10 mil millones para 2020 y a 22 mil millones para 2025. Esta cantidad de dispositivos IoT incluye todas las conexiones activas. Esto implicará un aumento del 17 % de los dispositivos interconectados [3], como podemos ver en la figura 1.2.

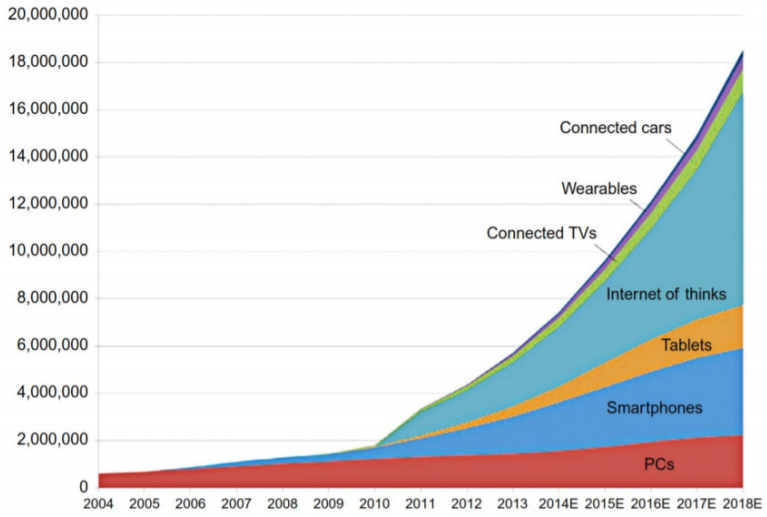


Figura 1.1: Número de dispositivos en uso globalmente (en miles) [3]

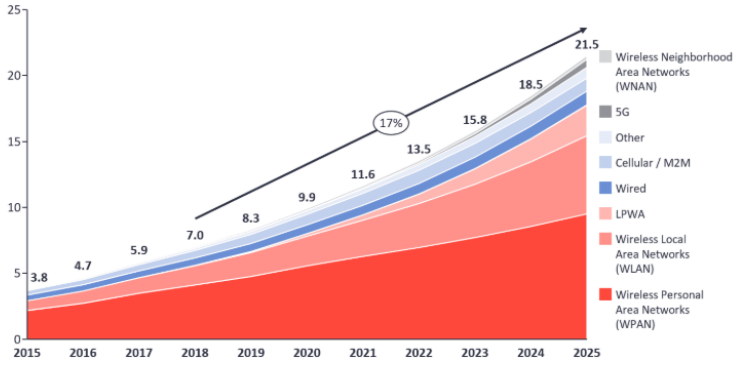


Figura 1.2: Número de conexiones IoT [3]

Las principales funcionalidades de las tecnologías IoT son: envío de datos de tamaño reducido, bajo consumo de energía y rentabilidad. Las principales redes inalámbricas se clasifican según su área de cobertura tal y como sigue:

- **Wireless Personal Area Networks (WPAN):** son redes que normalmente no superan los 100 metros en el rango máximo. Estas incluyen tecnologías inalámbricas como Bluetooth, Zigbee y Z-wave.
- **Wireless Local Area Networks (WLAN):** son redes que albergan una cobertura de hasta 1 kilómetro. La tecnología inalámbrica por excelencia es Wi-Fi.
- **Low-Power Wide Area Network (LPWAN):** son redes de área amplia (max. 20 kilometros) y de baja potencia. Los estándares principales son Sigfox, LoRa y NB-IoT.

- **Celular:** son redes que ofrecen cobertura a áreas celulares. Los principales estándares son 2G, 3G y 4G.

En España existen zonas rurales donde no es posible ofrecer cobertura por parte de los operadores. Por tanto, este trabajo se centra en el uso de diversas tecnologías de comunicación inalámbrica, es decir, se centrará en el uso de Bluetooth, LoRa y LoRaWAN para la conectividad de los miembros del equipo de emergencias en zonas sin cobertura. Los principales motivos para la elección de estas tecnologías son el uso de las bandas ISM y las características ofrecidas por cada una de estas.

1.2. Objetivos y logros conseguidos

En esta sección se exponen los principales objetivos y logros conseguidos de este proyecto.

1.2.1. Objetivos

Los principales objetivos del proyecto son:

- Análisis de los principales artículos y contribuciones relacionadas con las redes LoRaWAN y tecnologías BLE en entornos similares al de este proyecto.
- Desarrollo e implementación de una red de comunicaciones en zonas sin cobertura.
- Desarrollo e implementación de la comunicación Bluetooth Low Energy entre los dispositivos móviles y las pasarelas LoRa.
- Almacenamiento de información de usuario en base de datos, para su posterior visualización.

1.2.2. Logros conseguidos

Los logros conseguidos a lo largo del proyecto son:

- Recopilación de información y proyectos relacionados con la creación de redes en zonas sin cobertura. Además, se ha recopilado información necesaria de los protocolos y tecnologías empleadas para el desarrollo de nuestra red.
- Diseño e implementación de la red de comunicación empleando los protocolos BLE, LoRa y LoRaWAN.

- Diseño e implementación de una aplicación móvil compatible con el 98 % de los dispositivos Android. La app es totalmente funcional para los miembros del equipo de emergencias y cumple con la función de transmitir mensajes.
- Implementación de servidores y base de datos para el almacenamiento y visualización del comportamiento de la red.
- Implementación de un interfaz web para la visualización del estado de la red.

1.3. Metodología

En esta sección, se explicará las tareas o procedimientos que nos permitirán alcanzar nuestro objetivo. A continuación, se definen las fases a seguir para implementar la red deseada, a la vez que logramos con los objetivos anteriores.

En este proyecto utilizaremos una metodología lineal. Esta metodología nos permite definir de forma clara y concisa las líneas de trabajo y si se han conseguido las metas a lo largo del desarrollo del proyecto.

Primero, se definen las fases del proyecto para estructurar el modo de trabajo. Por este motivo, se han definido previamente los objetivos que queremos alcanzar. Al finalizar una tarea, es posible volver a esta para modificar sus funciones, si fuese necesario. Para el desarrollo de una solución efectiva y eficiente, debemos analizar previamente las posibles soluciones que existen en la actualidad. Este análisis nos permitirá poder distinguir nuestra solución de las demás.

El siguiente paso es definir claramente el funcionamiento de nuestra alternativa y llegado este momento, debemos analizar y establecer los requisitos. Posteriormente, se seleccionarán los componentes necesarios para el desarrollo de nuestra red. Previo a la programación de las diferentes partes de la red, será necesario preparar los entornos de desarrollo y documentarnos sobre su funcionamiento.

A continuación, procederemos al diseño, implementación e integración de los programas desarrollados. Es importante tener en cuenta las capacidades de los elementos empleados y ceñirse a las funcionalidades que estos nos permitan. En la siguiente fase será necesario realizar una serie de pruebas de conceptos y análisis de resultados que nos permitan concluir si cumplimos los objetivos previstos. Por último, se comentarán las conclusiones obtenidas y las posibles líneas de trabajos futuros.

1.4. Estructura de la memoria

En esta sección, se exponen las diferentes partes que forman la memoria, explicando brevemente de qué trata cada una:

- **Capítulo 1 - Introducción:** Este primer capítulo se compone de las siguientes partes:
  - *Contexto y motivación:* consiste en una primera visión del problema que se pretende resolver en este trabajo, así como destacar la importancia del mismo.
  - *Objetivos y logros conseguidos:* detalla los objetivos principales a resolver y cúales se han logrado.
  - *Metodología:* trata los pasos seguidos a lo largo del desarrollo del trabajo.
  - *Estructura de la memoria:* describe el contenido de cada capítulo de la memoria con el fin de ubicar al lector.
- **Capítulo 2 - Estado del Arte:** Este capítulo trata de mostrar las alternativas existentes en este momento para resolver el problema del proyecto.
- **Capítulo 3 - Fundamentos Teóricos:** Este capítulo alberga todos los conocimientos teóricos necesarios para la comprensión y realización del proyecto. Se subdivide el capítulo en:
  - *Bandas ISM:* explicación de qué son y para qué se emplean las bandas de ISM.
  - *Bluetooth Low Energy:* explicación de las características principales y del protocolo BLE.
  - *LoRa & LoRaWAN:* explicación de las características principales y del protocolo de comunicaciones para LoRaWAN.
- **Capítulo 4 - Planificación y Coste Estimado:** Este capítulo se subdivide en tres partes.
  - *Planificación Temporal:* descibe las etapas del proyecto a lo largo del tiempo mediante un diagrama de Gantt.
  - *Recursos:* indica los recursos hardware, software y humanos necesarios.
  - *Presupuesto Estimado del Proyecto:* consiste en un presupuesto final del proyecto.

- **Capítulo 5 - Diseño e implementación:** describe los pasos seguidos para el diseño, implemetación e integración de la solución obtenida para los objetivos prefijados.
- **Capítulo 6 - Pruebas:** expone la puesta en marcha de la solución y las pruebas de testeo realizadas para comprobar el correcto funcionamiento de la red creada.
- **Capítulo 7 - Conclusiones y Trabajos Futuros:** presenta las conclusiones obtenidas tras la elaboración del proyecto y un análisis de las posible vías de actuación para la futura actualización y mejora de funcionalidades.

Capítulo 2

Estado del Arte

Este capítulo pondrá en conocimiento algunos de los avances o investigaciones en el entorno de las redes en zonas sin cobertura. Además, se analizarán ejemplos de soluciones similares que existen en la actualidad al problema a solventar de conectividad en zonas sin cobertura.

En este tipo de proyectos, como se ha comentado previamente, existen diversas tecnologías inalámbricas para resolver el problema y, a su vez, distintos desarrollos de red y productos comerciales. A continuación, se expondrán los considerados más relevantes, clasificados de mayor a menor importancia hoy en día.

- **Lantern** [4]: es un dispositivo desarrollado por Outernet [5]. Este dispositivo autorrecargable y portable surgió a partir de una iniciativa que consiste en crear una infraestructura para ofrecer cobertura de Internet utilizando satélites y Wi-Fi. En la figura 2.1 se muestra el dispositivo de la alternativa Lantern.

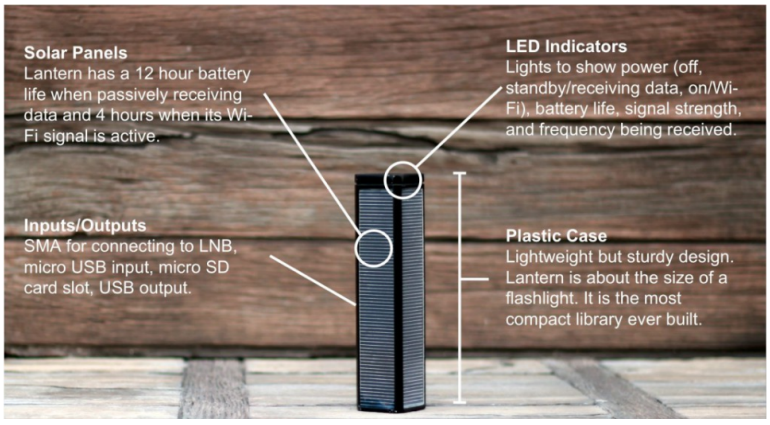


Figura 2.1: Dispositivo Lantern [4]

Lantern permite acceder anónimamente y gratis a Internet mediante la infraestructura de Outernet. Además, puede recibir y almacenar el contenido digital de señales satelitales constantemente. Posteriormente, el usuario puede conectarse mediante Wi-Fi con su smartphone e acceder al contenido.

El principal problema de esta solución viene dada por el mismo dispositivo, es decir, el dispositivo es capaz de emplear tecnología satelital y retransmitirla vía Wi-Fi, lo que supone que el alcance en el que el usuario puede moverse es limitado.

El precio actual del dispositivo en el mercado es de 91,62 €.

- **Commotion Wireless** [6]: consiste en un kit de autoconstrucción para la creación de redes inalámbricas comunitarias. Este kit se forma tanto de elementos hardware como software para la creación de una red mesh (mallada).

  Commotion Wireless tiene capacidad para mantener aplicaciones únicas y personalizadas para la red comunitaria a la que ofrece cobertura [7].

  Esta solución es ideal para la creación de cobertura en un entorno acotado como puede ser una comunidad de vecinos o una pequeña zona residencial. Por lo tanto, no es la mejor solución para entornos sin cobertura de largo alcance como son los propuestos en el problema.

- **SPOT X** [8]: es un sistema creado por la compañía Globalstar. Se constituye de un dispositivo hardware que proporciona conectividad a un smartphone a través de Bluetooth con la aplicación SPOT X.

Figura 2.2: Dispositivo SPOT X [8]

Esta aplicación permite mensajería satelital con cualquier número de teléfono móvil o dirección email, comunicación directa con los servicios de emergencias a través de mensaje S.O.S y geolocalización. El dispositivo SPOT X se muestra en la figura 2.2.

Esta alternativa incorpora un amplio rango de cobertura y varias funcionalidades muy útiles para solucionar el problema, pero aún así, no permite el mensaje en todas las circunstancias, ni se almacenan los mensajes para una posterior revisión.

Su precio actual en el mercado es de 231.33 €.

- **Uepaa! App** [9]: es una aplicación desarrollada por la empresa suiza Ueppa AG en colaboración con el Instituto Federal de Tecnología de Zurich (ETH) [10], la empresa suiza de telecomunicaciones Swisscom [11] y el servicio de rescate aéreo REGA [12].

  Esta aplicación fue pensada para accidentse en montañas. Sus principales funcionalidades son alarma específica de la empresa cliente, detección digitalizada de accidentes y ayuda directa por miembros de la aplicación. En la figura 2.3 se muestra el funcionamiento de la App Uepaa!

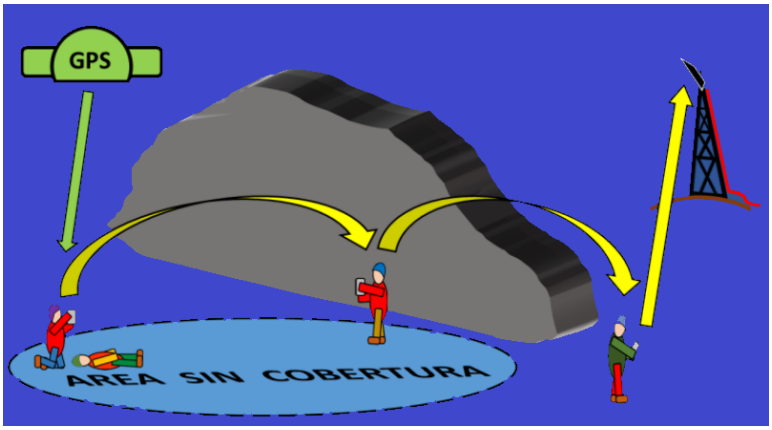


Figura 2.3: Funcionamiento Ueppa! App [13]

Este funcionamiento consiste en pulsar un botón de emergencia, entonces se envía un mensaje de la emergencia junto a las coordenadas obtenidas del GPS del smartphone. Si otro usuario de la aplicación se encuentra a menos de 2 kilómetros, recibe el mensaje y lo transmite formando una cadena multisalto. Incluso, el mensaje pasa a la red telefónica y es recibido por los servicios de emergencias en caso de no haber más usuarios alrededor y tener conectividad telefónica.

Uepaa! es una aplicación gratuita para teléfonos Iphone y android. Actualmente está disponible en Suiza, Alemania, Austria, Francia e Italia. Al igual que la alternativa anterior, esta solución no permite el envío de mensajes entre los miembros de un equipo de emergencias ni el almacenamiento de los mensajes para una posterior visualización.

- **Beartooth** [14]: es un dispositivo que se conecta al smartphone a través de Bluetooth y permite hablar, enviar de mensajes y geolocalización sin necesidad de conectividad externa.

  Beartooth crea una red local entre sus usuarios conectados en un rango de entre 3 y 8 kilómetros. No permite la conectividad a Internet, únicamente entre sus usuarios. La aplicación y el dispositivo asociado se muestran en la figura 2.4.

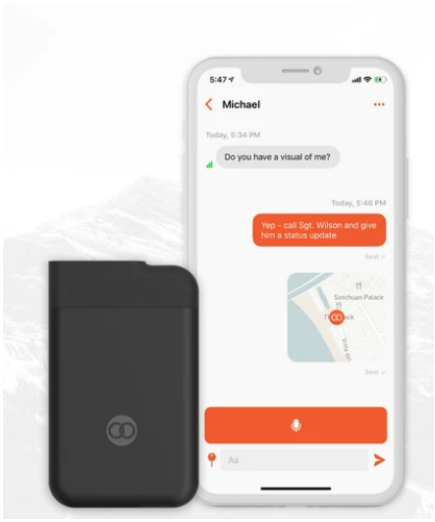


Figura 2.4: Dispositivo y App Beartooth [14]

Esta solución es la más completa de las alternativas mencionadas anteriormente, esta solución implementa el envío de la localización y de mensajes entre usuarios. Pero estos mensajes son únicamente de ubicaciones donde ha ocurrido un accidente.

Su precio actual en el mercado es de 231.33 €.

Contemplando todas las alternativas anteriores, se ha decidido incorporar en nuestra solución una serie de funcionalidades que no incorporan las mencionadas. Estas funcionalidades se corresponden con el envío de la localización y de mensajes grupales entre los distintos miembros del grupo de emergencias. El problema de un amplio rango de cobertura en zonas que no se tiene cobertura previamente. Para solucionarlo se ha empleando una red LoRaWAN que nos permite rangos de cobertura de 3-30 km en zonas

rurales. Además, en nuestra solución se generarán una serie de servidores que nos permitan la visualización posterior de los mensajes enviados por los miembros de un equipo de emergencias.

Capítulo 3

Fundamentos Teóricos

En este capítulo nos centramos en entender los conceptos teóricos necesarios para fundamentar el proyecto. Primero, se hará una breve descripción sobre las bandas ISM. A continuación, se tratarán las principales tecnologías y protocolos de comunicaciones, incluyendo sus principales características, su arquitectura de red y su seguridad.

Entre estas tecnologías mencionaremos las empleadas en el desarrollo del proyecto. Estas son Bluetooth Low Energy (BLE), LoRa y LoRaWAN. En este último apartado, se describirá la arquitectura de la plataforma Chirpstack y el protocolo MQTT.

3.1. Bandas ISM

Las bandas radio Industrial, Scientific and Medical (ISM) se establecieron por primera vez en la Conferencia Internacional de las Telecomunicaciones de la ITU en Atlantic City, 1947. Estas bandas radio ISM, en los úñtimos años, se han compartido por la comunidad con licencia libre para aplicaciones industriales, científicas y médicas [15].

En Europa, el uso de las bandas ISM está cubierto por las regulaciones de la Comisión Europea, basándose en las recomendaciones técnicas de la CEPT [16] y las normas del ETSI [17].

En relación a la figura 3.1, podemos clasificar las tecnologías que emplean cada una de estas bandas como:

- **Banda 868 - 900 MHz:** utilizada por Zigbee, LoRa y Sigfox.
- **Banda 2.4 GHz:** utilizada por Wi-Fi, Bluetooth y Zigbee.
- **Banda 5 GHz:** utilizada por Wi-Fi.

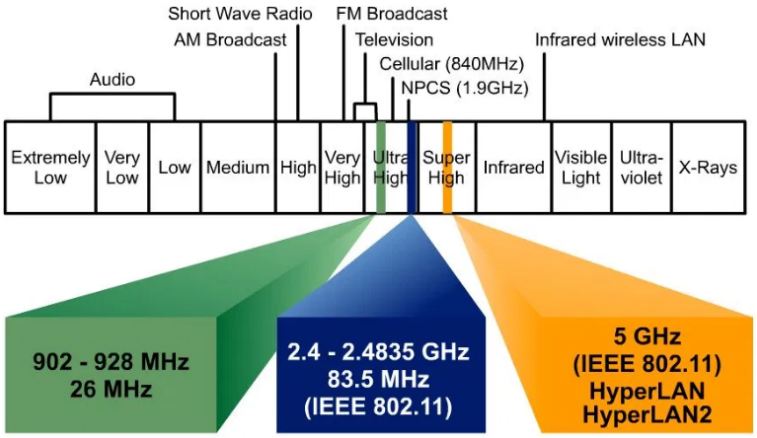


Figura 3.1: Bandas Radio ISM [18]

3.2. Bluetooth Low Energy

Bluetooth [19] es una tecnología creada por Bluetooth Special Interest Group, Inc. (SIG) [20] en 1989. Esta tecnología inalámbrica de corto alcance está basada originalmente en el estándar IEEE 802.15.1 [21]. Este protocolo permite el intercambio de información a baja velocidad entre dispositivos inalámbricos compatibles a través de una banda de frecuencia de radio ISM (Banda 2.4 GHz) [22]. A continuación, se muestra el logo de Bluetooth en la figura 3.2

Figura 3.2: Logo Bluetooth [19]

Bluetooth Low Energy es parte de la versión 4.0 del núcleo de especificaciones Bluetooth [23]. Esta versión fue introducida en junio de 2010 por el grupo SIG. Aunque se basa en Bluetooth Classic, podemos considerar BLE como una tecnología inalámbrica independiente, dado que intenta solventar objetivos diferentes a Bluetooth Classic.

3.2.1. Características principales

El principal objetivo de BLE es diseñar un estándar de radio con el menor consumo de energía posible, es decir, optimizado para tener un bajo

coste, con bajo ancho de banda, baja potencia y baja complejidad. Además, facilita las comunicaciones entre dispositivos en un corto rango de cobertura, con la idea de poder monitorizar y controlar las transmisiones de datos.

Actualmente en el mercado podemos encontrarnos tres versiones diferentes en los dispositivos Bluetooth [20].

- **Bluetooth clásico (BR/EDR):** implementa el estándar inalámbrico clásico Bluetooth.
- **Monomodo (BLE, Bluetooth Smart):** implementa el estándar BLE y puede comunicarse con otros módulos monomodo y con los de modo dual pero no con los que sólo soportan BR/EDR.
- **Modo dual (BR/EDR/LE, Bluetooth Smart Ready):** implementa ambos estándares, BR/EDR y BLE y es capaz de conectarse con cualquier dispositivo Bluetooth.

En la figura 3.3, podemos apreciar como realiza la conectividad entre dispositivos Bluetooth con distintas versiones.

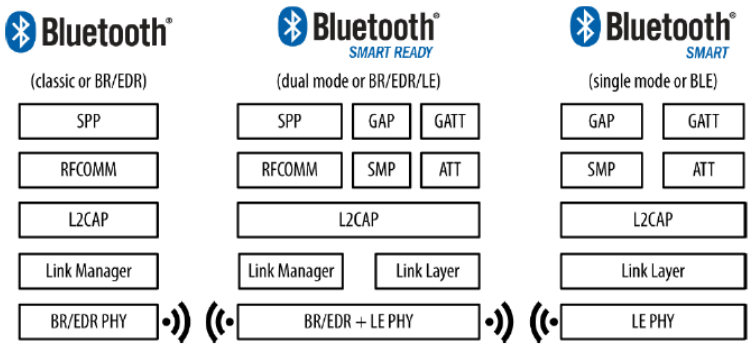


Figura 3.3: Conectividad entre dispositivos con diferentes versiones [24]

Dentro de la versión 4, existen tres subversiones: 4.0 (junio de 2010), posteriormente han surgido dos más, la 4.1 (diciembre de 2013) y 4.2 (diciembre de 2014). Algunas de sus diferencias son:

- Los dispositivos Bluetooth 4.2 son retrocompatibles con las otras subversiones.
- La subversión Bluetooth 4.2 extiende la longitud de los paquetes de datos de los 27 bytes a los 251 bytes.
- La subversión Bluetooth 4.2 incluye el protocolo de seguridad Ellimptic Curve Diffie-Hellman (ECDH). Es un protocolo criptográfico que permite que dos dispositivos que no hayan interactuado antes puedan intercambiar información de forma segura.

3.2.2. Protocolo de Comunicación

El protocolo Bluetooth Low Energy mantiene una estructura de tres partes: controlador, host y aplicación. Estos albergan todas las funcionalidades de su pila de protocolos, como se muestran en la figura 3.4.

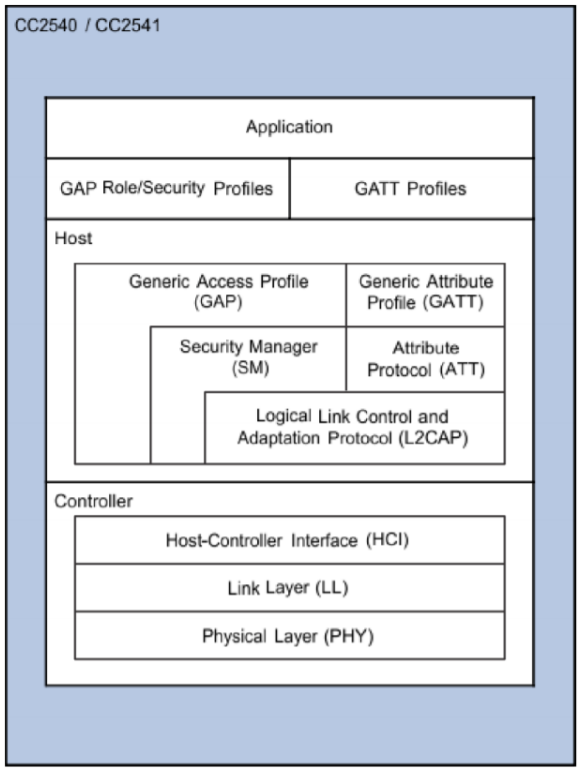


Figura 3.4: Pila de Protocolos de BLE [25]

Controlador

- **Capa Física (PHY)**

  La capa física se constituye por los circuitos que proporcionan la comunicación (modulación y demodulación de la señal, transformación analógico-digital ...). Este protocolo opera en la banda ISM de 2.4 GHz (2.4000GHz - 2.4835GHz) que define 40 canales de 2MHz de ancho de banda. Existen dos tipos de canales:

- *Canales de anuncio:* utilizados para el descubrimiento de dispositivos, transmisión de mensajes y configuración de conexiones. Estos canales (37,38 y 39) se utilizan periódicamente y han sido asignados para minimizar la colisión con los canales Wi-Fi más frecuentes (1,6, y 11). Estos se pueden apreciar en la figura 3.5.

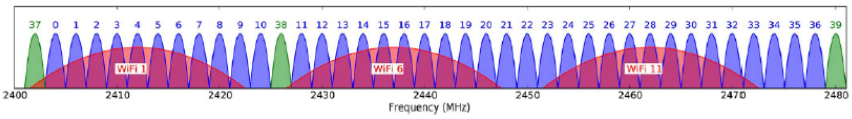


Figura 3.5: Canales BLE y los canales más usado de Wi-Fi [26]

- *Canales de datos:* utilizados para las comunicaciones bidireccionales entre dispositivos interconectados. Dado que se podría colisionar con los canales Wi-Fi o con otros dispositivos BLE que quieran transmitir sobre el mismo canal, se utiliza una técnica Spread-Spectrum Frecuency Hopping. Esta técnica consiste en dividir la banda de frecuencias en varios canales y, durante la comunicación, se producen saltos de un canal a otro, como se muestra en la figura 3.6.

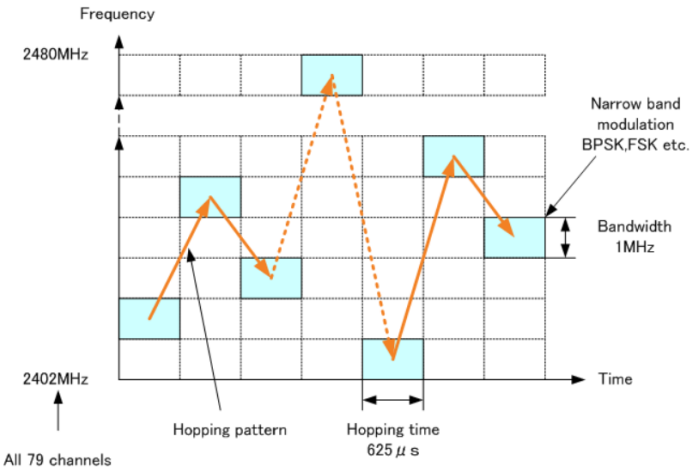


Figura 3.6: Funcionamiento de Spread-Spectrum Frecuency Hopping [27]

Todos los canales usan un esquema de modulación Gaussian Frecuency Shift Keying (GFSK) que reduce el pico de consumo. Además, el máximo ratio de modulación es 1Mbps, aunque no llega a alcanzarse.

- **Capa de Enlace (LL)**

  La capa de enlace interactúa directamente con la capa física y se encarga de gestionar las conexiones entre dispositivos. BLE mantiene una estructura asimétrica aplicando la estructura maestro-esclavo, donde el maestro suele ser el dispositivo con más recursos (smartphone, tablet ...) y el esclavo, dispositivos más simples y con más limitaciones de memoria. En esta capa, se definen cuatro posibles roles:

  - *Anunciante:* envía mensajes de anuncio.
  - *Escaneador:* escanea paquetes de anuncio.
  - *Maestro:* inicia la conexión y gestiona después.
  - *Esclavo:* acepta peticiones de conexión.

  Además, en la capa de enlace se describe su funcionamiento como una máquina de estados. En la figura 3.8, podemos determinar cinco estados:

  - *Espera:* no transmite ni recibe paquetes.
  - *Anuncio:* transmite paquetes de anuncio, escucha y contesta a los posibles mensajes de petición-respuesta de escaneo.
  - *Escaneo:* escucha a paquetes de anuncio.
  - *Inicio:* escucha a paquetes de anuncio y responde para iniciar una conexión.
  - *Conectado*

  En la figura 3.7, se muestra la máquina de estados.

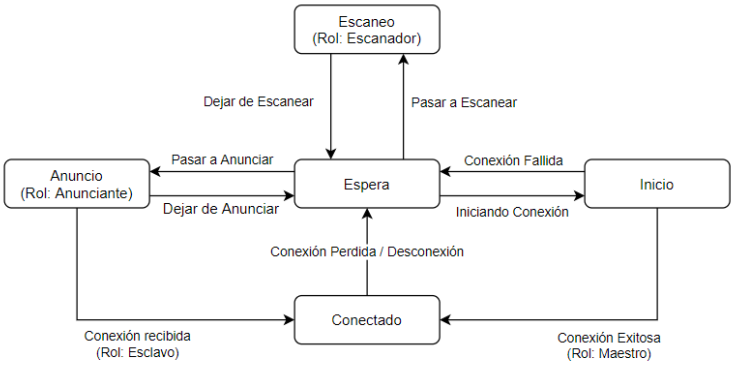


Figura 3.7: Maquina de Estados de la Capa de Enlace

La estructura del paquete BLE en la capa de enlace viene dada por la figura 3.8. Y se compone de:

- *Preámbulo [1Byte]:* permite la sincronización de los paquetes.
- *Dirección de Acceso [4Bytes]:* si es un anuncio tendrá un valor fijo (0x8E89BED6); en cambio, si se trata de un paquete de datos, tendrá un valor aleatorio generado por el maestro.
- *PDU [2-39 Bytes]:* este campo depende del tipo de mensaje/proceso a transmitir. Las PDUs pueden ser de anuncio, de escaneo, de inicio de conexión, de datos ...
- *CRC [3 Bytes]:* permite comprobar y detectar fallos en la trasmisión.

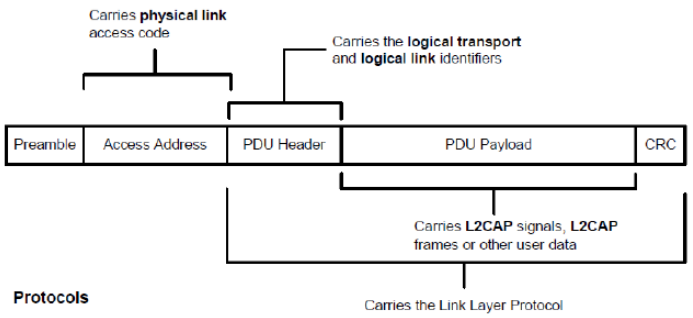


Figura 3.8: Estructura del Paquete de la Capa de Enlace [28]

- **Interfaz de Control de Host (HCI)**

La Interfaz de Control de Host (HCI) es la encargada de la comunicación entre el controlador y el host. La interfaz solamente permite un tipo de formato de mensaje, aunque engloba tanto a los paquetes de anuncios como a los paquetes de datos.

Host

- **Protocolo de Control Lógico y Adaptación de Enlace (L2CAP)**

Este protocolo L2CAP se encarga de multiplexar los protocolos de las capas superiores y encapsularlos en un formato estándar de BLE. Este protocolo se asemeja al protocolo TCP en el sentido de que permite coexistir a diversos protocolos superiores sobre un mismo enlace físico. La cabacera del protocolo L2CAP es de 4 Bytes y los protocolos que directamente superiores en la pila son: Protocolo de Atributo (ATT) y Protocolo de Gestión de Seguridad (SMP) [23].

- **Protocolo de Atributo (ATT)**

  El Protocolo de Atributo [23] es simple, se trata de un protocolo cliente/servidor donde la comunicación se realiza a través de solicitud-respuesta, basado en los atributos que contiene cada servidor. Los atributos están formados por un identificador de dirección, un identificador de tipo de contenido, un conjunto de permisos (escritura y lectura) y un valor.

- **Protocolo de Gestión de Seguridad (SMP)**

  Este protocolo [23] se forma por un conjunto de algoritmos de seguridad que permiten a la pila de protocolos la capacidad de generar e intercambiar claves de seguridad. Este protocolo consiste en tres procedimientos:

  - *Emparejamiento:* genera una clave de cifrado común y temporal.
  - *Vinculación:* intercambia las claves de seguridad permanentes que se encuentran en memoria y crea un vínculo entre los dos dispositivos.
  - *Cifrado de restablecimiento:* define cómo usar las claves almacenadas para restablecer próximas conexiones.

- **Perfil Genérico de Atributo (GATT)**
  El Perfil Genérico de Atributo (GATT) [23] se basa en el Protocolo de Atributo (ATT). El GATT define la organización y el intercambio de los datos entre aplicaciones, a través de una arquitectura cliente/servidor.

  Bluetooth SIG define algunos perfiles GATT como Find Me Profile, Proximity Profile, ... [29]. Los dispositivos interctúan entre ellos dependiendo de dos perfiles principalmente.

  - *Cliente:* se encarga del envío de peticiones al servidor y recibe respuestas desde el servidor. Dado que no conoce al servidor previamente, debe preguntar primero por los atributos del servicio.
  - *Servidor:* se encarga de recibir peticiones del cliente y dar respuesta. A su vez, almacena los datos y los usa según el servicio solicitado.

  Un servicio está definido a partir de un Universally Unique Identifier (UUID) de 16 Bytes. Los UUID para BLE deben tener la base XXXXXXXX-0000-1000-8000-00805F9B34FB, e.g. 12345678-0000-1000-8000-00805F9B34FB.

Los servicios, que se muestran en la figura 3.9, se constituyen por atributos y cada atributo por:

- *Gestor:* indica la dirección del servicio GATT unívocamente.
- *Tipo:* indica el tipo de atributo.
- *Permisos:* describe el tipo de procesos permitidos, e.g. escritura, lectura, autenticación ...
- *Valor*

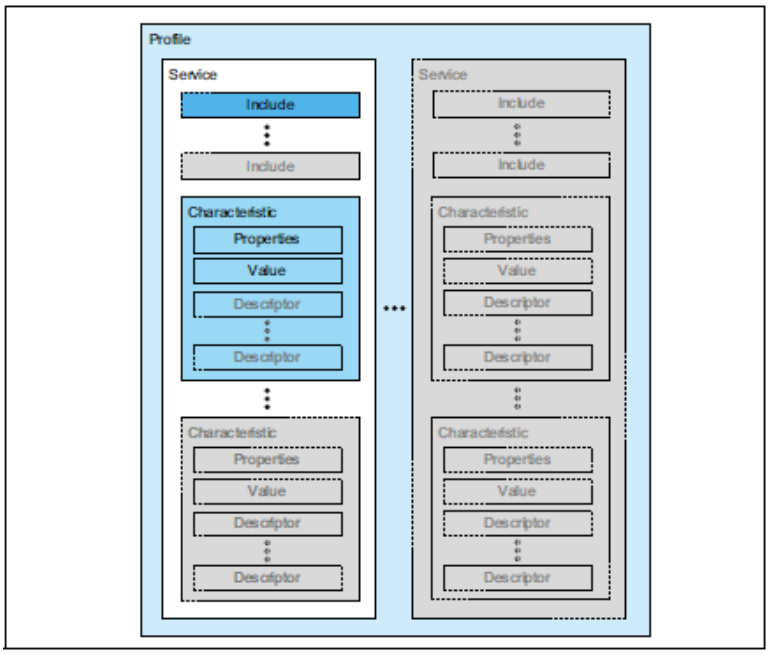


Figura 3.9: Perfil GATT [23]

- **Perfil Genérico de Acceso (GAP)**

El Perfil Genérico de Acceso (GAP) [23] se centra en el control, detección, conexión y difusión de dispositivos. Obviamente, estos perfiles son transversales a los protocolos de BLE.

Un dispositivo BLE puede trabajar en uno o más roles a la vez, aunque cada rol tendrá sus normas de comportamiento. Existen cuatro alternativas de rol:

- *Central:* permite establecer múltiples conexiones siendo el maestro. Escucha paquetes de anuncio de otros dispositivos y conecta con el dispositivo.

- *Periférico:* permite la conexión siendo el esclavo de esta. Envía paquetes de anuncio a central y espera para establecer la conexión.
- *Emisor:* envía mensajes de anuncio con datos periódicamente.
- *Observador:* recibe datos de los dispositivos emisores.

3.3. LoRa & LoRaWAN

LoRa Alliance [30] es una asociación tecnológica abierta y sin fines de lucro que se ha expandido desde marzo de 2015. Sus miembros colaboran y comparten experiencias para promover e impulsar el éxito del protocolo LoRaWAN como el estándar global abierto líder para una conectividad IoT LPWAN segura y de nivel de operador. A continuación, se muestra el logo de LoRa Alliance en la figura 3.10.

Figura 3.10: Logo LoRa Alliance [30]

LoRa Wide Area Network (LoRaWAN) es una especificación para redes Low-Power Wide Area Network (LPWAN) propuesta por LoRa Alliance [30] y diseñada para dispositivos de bajo consumo. LoRaWAN define tanto el protocolo de comunicaciones como la arquitectura de red. Así, indica cómo se unen los dispositivos de una red LoRaWAN, gestionando sus canales y parámetros de conexión como el canal usado, el ancho de banda, el cifrado ...

3.3.1. Protocolo de Comunicación

El protocolo LoRaWAN [31] describe las capas: Capa Física (PHY) y Capa de Enlace (LL); aunque más específicamente, la capa Medium Access Control (MAC), dentro del sistema de referencia OSI. En la figura 3.11 se representan las capas mencionadas.

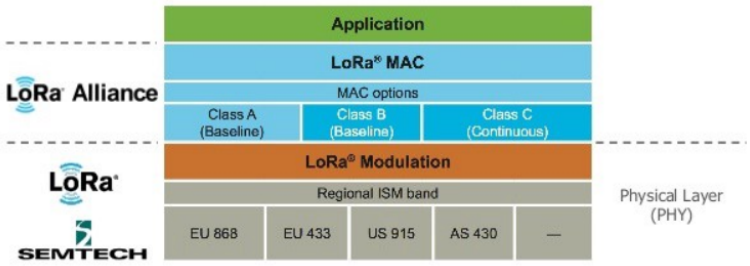


Figura 3.11: Pila de Protocolos de LoRaWAN [32]

Capa Física (PHY)

La capa física se constituye por los circuitos que proporcionan la comunicación (modulación y demodulación de la señal, transformación analógico-digital . . .). Este protocolo opera en la banda ISM de 433 MHz (Asia), 868 MHz (Europa) o 915 MHz (EEUU).

Long Range (LoRa) es un esquema de modulación inalámbrico propietario de Semtech [33] que se emplea sobre la capa física. Su objetivo principal es establecer enlaces de comunicación a largo alcance mediante el uso de baja potencia, aunque permite alcanzar amplios rangos de cobertura para la comunicación.

LoRa permite el envío y recepción de información punto a punto y, para ello, utiliza la técnica denominada Chirp Spread Spectrum (CSS). Esta tecnología implica que la señal usa más ancho de banda del necesario. Esto permite una recepción de múltiples señales a la vez, mantienendo las mismas características de baja potencia que la modulación Frequency Shift Keying (FSK), pero incrementando el rango de la comunicación [34]. Un pulso chirp es una frecuencia de barrido en el ancho de banda correspondiente (125kHz, 250kHz ...). Se puede ver una representación en la figura 3.12.

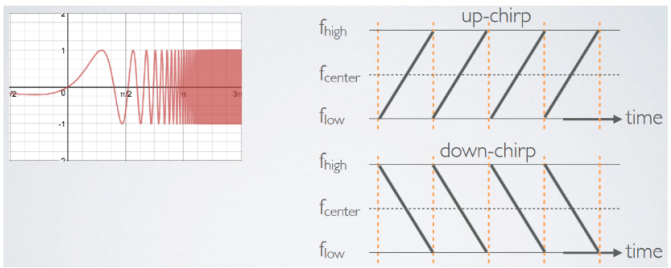


Figura 3.12: Representación de Chirp Spread Spectrum [35]

Además, LoRa define una serie de parámetros de comunicación [36] que son:

- *Canal:* es un número que representa la portadora. Al situarnos sobre la zona Europea, se emplea la portadora de 868 MHz.
- *Ratio de codificación:* indica la forma de codificar para la detección y corrección de errores.
- *Ancho de Banda (BW):* indica el rango de frecuencias del canal.
- *Chirp:* es una rampa de f_{min} a f_{max} (up-chirp) o f_{max} a f_{min} (down chirp). Los chirps que transportan datos son aquellos que se desplazan cíclicamente. La información se encuentra en estos cambios cíclicos.
- *Spreading Factor (SF):* define el número de bits usados para codificar un símbolo. Este ensanche espectral permite un compromiso entre robustez y cobertura. Los SFs son ortogonales entre ellos, por lo que diferentes dispositivos pueden transmitir simultáneamente en diferentes SFs sin colisiones. A continuación, se representan los factores de dispersión en la figura 3.13.

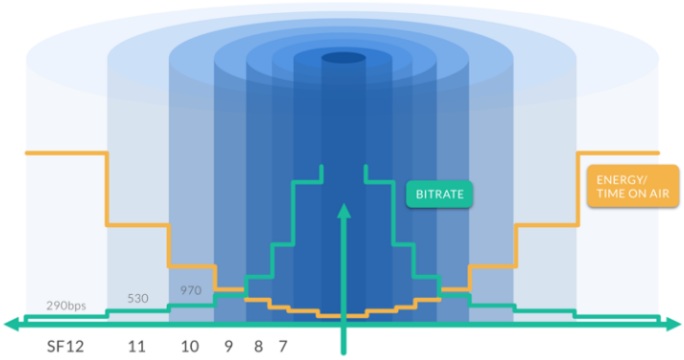


Figura 3.13: Representación de los Factores de Dispersión [37]

Capa MAC

La especificación LoRaWAN define tres tipos de dispositivos según la bidireccionalidad [38]:

- **Clase A:** los dispositivos de esta clase admiten comunicaciones bidireccionales. Los mensajes del enlace ascendente (desde el dispositivo al servidor) se permiten enviar en cualquier instante. Después le siguen dos ventanas de recepción de enlace descendente (desde el servidor al dispositivo) cortas (1-2 seg.). Entonces, el dispositivo debe programar

la siguiente transmisión en función de las necesidades de la comunicación añadiendo una variación aleatoria del tiempo (similar al protocolo ALOHA). Esta clase se caracteriza por ser la clase que necesita menor potencia, dado que el servidor solo se comunica después de la transmisión de un dispositivo final.

- **Clase B:** es una extensión de la clase A. Consiste en añadir ventanas de recepción para los mensajes en el enlace descendente. Para ello, se emplean balizas sincronizadas (beacon) enviadas por el gateway, permitiendo al servidor saber cuándo escuchar al dispositivo.

- **Clase C:** es una extensión de la clase A. Consiste en mantener abierta las ventanas de recepción en todo momento, excepto cuando se transmite. Esto permite una comunicación con menor latencia, a pesar de un consumo excesivo de la energía.

Los distintas clases de LoRaWAN se encuentran representadas en la figura 3.14.

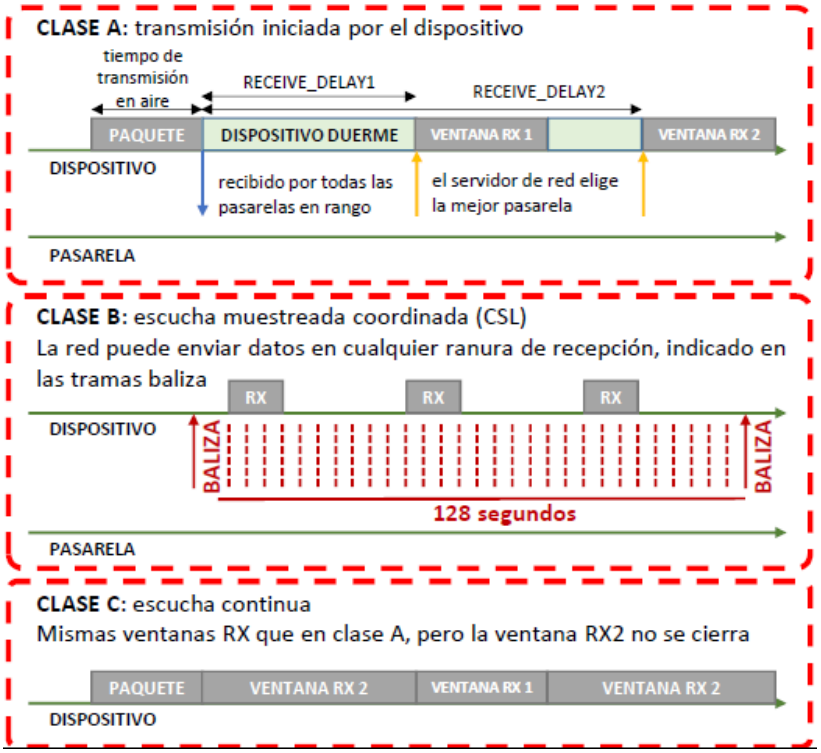


Figura 3.14: Funcionamiento de las Clases de LoRaWAN [39]

La capa Medium Access Control (MAC) se encarga de gestionar el acceso al medio por parte de diferentes dispositivos LoRa, gestionando sus canales y parámetros de conexión. El uso de los canales viene limitado por ciclo de trabajo máximo. El ciclo de trabajo indica la fracción de tiempo que el medio puede encontrarse ocupado (1 % en el caso de Europa, pudiendo llegar a ser del 0.1 % según la región). Aunque LoRaWAN permite la confirmación de tramas, en general no se utiliza ya que la pasarela también tiene la limitación del ciclo de trabajo.

3.3.2. Arquitectura de Red

LoRaWAN define una arquitectura de red y la funcionalidad de seguridad que implementa a varias capas.

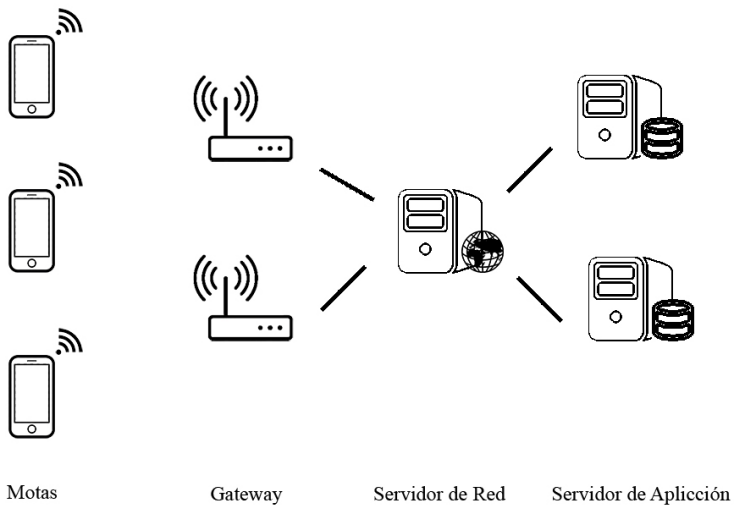


Figura 3.15: Arquitectura de Red LoRaWAN

En la figura 3.15 se muestra la red LoRaWAN al completo. Esta red está formada por cuatro tipos de dispositivos [40]:

- **Nodo Final (mota):** dispositivos integrados de comunicación de baja potencia (e.g. un dispositivo basado en el microcontrolador ESP32).
- **Gateway:** dispositivo que recibe las transmisiones de las motas y envía datos de respuesta a estas (similar a un bridge).
- **Servidor de Red:** dispositivo que enruta los mensajes entre las motas y los servidores de aplicación y viceversa.
- **Servidor de Aplicación:** dispositivo que alberga el software que se ejecuta como aplicación.

Seguridad en LoRaWAN

Esta arquitectura presenta una característica muy relevante y especificada en el protocolo LoRaWAN. Esta consiste en una seguridad de red a dos capas, es decir, se emplea una seguridad a nivel de red (capa 3 de OSI) y otra a nivel de aplicación (capa 7 de OSI).

La seguridad a nivel de red se encarga de la autenticación de los nodos de la red e integridad de los mensajes en la red; mientras que la seguridad a nivel de aplicación consiste en confidencialidad de los nodos. En pocas palabras, se encarga de que el servidor de red no tenga acceso a los datos finales de usuario, separando así la seguridad en la infraestructura y en el servicio. En resumen, la seguridad en LoRaWAN permite autenticación mutua, integridad y confidencialidad de los paquetes enviados.

Cada dispositivo final perteneciente a la red LoRaWAN debe estar correctamente personalizado y activado, es decir, debe de incorporar dos claves que se pueden derivar de los datos enviados.

- **Network Session Key (NwkSKey):** es la clave de seguridad contra el servidor de red.
- **Application Session Key (AppSKey):** es la clave de seguridad contra el servidor de aplicación.

Además, existen dos tipos de activación [41] posibles que incorporan algunos parámetros extras, estos podemos verlos gráficamente en la figura 3.16.

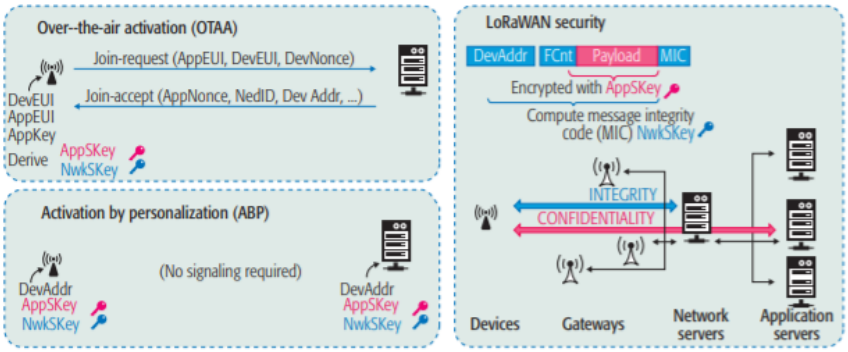


Figura 3.16: Tipos de Activación LoRaWAN [42]

- **Over-The-Air Activation (OTAA):** la mota o nodo final se encarga de crear la nueva sesión con el servidor de red. Entonces se genera una dirección nueva y las claves de sesión: NwkSKey y AppSKey . Después, se realiza el proceso de unión donde se tranmiten los parámetros Device Ifentifier (DevEUI), Application Identifier (AppEUI) y Application Key (AppKey).
- **Activation By Personalization (ABP):** el nodo final debe estar correctamente personalizado con los siguientes parámetros: Device Address (DevAddr), AppEUI, NwkSKey y AppSKey. Entonces se produce el proceso de unión.

Chirpstack

La parte más importante de la red LoRaWAN son los servidores. Estos se encargan de la comunicación entre la aplicación y el nodo final y viceversa. Debido a su relevancia, surgió a el proyecto LoRaServer, más tarde llamado ChirpStack [43]. Este proyecto de código abierto permite la construcción de una red LoRaWAN a través de los siguientes componentes:

- **Chirpstack Gateway Bridge:** se encarga de las comunicaciones actuando como una pasarela. Para ello, reenvía los paquetes del protocolo UDP del packet forwarder (implementado en la pasarela) sobre el protocolo MQTT.

  Message Queing Telemetry Transport (MQTT) es un protocolo de comunicaciones basado en el paradigma publicador-suscriptor. El nodo broker se encarga de gestionar la red y reenviar los mensajes, mientras los nodos clientes (suscriptores) publican y reciben los mensajes.
- **Chirpstack Network Server:** se encarga de gestionar las tramas del enlace ascendente y programar las transmisiones de datos en el enlace descendente.
- **Chirpstack App Server:** se encarga de las solicitudes transmitidas desde los nodos finales. Además, incluye un interfaz web para la configuración de dispositivos, gateways y aplicaciones.

Estos componentes permiten la creación de una infraestructura privada para una red LoRaWAN como la que se muestra en la figura 3.17 . Otra forma de implementarlo es utilizando la infraestructura pública como e.g. la desplegada por The Things Network [44].

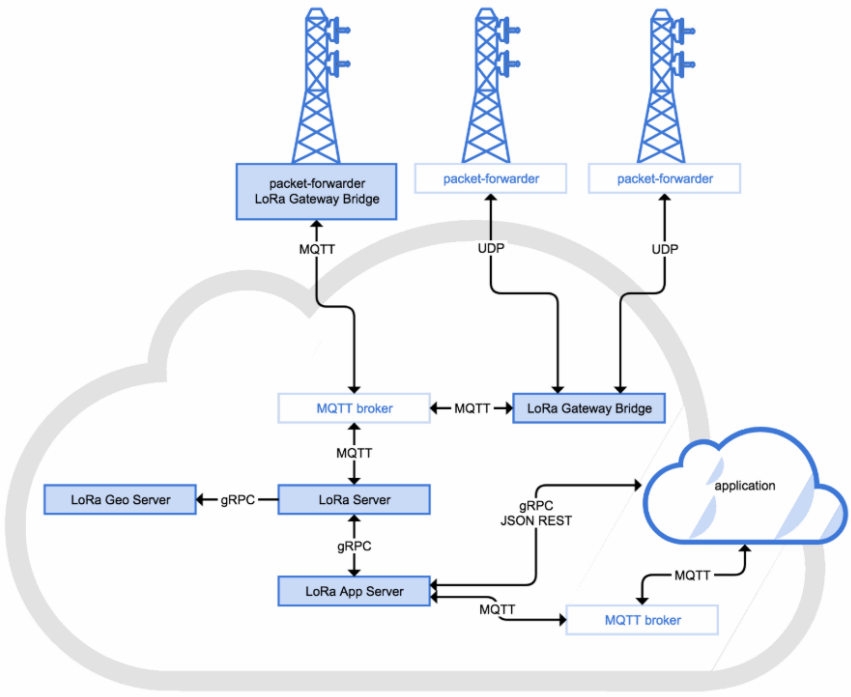


Figura 3.17: Arquitectura de la Plataforma Chirpstack [43]

Capítulo 4

Planificación y Coste

En este capítulo se expondrá la planificación temporal para la realización del proyecto, los recursos hardware, software y humanos empleados en el desarrollo del mismo. Se concluirá con un presupuesto estimado del proyecto.

4.1. Planificación Temporal

El proyecto total ha tenido una duración de 245 días. Al comienzo del proyecto se definieron claramente los pasos a seguir a lo largo del mismo. Estos se recogen en la siguiente planificación inicial:

1. **Estudio y Análisis**

   - **Propuesta y Aceptación del Proyecto (18 días):** periodo de tiempo entre la propuesta del proyecto para el TFG y la asignación y aceptación del TFG oficialmente.
   - **Revisión del Estado del Arte (11 días):** estudio de soluciones alternativas al problema planteado en el TFG y el estado del mercado para dichas soluciones.
   - **Estudio de los Fundamentos Teóricos (27 días):** proceso de recopilación de información y estudio y compresión de las tecnologías seleccionadas para el desarrollo de la solución.
   - **Documentación (14 días):** redacción de la documentación hasta el momento y verificación de las referencias empleadas.

2. **Diseño e Implementación**

   - **Diseño de la Red (20 días):** análisis de los requerimientos necesarios y proceso del diseño de la red para dar solución al problema propuesto.

- **Implementación del Gateway y Servidores LoRa (13 días):** proceso de implementación del software necesario y la configuración para la puesta en marcha del gateway y los servidores.
- **Diseño e implementación de las Motas (13 días):** proceso de implementación del software necesario y la configuración para la puesta en marcha de las motas.
- **Diseño e implementación de la Aplicación Móvil (27 días):** proceso de diseño, desarrollo e implementación de una aplicación móvil funcional con la red creada.
- **Diseño e implementación de la Interfaz Web (20 días):** proceso de implementación de la base de datos y servidor web para la visualización de la funcionalidad de la red.

3. **Pruebas**

   - **Pruebas (34 días):** realización de pruebas de funcionamiento de las tecnologías desarrolladas hasta el momento.

4. **Redacción de Memoria y Entrega**

   - **Redacción de la memoria (28 días):** en esta fase se realiza la redacción completa del proyecto. Esta memoria recogerá de forma estructurada toda la información y resultados obtenidos a lo largo del desarrollo del TFG.
   - **Entrega de la memoria (7 días):** envío de la solicitud de defensa y entrega de material.
   - **Defensa del TFG (4 días):** defensa del TFG ante el Tribunal indicado.

En la siguiente página, se visualiza un diagrama de Gantt, figura 4.1, para facilitar la comprensión de la planificación inicial del proyecto, aunque se han realizado una posterior corrección de fechas en la parte referente a la documentación y entrega de la memoria para ser coherente con las últimas fechas impuestas. Además, en la figura 4.2 se muestra un cuadro resumen con el tiempo aproximado a emplear en cada tarea.

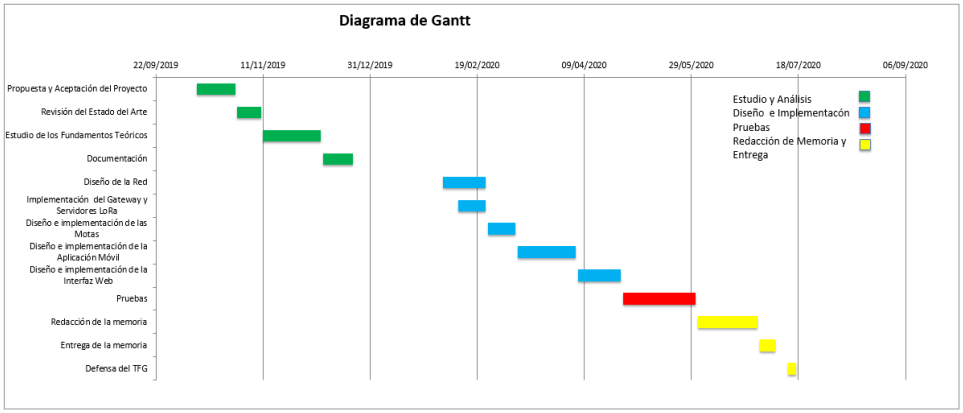


Figura 4.1: Diagrama de Gantt

Nombre de la tarea	Fecha de inicio	Fecha final	Duración (dias)
Propuesta y Aceptación del Proyecto	11/10/2019	29/10/2019	18
Revisión del Estado del Arte	30/10/2019	10/11/2019	11
Estudio de los Fundamentos Teóricos	11/11/2019	08/12/2019	27
Documentación	09/12/2019	23/12/2019	14
Diseño de la Red	03/02/2020	23/02/2020	20
Implementación del Gateway y Servidores LoRa	10/02/2020	23/02/2020	13
Diseño e implementación de las Motas	24/02/2020	08/03/2020	13
Diseño e implementación de la Aplicación Móvil	09/03/2020	05/04/2020	27
Diseño e implementación de la Interfaz Web	06/04/2020	26/04/2020	20
Pruebas	27/04/2020	31/05/2020	34
Redacción de la memoria	01/06/2020	29/06/2020	28
Entrega de la memoria	30/06/2020	07/07/2020	7
Defensa del TFG	13/07/2020	17/07/2020	4

Figura 4.2: Tabla de Planificación

4.2. Recursos

Esta subsección presenta los recursos utilizados durante el desarrollo del proyecto. Estos recursos se dividen en tres tipos: software, hardware y humanos.

4.2.1. Recursos Software

Los recursos software hacen referencia a los programas utilizados en el proyecto. A lo largo del proyecto se ha intentado emplear en la medida de lo posible software gratuito. Estos software usados son:

- **IDE de Arduino [45]:** es un entorno de desarrollo que nos facilita la labor de programar las aplicaciones de las motas ESP32. Es necesaria su configuración previa, pero cuenta con un editor de texto, un compilador, un depurador y la facilidad de implementación en la placa. En el proyecto se ha usado la versión 1.8.12.
- **Android Studio [46]:** es un entorno de desarrollo que nos facilita la labor de programar las aplicaciones del smartphone. Este programa incorpora un editor de texto, un compilador basado en Gradle, un depurador, un emulador rápido, integrador con GitHub y herramientas Lint para identificar problemas de rendimiento, usabilidad y compatibilidad entre versiones. Se ha utilizado la versión 3.6.3.
- **Chirpstack [47]:** es una plataforma de código abierto para la implementación de componentes para redes LoRaWAN. Además, incluye la interfaz web para la administración de dispositivos y API para la integración. Todos los componentes tienen licencia bajo la licencia MIT y pueden usarse con fines comerciales.
- **The Things Network [44]:** es una plataforma pública que implementa la estructura de redes LoRaWAN y las comparte con la comunidad. Además, incluye la interfaz web para la administración de dispositivos.
- **TTN Mapper [48]:** es una aplicación móvil y una página web que nos facilita la visualización de los dispositivos LoRa y la toma de mediciones de cobertura sobre plano.
- **nRF Connect [49]:** es una aplicación móvil que le permite escanear y explorar sus dispositivos Bluetooth Low Energy y comunicarse con ellos. Admite varios tareas como:
  - Escanear dispositivos Bluetooth Low Energy.
  - Mostrar gráficos de RSSI.
  - Permite leer / escribir características.

  - Permite habilitar / deshabilitar notificaciones e indicaciones.
  - Descubre y analiza servicios y características.

- **Visual Studio Code [50]:** es un editor de texto de los más conocidos y utilizados por la comunidad. Al ser un editor de texto, nos permite escribir y compilar diversos lenguajes de programación como son JavaScript, Html, CSS, TypeScript, etc. En este proyecto se ha empleado la versión 1.46.

- **PlatformIO [51]:** es un entorno de desarrollo de código abierto para entornos IoT, independientemente del hardware usado. Maneja todos los requerimientos a nivel de librerías de forma centralizada, posibilitando la migración entre plataformas y la compartición de código entre distintos miembros en equipo.

- **Wireshark [52]:** es el analizador de protocolos de red que permite ver lo que sucede en la red. Esta herramienta contiene un conjunto de características como: inspección de protocolos en tiempo real, captura en vivo y análisis outline.

En la tabla 4.1 se presenta el presupuesto del software empleado en el proyecto.

Software	**Coste**
Licencia Arduino IDE	0 €
Licencia Android Studio	0 €
Licencia Chirpstack	0 €
Licencia The Things Network	0 €
Licencia nRF Connect	0 €
Licencia Visual Studio Code	0 €
Licencia PlatformIO	0 €
Licencia Wireshark	0 €
Total	0 €

Tabla 4.1: Presupuesto de Recursos Software

4.2.2. Recursos Hardware

Los recursos hardware representan los componentes tecnológicos empleados en el proyecto. Estos elementos se definen a continuación:

- **Ordenador Personal:** es la principal herramienta necesaria para el desarrollo del proyecto, tanto en el ámbito tecnológico como en el administrativo. En el proyecto se ha utilizado un "HP Pavilion Laptop 15-cs2xxx". Se ha empleado para las etapas de investigación, diseño, implementación, pruebas y documentación principalmente.
- **Smartphone Personal:** este dispositivo se emplea para el diseño, implementación y pruebas de la aplicación móvil y de la red completa. Se ha utilizado un smartphone "Huawei P9 Lite (VNS-L31)".
- **Motas:** son los dispositivos que se encargan de la recepción de los mensajes a través de la aplicación móvil y actúan de repetidor para transmitirlas a las otras motas. En el proyecto, se han empleado "TTGO ESP32 LoRa".
- **Gateway LoRa:** se ha utilizado una pasarela "IMST Gateway Lite". Esta pasarela está formada por una Raspberry Pi 1 Modelo B+ y un concentrador IMST ic880A. El concentrador nos permite un alto rendimiento multicanal para la transmisión/recepción LoRa. Además de las funciones de gateway de la Raspberry, se ha empleado para la instalación de los servidores LoRa en su interior.

En la tabla 4.2 se representa las características principales de los componentes hardware empleados en el trabajo.

Componente	HP Pavilion Laptop	Huawei P9 Lite	TTGO ESP32 LoRa	IMST Gateway Lite
S.O	Windows 10 Ver. 1909	Android 7.0	-	Raspbian Ver. Buster
Procesador	Intel Core i7-8565U	Kirin 650	Xtensa LX6	ARM 6
Memoria RAM	12 GB	3 GB	512 KB	1GB
Almacenamiento	1 TB	16 GB	16 MB	32 GB

Tabla 4.2: Caracterísitas de los Recursos Hardware

En la tabla 4.1 se muestra el presupuesto de los componentes hardware utilizados.Considerando que el coste por unidad de los componentes de larga duración (laptop y spartphone) se han calculado considerando una amortizacion en 3 años y un tiempo de uso de 8 meses.

4.2.3. Recursos Humanos

En los recursos humanos se tendrá en cuenta a las personas que trabajaron en el proyecto:

Componente	Cantidad	Coste por Unidad	Coste
HP Pavilion Laptop	1	266.66 €	266.66 €
Huawei P9 Lite	1	28.22 €	28.22 €
TTGO ESP32 LoRa	4	14.20 €	56.80 €
IMST Gateway Lite	1	199.00 €	199.00 €
		Total	550.68 €

Tabla 4.3: Presupuesto de Recursos Hardware

- **Félix Delgado Ferro:** Estudiante del Grado en Ingeniería de Tecnologías de Telecomunicaciones de la Universidad de Granada.
- **Jorge Navarro Ortiz:** Profesor Titular de Universidad del Departamento de Señal Teoría, Telemática y Comunicaciones de la Universidad de Granada. Tutor del proyecto.

En la tabla 4.4 se muestra el presupuesto referente a los recursos humanos.

Persona	Horas Trabajadas	Precio por Hora	Salario
Félix Delgado Ferro	480	20 €	9600 €
Jorge Navarro Ortiz	40	50 €	2000 €
		Total	11600 €

Tabla 4.4: Presupuesto de Recursos Humanos

4.3. Presupuesto Estimado del Proyecto

Por último, se muestra el presupuesto estimado contando todos los costes de los recursos empleados y descritos anteriormente. Estos se recogen en la tabla 4.5.

Concepto	Presupuesto
Recursos Software	0 €
Recursos Hardware	550.68 €
Recursos Humanos	11600 €
Total	12150.68 €

Tabla 4.5: Presupuesto Estimado del Proyecto

El coste final del proyecto es de 12150.68 €, DOCE MIL CIENTO CINCUENTA EUROS Y SESENTA Y OCHO CÉNTIMOS.

Capítulo 5

Diseño e Implementación

En este capítulo se describen los pasos seguidos en el diseño, implementación e integración de la red TeamUp.

5.1. Diseño de la Red TeamUp

La idea principal del proyecto es desarrollar una red privada con infraestructura propia y móvil, capaz de transmitir información (mensajes y localización) desde cualquier dispositivo móvil al nodo central incluso si estos nodos se encuentran en zonas fuera de cobertura.

Durante el proceso de diseño de la red se analizarán las funcionalidades de cada uno de los dispositivos que conforman la red y las conexiones que se establecen entre cada par de nodos. El diseño final de la red podemos verlo en la figura 5.1.

1. **Dispositivo móvil:** el dispositivo móvil o smartphone se considera un dispositivo personal que únicamente se emplea por un miembro del equipo de emergencias. Este dispositivo incorpora la aplicación desarrollada para este proyecto (**TeamUp**).

2. **Mota:** es un dispositivo móvil que llevará consigo cada miembro del equipo de emergencias y que realiza la función de bridge. Es decir, se encarga de la recepción de mensajes desde el dispositivo móvil mediante el protocolo de comunicaciones BLE y cambio de tecnología del mensaje a LoRa para la retransmisión de los mensajes hasta el gateway.

3. **Gateway:** es un dispositivo que se encuentra cerca de nuestro nodo central (raíz). El gateway se encarga de la recepción de los mensajes LoRa provinientes de las motas y lo retransmite vía Wireless-Fidelity (Wi-Fi) al nodo central.

4. **Nodo central:** el nodo central puede permanecer de forma estática, es decir, mantenerse en un sitio fijo mientras se realiza una operación o dinámica, es decir, puede ser parte del equipo táctico móvil de un equipo de emergencias dependiendo del rango de cobertura necesaria. El nodo central se implementa e integra sobre un host, en nuestro caso, una Raspberry Pi. Se encarga de mantener actualizados los servidores (chirpstack y web).

   Estos servidores se emplean para el almacenamiento de los mensajes que se enviaron desde la aplicación móvil. Además, permite la recepción de solicitudes web. La respuesta a estas solicitudes corresponden con la visualización de una web donde se tendrán distintas funcionalidades como registro de usuarios, visualización de mensajes ...

5. **Interfaz Web:** la interfaz web es una GUI creada para facilitar la visualización de los datos a través de los navegadores web de los dispositivos inteligentes.

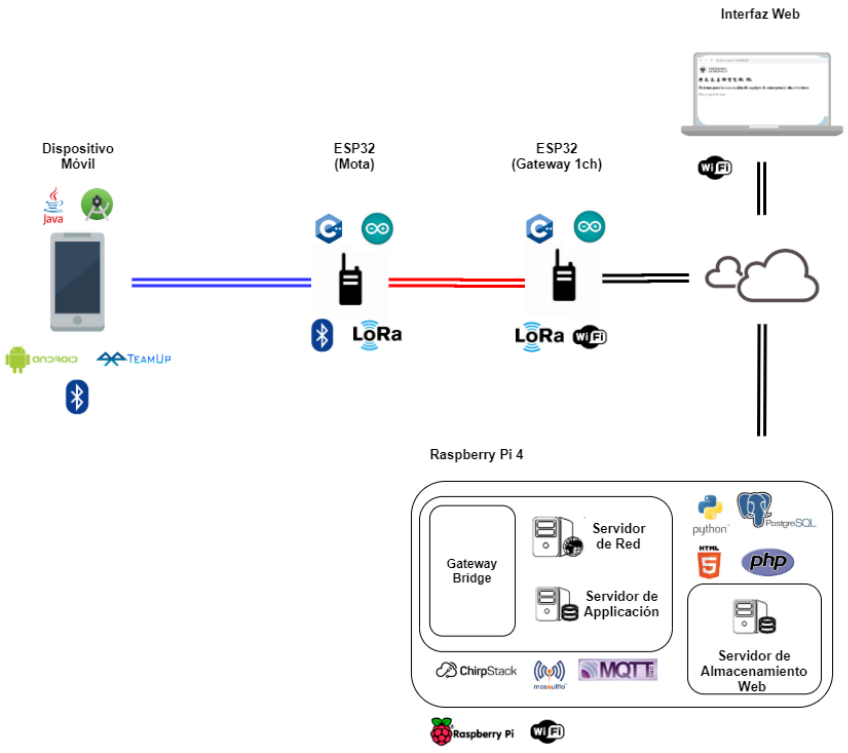


Figura 5.1: Diseño de la Red de Comunicaciones TeamUp

Empleando la red completa se permite el envío de los mensajes desde cualquier miembro de equipo de emergencias y la visualización de estos mensajes en formato web. Los dispositivos intermedios retransmiten la información hasta llegar a los servidores que permiten el almacenamiento de los mensajes y su posterior visualización.

5.2. Diseño e Implementación de la App TeamUp

La aplicación TeamUp ha sido desarrollada explícitamente para el proyecto empleando el entorno de desarrollo Android Studio y el lenguaje de programación Java. El desarrollo de la app consta de dos partes: diseño e implementación.

5.2.1. Diseño App TeamUp

El diseño de la aplicación consiste principalmente en la visualización de las distintas pantallas que la incorporan y en cómo el usuario puede navegar entre ellas. En la figura 5.2 se muestran estas pantallas y su forma de navegar de forma visual.

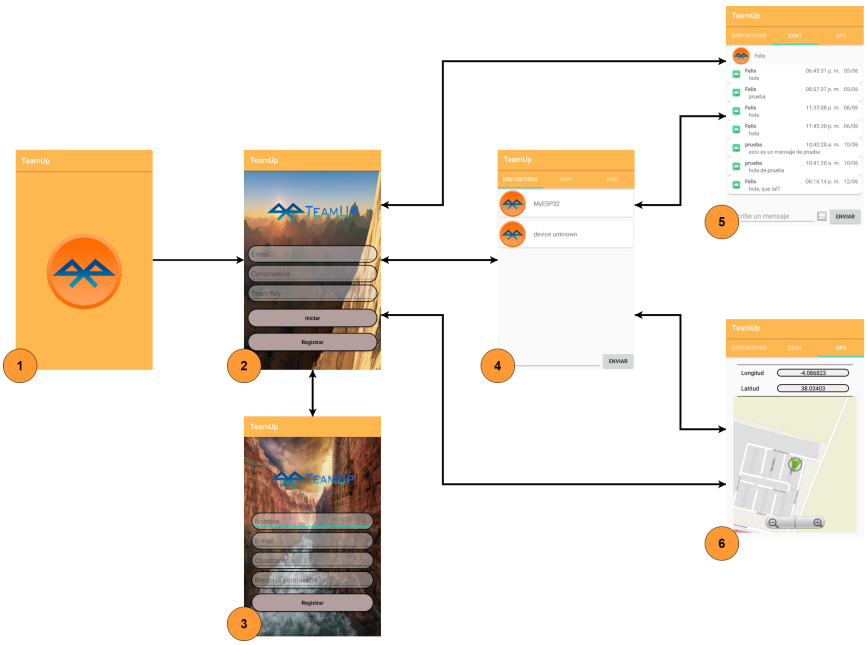


Figura 5.2: Navegación de la App TeamUp

1. **Portada:** pantalla que muestra el logo mientras se arranca la aplicación.

2. **Inicio de Sesión:** esta pantalla permite el inicio de sesion de un usuario que este registrado en la app y, por tanto, almacenado en la base de datos de la red.

3. **Registro:** esta pantalla permite el registro de nuevos usuarios en la aplicación.

4. **Dispositivos:** esta pantalla muestra los dispositivos escaneados que tengan conectividad BLE y permite el envío de mensajes a nuestras motas.

5. **Chat:** esta pantalla muestra los mensajes enviados y recibidos por los distintos miembros del equipo en forma de lista.

6. **GPS:** esta pantalla muestra las coordenadas y un mapa con la ubicación actual del miembro del equipo de emergencias.

5.2.2. Implementación App TeamUp

La implementación de la aplicación consiste en el desarrollo de las funcionalidades que debe de realizar para pasar entre distintos estados dentro de la misma aplicación. Dicho desarrollo se ha realizado de forma que la aplicación sea compatible para todos los dispositivos que incorporen una versión Android igual o superior a la 4.0.3 (Ice Cream Sandwich). Para ello, se han empleando los recursos por defecto de Android Studio y se han añadido una serie de librerías extras que se mencionan a continuación:

- **Firebase:** es una plataforma de Google que proporciona funciones como estadísticas, bases de datos, informes de fallos y mensajería, de manera que permite mejorar la eficiencia y el enfoque de los usuarios de la aplicación. Para la incorporación de estas funcionalidades se han incluido estas librerías:

```
firebase-core:17.4.2'
firebase-database:17.0.0'
firebase-storage:17.0.0'
firebase-auth:17.0.0'
```

- **Open Street Map:** es una proyecto de ámbito social y open-source. La librería incorporada permite el uso libre de información geográfica a través de mapas, imagenes ortográficas y otras fuentes. En la aplicación TeamUp se ha incluido la librería:

  ```
  osmdroid-android:6.0.0
  ```

- **Librerías Extras de Diseño:** son un conjunto de librerías que se han incluido para mejorar la visualización de la apliacación.

  - *RecyclerView:* implementa las funcionalidades para la visualización de listas de elementos de forma dinámica.
  - *CardView:* implementa el patrón de tarjeta de Material Design con esquinas redondeadas y sombras paralelas.
  - *CircleImageView:* implemanta el patrón de imagenes circulares.
  - *Glide:* implementa las funcionalidadesd de carga de imágenes y gestión de medios de código abierto rápido y eficiente para Android.

  ```
  recyclerview:1.1.0
  recyclerview-selection:1.1.0-rc01
  cardview:1.0.0
  circleimageview:3.1.0
  glide:4.11.0
  ```

Por motivos de simplificación de la implementación de cada una de las funcionalidades, se ha decidido subdividir estas en cinco paquetes.

Inicio de Sesión y Registro

Este paquete implementa las funcionalidades necesarias para el registro de nuevos usuarios y su inicio de sesión. Esta acción le permitirá acceder a las funcionalidades internas de la aplicación confirmando que el usuario esta autenticado. En la figura 5.3 se muestra un diagrama de estados que representa el registro e inicio de sesión del usuario.

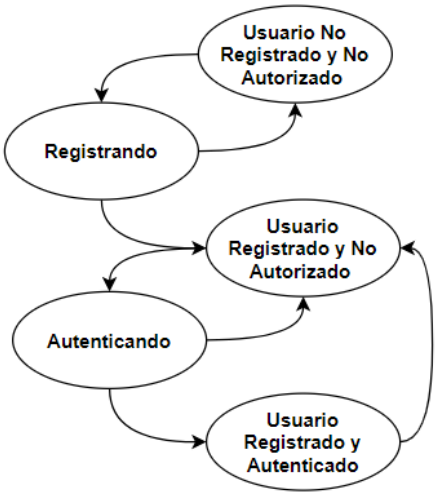


Figura 5.3: Diagrama de Estados del Paquete Login & Register

El desarrollo de este paquete consta de tres clases que implementan tres actividades de la aplicación. Estas clases emplean las pantallas 1,2 y 3 de la parte de diseño. Estas clases se albergan en un diagrama UML en la figura 5.4.

- **SplashScreenActivity:** Esta clase muestra una pantalla de portada que muestra el logo de la aplicación mientras esta se inicia.
- **LoginActivity:** Esta clase se encarga de la autorización y autenticación de los usuarios en la aplicación mediante el uso de Firebase como base de datos online para la confirmación y autenticación de los usuarios de forma correcta.
- **RegisterActivity:** Esta clase se encarga del registro de nuevos usuarios en la base de datos online Firebase para el almacenamiento de nuevos usuarios de la aplicacion de forma segura.

Principal y Utilidades

Este paquete implementa las clases necesarias para el control y la gestión de las funcionalidades predeterminadas de la aplicación. Además, algunas de estas clases simplemente son contenedoras de información y se emplean en dos o más paquetes de la aplicación.

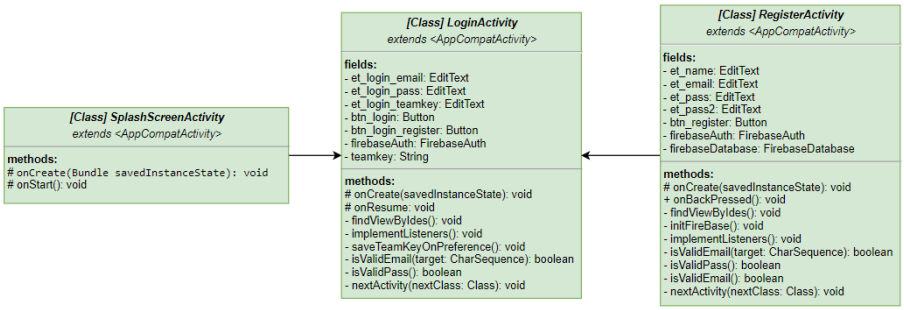


Figura 5.4: Diagrama UML del Paquete Login & Register

A diferencia del resto de paquetes, este no incorpora una visualización gráfica. Aún así, es fundamental para el correcto funcionamiento del resto de la aplicación. En la figura 5.5 se mostrará el diagrama UML y se comentarán las clases que lo incorporan:

- **MainActivity:** Esta clase es la clase principal y se encarga del control y la gestión de tres fragment con funcionalidades diferentes (escaneo de dispositivos y envío de mensajes vía BLE, chat y localización GPS).

  Nota: Un Fragment representa un comportamiento o una parte de la interfaz de usuario en una aplicación. Estos se pueden combinar y tienen su propio ciclo de vida, es decir, recibe sus propios eventos y controla sus propias variables.

- **SectionsPagerAdapter:** Esta clase se encarga de devolver un fragment correspondiente a uno de los tabs.

- **PageViewModel:** Esta clase se encarga de controlar el indice del tabs (identificativo de los fragments) mientras se utiliza la aplicacion.

- **PlaceholderFragment:** Esta clase se encarga de controlar y dar soporte a la actividad principal y controlar las vistas de los fragments (Ble, Chat y Gps)

- **User:** Esta clase es una clase contenedora que almacena la información relevante de un usuario (la informacion más delicada se almacena de forma segura en la base de datos).

- **OnBluetoothDeviceClickedListener:** Interfaz que implementa la escucha de interacciones del usuario con los fragments.

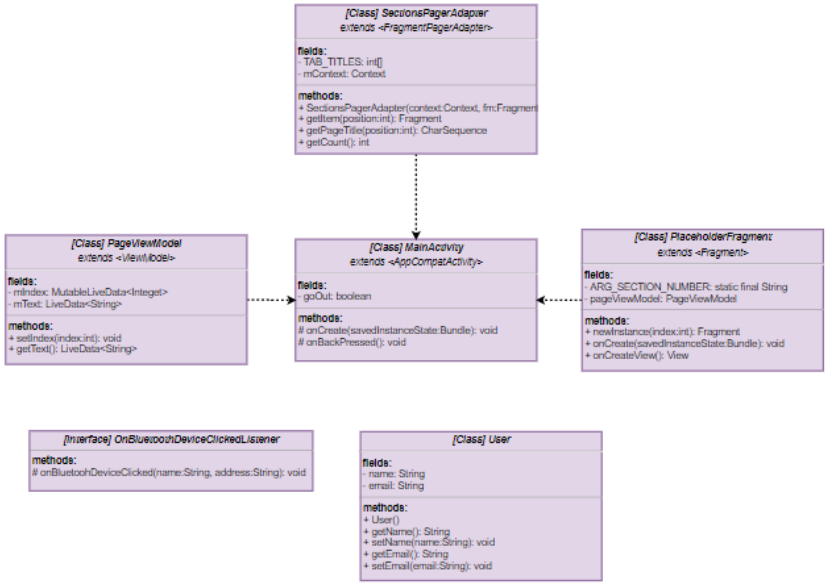


Figura 5.5: Diagrama UML del Paquete Main & Utils

Bluetooth Low Energy (BLE)

Este paquete implementa las funcionalidades necesarias para la conectividad con dispositivos BLE En la figura 5.6 se muestra el diagrama de estados del paquete BLE.. Algunas de las funcionalidades más relevantes son: escaner de dispositivos BLE en alrededores, conexión con dispositivos BLE y envío y recepción de mensajes BLE. En dicha implementación, hemos ayudado de algunas clases de la librería que se encuentra en el siguiente GitHub: `https://github.com/kakopappa/bluetooth_android_esp32_example`

El desarrollo de este paquete consta de siete clases que implementan las funcionalidades relacionadas con el diagrama de estados anterior. En la figura 5.7 se representan todas las clases mediante un diagrama UML.

- **BleFragment:** Esta clase se encarga de la funcionalidades relacionadas con la conectividad BLE. Es decir, inicia el servicio BLE con el último dispositivo BLE si este se encuentra disponible o con el dispositivo BLE indicado.

  Además, posibilita el envío de mensajes a través de este protocolo a una mota (bridge) que hará llegar la informacion a un servidor.

- **BleDeviceAdapter:** Esta clase se encarga de la adaptación de los dispositivos BLE escaneados en una lista dinámica.

- **BleMsg:** Esta clase es una clase contenedora que almacena la información de un mensaje de tipo texto para su posterior envío vía BLE.

Figura 5.6: Diagrama de Estados del Paquete BLE

- **BleSpecialMsg:** Esta clase es una clase contenedora que almacena la información de los mensajes especiales (localización GPS, registro de usuario ...) para su posterior envío vía BLE.
- **BluetoothLeService:** Esta clase extiende el servicio de Bluetooth Low Energy (BLE). Dentro de este servicio, se tienen las ejecuciones necesarias para la conectividad vía BLE.

  Nota: Un Servicio permite la ejecución de operaciones de larga duración sin interfaz de usuario. Se puede ejecutar en primer o segundo plano dependiendo del modo de enlace empleado en el servicio.
- **BluetoothLeScan:** Esta clase es el controlador del escaner de dispositivos BLE. Se encarga de inciar y parar el escaner y, de la conexión con el dispositivo BLE indicado por el usuario.
- **MyApplication:** Esta clase contenedora que almacena las varibles necesarias para caracterizar una aplicación.

A parte de estas clases, existen dos subclases (interfaces) privadas que se implementan dentro de *BleFragment*.

- **BluetoothScanCallBack:** Este interfaz nos permite el uso de un callback específico para el escaner de dispositivos BLE.
- **OnBluetoothDeviceClickedListener:** Este interfaz nos permite implementar la interacción con los dispositivos BLE de la lista dinámica.

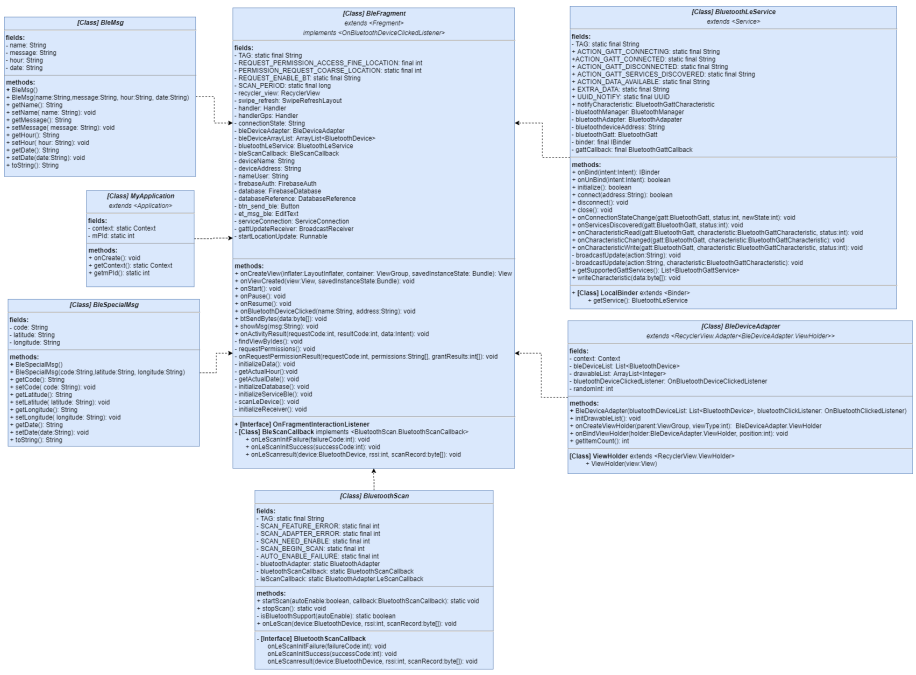


Figura 5.7: Diagrama UML del Paquete BLE

Chat

Este paquete implementa las funcionalidades de chat. El chat creado es grupal y depende principalmente de la clave de grupo (*Team key*). Por defecto, si la clave se encuentra vacía, entonces, la clave es *Chat*. En caso contrario, se genera un nodo de almacenamiento en la base de datos con la clave dada. Esta clave debe ser conocida para todos los miembros del equipo.

En este paquete se visualizarán los mensajes enviados por los diferentes miembros del equipo de emergencias y permitirá la posibilidad de envío de estos mensajes. Para la implementación se han empleado cuatro clases que se describen a continuación y se muestran en la figura 5.8.

- **ChatFragment:** Esta clase se encarga de la visualizacion y control de la actividad Chat, es decir, la visualización de los mensajes listados y el envío de mensajes entre los miembros del equipo.

- **HolderMsg:** Esta clase es una clase contenedora que almacena el soporte de elementos de los mensajes del chat.

- **AdapterMsg:** Esta clase se encarga de la adaptación de los mensajes a un recyclerView o lista.

- **Msg:** Esta clase es una clase contenedora que almacena la información de un mensaje (nombre del emisor, mensaje, fecha y hora ...).

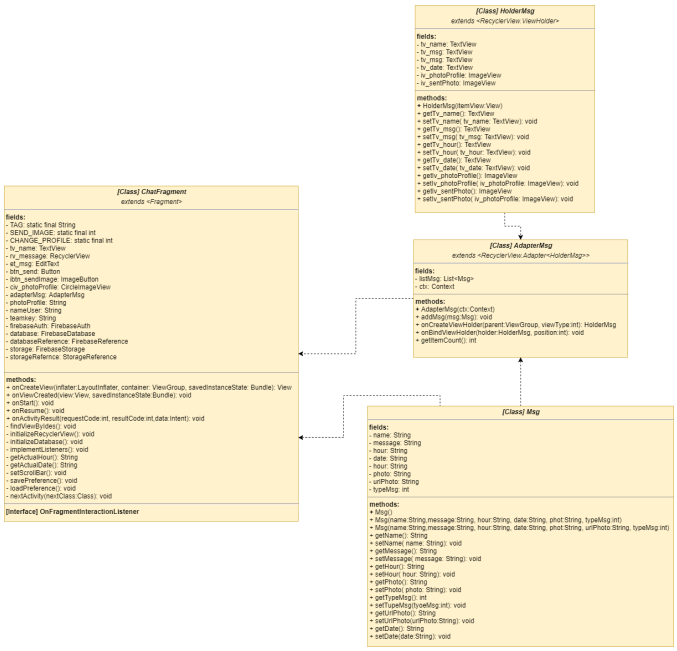


Figura 5.8: Diagrama UML del Paquete Chat

GPS

Este paquete implementa las funcionalidades relacionadas con la localización GPS del usuario. La localización GPS del usuario se obtiene directamente del provider una vez pasado un tiempo determinado o, en caso, de superar una distancia mínima con respecto a la última tomada.

El desarrollo de este paquete consiste en la visualización de las coordenadas (longitud y latitud) y de la localización del usuario en un mapa Open Street Map (OSM). Además, en caso de haber activado la localización antes de conectarnos al dispositivo BLE, entonces se envía a los servidores la localización GPS cada período aproximado de 3 minutos. Estas funcionalidades se muestran en la figura 5.9 y se han implementado sobre las siguientes clases:

- **GpsFragment:** Esta clase se encarga de iniciar el servicio de localización. Además, controla la visualización de las coordenadas del usuario y control del posicionamiento del ususario sobre el mapa OSM.
- **GpsService:** Esta clase extiende un servicio GPS. Dentro de este servicio se implementa el *LocationListener* que se encarga de actualizar la localización del usuario y sus respectivas coordenadas.

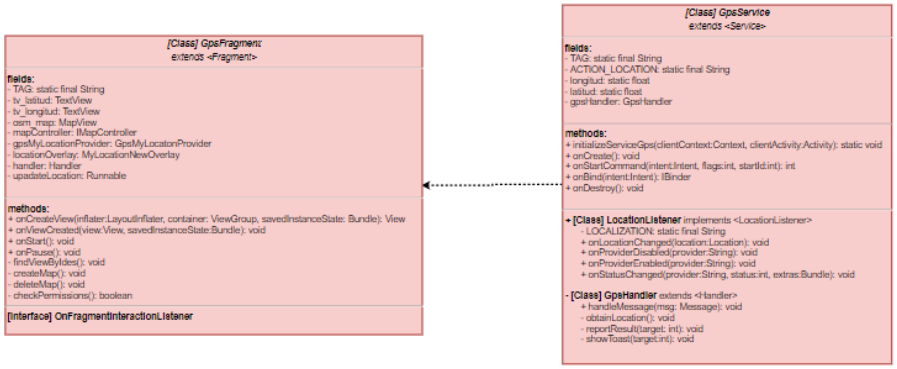


Figura 5.9: Diagrama UML del Paquete GPS

En la figura 5.10 se representa el diagrama completo de la aplicación móvil desarrollada.

Figura 5.10: Diagrama UML App TeamUp

5.3. Implementación y Configuración de las Motas

Las motas son elementos que realizan la función de bridge, es decir, hacen el cambio de tecnología entre BLE y LoRa. De este modo, los mensajes transmitidos por los usuarios pueden llegar a su destino. Por motivos económicos y de sencillez se han empleado dispositivos ESP32 de Espressif. Estos dispositivos se han elegido debido a su bajo coste económico y energético y, además, por incorporar las dos tecnologías necesarias para la labor. Cada miembro del equipo de rescate debe llevar una mota consigo para ampliar el rango de cobertura LoRaWAN.

El desarrollo del software interno se ha realizado con el IDE de Arduino y el lenguaje de programación C++. Debemos tener en cuenta que el dispositivo empleado tiene capacidad energética y de almacenamiento reducida, por tanto, la implementación deberá ser sencilla y funcional. A continuación, se muestran en la figura 5.11 las configuraciones más relevantes para el correcto funcionamiento del dispositivo.

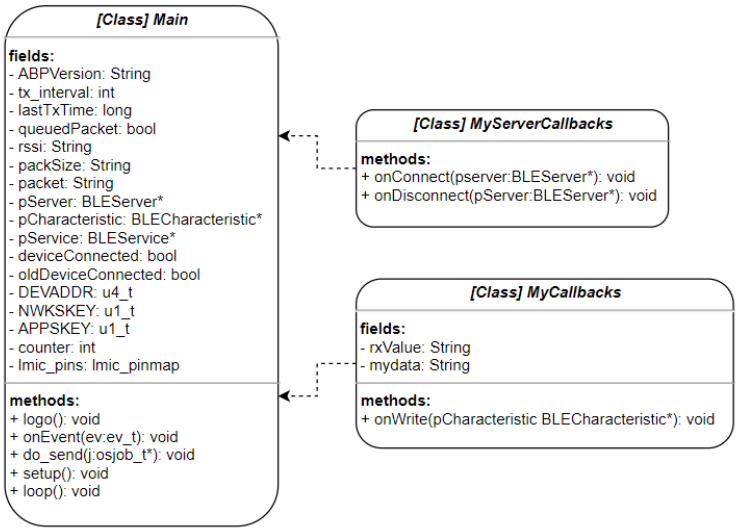


Figura 5.11: Diagrama UML de la Mota

- **MyServerCallbacks:** Esta clase implementa el callback propio del servidor BLE. Este callback se encarga de la conexión y desconexión del servidor con dispositivos BLE que actúan como clientes de la conexión.

- **MyCallbacks:** Esta clase implementa la funcionalidad de recepción de mensajes vía BLE y retransmisión en LoRa. Esta clase obtiene la información a través de las características BLE. Posteriormente, almacena esta información y la retransmite vía LoRa mediante la librería *LMIC* o *MCCI*.

- **Main:** Esta clase se encarga del control y gestión de la mota. El procedimiento seguido consiste en:

    1. Comprobación y selección de la librería de transmisión LoRa. Estas librerías varían la configuración según la versión empleada.
    2. Configuración de la placa para la transmisión LoRa mediante las variables NwkSKey, AppSKey y DevEUI. Estas modificaciones son necesarias para la recepción de los mensajes LoRa en los servidores y para identificar el dispositivo emisor.
    3. Configuración del callback BLE. Este se configura a través de tres elementos (characteristic, server y service). A estos se le asignan una serie de permisos, que en nuestro caso son: *UUID*, *READ*, *WRITE*, *NOTIFY* y *INDICATE*.

Como se ha comentado, las motas al inicio del desarrollo de la red se implementaron sobre dispositivos ESP32. Pero estos dispositivos están muy limitados en capacidad, es decir, únicamente se pueden emplear como dispositivos de clase A en LoRa. Esto se traduce en que el envío y recepción de las tramas LoRa sufren de una latencia considerable debido al retardo por la limitación del ciclo de trabajo.

Por este motivo, posteriormente, se ha realizado una implementación de las motas sobre un tipo de dispositivo que alberga la posibilidad de emplear la clase C, es decir, los dispositivos permanecen en estado de escucha constantemente, mejorando el tiempo de interacción entre ususario a costa de empeorar el coste energético. Los dispositivos empleados en este caso son *Pycom LoPy 4.0* que incorpora las tecnologías LoRa, sigfox, Wi-Fi ... El software empleado mantiene la misma funcionalidad que en el caso anterior, pero se ha desarrollado utilizando Visual Studio Code y PlatformIO como herramientas y el lenguaje Python, basándonos en las referencia y APIs ofrecidas por la comunidad *Pycom*, tanto para la parte BLE [53] y la parte LoRa [54].

5.4. Implementación y Configuración del Gateway

Las pasarelas o gateways son elementos fundamentales en el funcionamiento de la red implementada, dado que son nodos intermedios entre los servidores y los usuarios. El gateway se encarga de la recepción de los mensajes LoRa provinientes de las motas y lo retransmite mediante Wi-Fi al nodo central.

En el desarrollo del proyecto se ha optado por la implementación y configuración del gateway dependiendo de tres variables: coste económico y características LoRa (número de canales y tipo de dispositivo empleado). A continuación, se describirán las implementaciones para cada uno de los dispositivos.

5.4.1. IMST Gateway Lite

Este tipo de gateway fue la primera en implementar debido a su disposición en el laboratorio y las altas capacidades que incorpora al barrer los ocho canales disponible en LoRa. La implementación y configuración del gateway se realiza sobre las Raspberry Pi 1 que incorpora en el interior mediante una conexión SSH.

Primero se debe instalar la pasarela para el envío de datos al LoRa Gateway Bridge. Posteriormente, se debe modificar la dirección Internet Protocol (IP), nombre del host del LoRa Gateway Bridge y el puerto (1700 por defecto). Estas configuraciones se realizan en el fichero de configuración.

```
1 " gateway_conf ": {
2     " gateway_ID ": " C86000FFFE6A8EA9 ",
3     " servers ": [ { " server_address ": "192.168.100.101" , "
          serv_port_up ": 1700 , " serv_port_down ": 1700 , "
          serv_enabled ": true }, { "s$
4     " ref_latitude ": 38.03403 ,
5     " ref_longitude ": -4.08682 ,
6     " ref_altitude ": 212 ,
7     " contact_email ": " usuario@correo.ugr.es ",
8     " description ": " LORAGW01 "
9 }
```

Una vez configurado, solamente se debe arrancar y establecerse para que se arranque al iniciar el equipo.

```
>> sudo systemctl start chirpstack-gateway-bridge
>> sudo systemctl enable chirpstack-gateway-bridge
```

5.4.2. ESP32 - Single Channel Gateway

Debido a motivos varios como el coste económico y el aislamiento social que nos impedía ir al laboratorio a seguir trabajando con la explicada anteriormente, se decidió implementar una pasarela monocanal sobre los dispositivos disponibles, que en nuestra situación eran las placas ESP32 y un host (Raspberry Pi 4). El funcionamiento del gateway se sigue manteniendo excepto porque este modo únicamente emplea el primer canal para emitir y recibir mensajes LoRa.

La implementación de esta pasarela ha sido empleando una librería desarrollada por Marteen Westenberg que tiene publicado en su repositorio GitHub `https://github.com/kersing/ESP-1ch-Gateway-v5.0`. Esta librería está formada por una serie de archivos de configuración predefinidos.

A continuación se explican los procedimientos de configuración más relevantes. Para ello, abrimos el archivo de configuración *ESP-sc-gway.h* que nos permite la configuración de los siguientes aspectos:

- **Servidor Chirpstack:** por defecto, la configuración del servidor está indicando un servidor The Things Network (TTN). En nuestro caso, se desea enlazar con un servidor Chirpstack que tendremos configurado en un host conocido.
- **Definición Gateway:** se definene las características que identifican a nuestro gateway (descripción, email, plataforma de desarrollo y coordenadas predeterminadas).
- **Spreading Factor:** es un parámetro importante a la hora de establecer un único canal. Este parámetro nos especifica la velocidad de datos con la que es posible la comunicación vía LoRa, la potencia de transmisión, la subfrecuencia y el tiempo en el aire. Nosotros empleamos un spreading factor de 7.
- **Detección de Actividad del Canal:** se implementa esta función que busca las cabeceras válidas de LoRa y determina el spreading factor en consecuencia. Esto permite que la pasarela sea más versátil y emplee un modo de escucha continuada.
- **Canal de Escucha:** identifica la frecuencia del canal y si spreading factor empleada por la pasarela. Se ha definido como canal el primero de los posibles, por tanto, la pasarela únicamente transmite y recibe de los nodos que funcionen en este canal.
- **Servidor NTP:** identifica al servidor ntp, el país, región y la zona horaria donde se encuentra la pasarela.

- **Características Gateway:** se especifican los valores necesarios para el correcto cifrado y transmisión entre el gateway y el servidor Chirpstack (NwkSKey, AppSKey y DevAddr).
- **Conectividad Wi-Fi:** se indican los nombres y contraseñas de nuestras conexiones Wi-Fi para que se pueda enlazar con los servidores. Además, se debe de dejar una primera línea vacía que se reserva para el gestor Wi-Fi.

Comentados todos los aspectos modificados en la configuración de la pasarela monocanal, ilustramos los cambios en el archivo.

```
#define _TTNPORT 1700
#define _TTNSERVER "192.168.0.25"

#define _DESCRIPTION "My ESP32 Gateway 868.1MHz SF7"
#define _EMAIL "e.felixdelgado@go.ugr.com"
#define _PLATFORM "ESP32"
#define _LAT 38.03403
#define _LON -4.08682
#define _ALT 212

#define _SPREADING SF7
#define _CAD 1
#define _STRICT_1CH     1

#define NTP_TIMESERVER "nl.pool.ntp.org"
#define NTP_TIMEZONES   1

#define _DEVADDR { 0x26, 0x00, 0x00 0x00 }
#define _APPSKEY { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0
    x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }
#define _NWKSKEY { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0
    x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }

wpas wpa[] = {
  { "" , "" },
  { "WifiTeamUp", "teamupkey" }
  { "vodafoneD778", "*********" }
};
```

Estando configurado el gateway, simplemente debemos cargar el código y lanzarlo. Si visualizamos la figura 5.12 que representa el monitor serie del IDE de Arduino, podemos ver como realiza la puesta en marcha.

```
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1216
ho 0 tail 12 room 4
load:0x40078000,len:10864
load:0x40080400,len:6432
entry 0x400806b8
SPIFFS loaded success
Assert=Do Asserts
debug=1
readConfig:: Starting
SSID=vodafoneD778
CH=0
SF  =7
FCNT=0
DEBUG=0
CAD=1
HOP=0
NODE=0
BOOTS=153
RESETS=0
WIFIS=161
VIEWS=7
REFR=1
REENTS=0
NTPETIM=15010
NTPERR=10
NTPS=61
MAC: 00:7d:00:b9:00:e4, len=17
0:1. WiFi connect to: vodafoneD778
WLAN reconnected
Host esp32-b986e4 WiFi Connected to vodafoneD778
Gateway ID: 807D3AFFFFB986E4, Listening at SF7 on 868.10 Mhz.
setupTime:: Time not set (yet)
Time: Friday 12:05:04
Gateway configuration saved
WWW Server started on port 80
IP :192.168.0.27
```

Figura 5.12: Puesta en marcha del Gateway ESP32 Monocanal

Una vez lanzado, podemos comprobar su estado a través del servidor web que implementa. Accediendo a el a través de un navegador y su dirección IP. En este interfaz se diferencian cinco apartados: estadísticas de paquetes, historial de mensajes, estado del sistema, configuración Wi-Fi y configuración de la pasarela. Esta interfaz podemos verla en la figura 5.13.

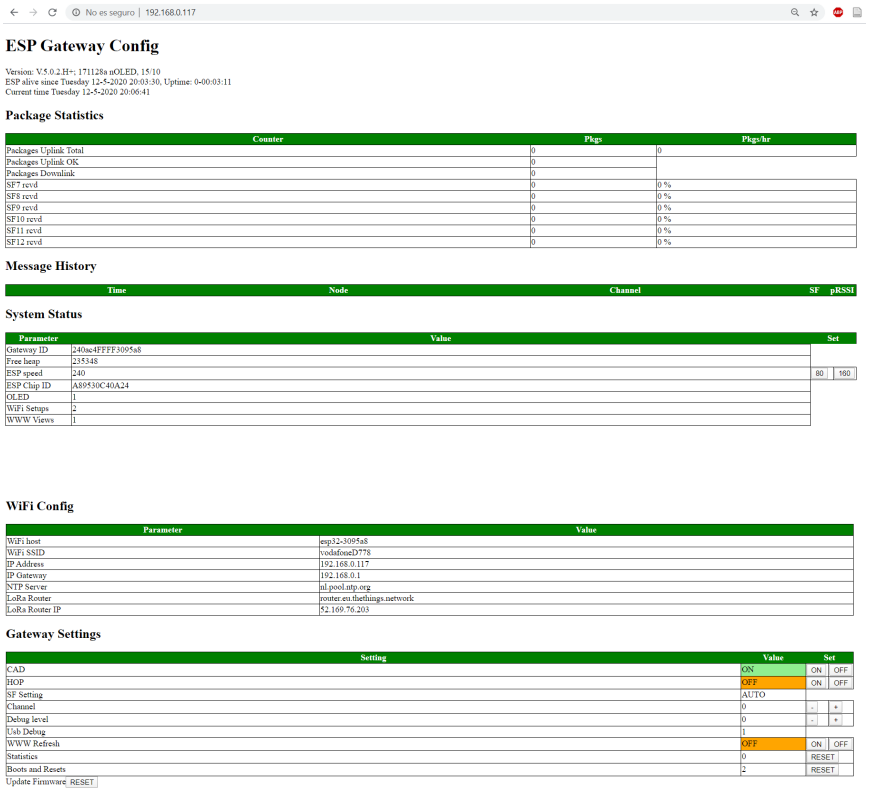


Figura 5.13: Interfaz Web Gateway ESP32 Monocanal

5.4.3. LoPy 4.0 - Nano Gateway

Por el mismo motivo comentado en las motas, se ha decidido realizar una implementación funcional sobre el dispositivo *LoPy 4.0* de *Pycom*. Al igual que en caso de las motas se ha empleado Visual Studio Code y PlatformIO para el desarrollo y se ha empleado ayuda de las referencias ofrecidas por la comunidad *Pycom* [55]. El desarrollo de la pasarela consta de tres archivos:

- **main.py:** se encarga de la puesta en marcha y la llamada a las librerías y archivos de configuración necesarios para iniciar la pasarela.
- **config.py:** se encarga de las configuraciones del servidor y la red a la que se encuentra conectada la pasarela. Especifica nuestro servidor Chirpstack, la frecuencia empleada en la comunicación LoRa, el nombre y contraseña de la red Wi-Fi y se genera el identificador del gateway.
- **nanogateway.py:** se encarga de controlar la generación y el reenvío de paquetes de datos LoRa.

Comprobamos el funcionamiento de la nueva pasarela y observamos como se realiza la puesta en marcha y obtenemos que esta responde correctamente, como se muestra en la figura 5.14.

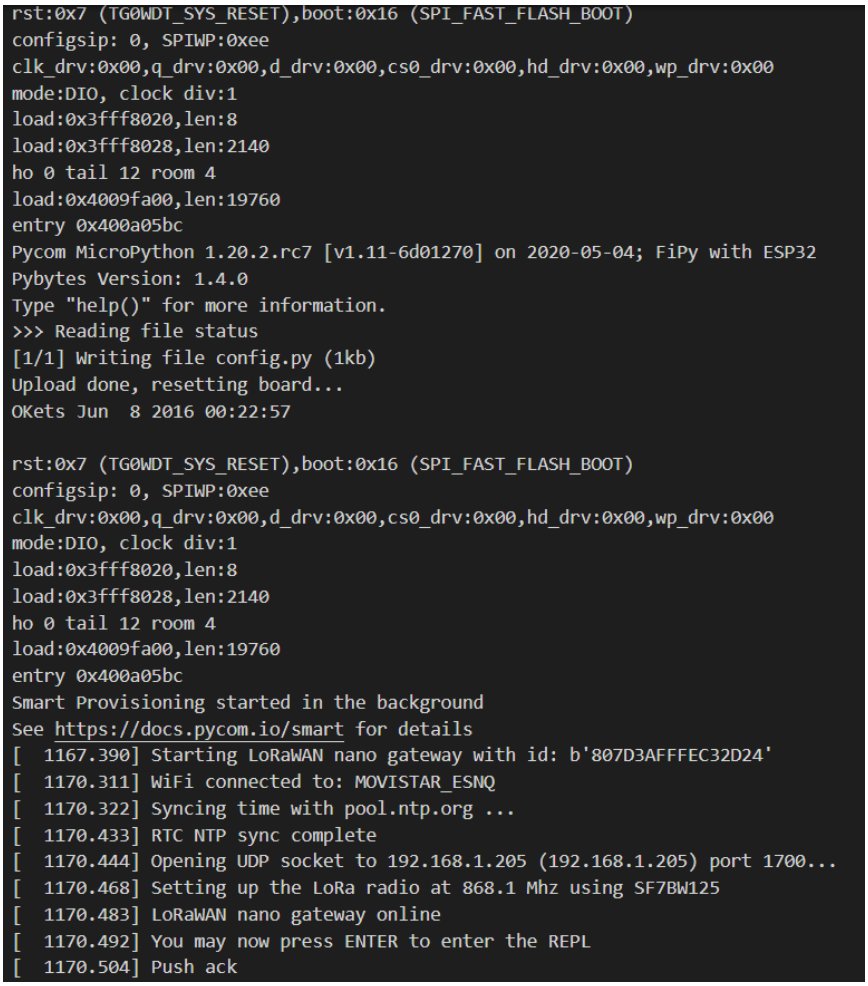


Figura 5.14: Nano Gateway LoPy 4.0

5.5. Implementación y Configuración de los Servidores Chirpstack

Los servidores Chirpstack son parte del núcleo de la red y se incorporan en la Raspberry Pi 1 (caso de uso de IMST Gateway) o Raspberry Pi 4 (otros gateways). Los servidores son fundamentales en el funcionamiento de la red TeamUp dado que se encargan de la recepción de mensajes, desencriptado y almacenamiento de los mensajes. La seguridad implementada sobre la red LoRaWAN se muestra en la figura 5.15.

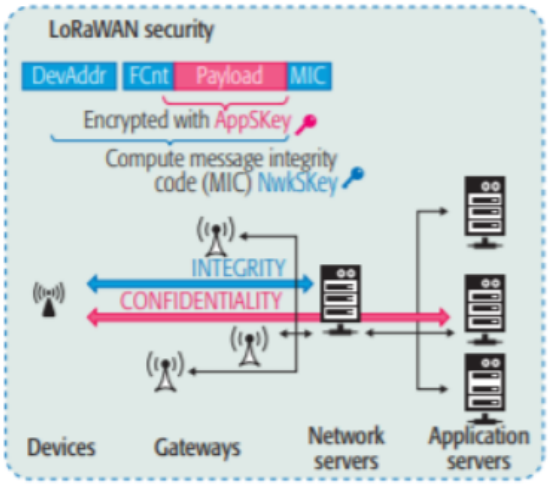


Figura 5.15: Seguridad de los Servidores Chirpstack [42]

Los red implementada sobre los servidores Chirpstack mantiene una seguridad basada en confidencialidad e integridad. Los mensajes están cifrados desde las motas a los servidores de aplicación y se mantiene la integridad de estos desde las motas a los servidores de red.

La instalación y configuración de los servidores Chirpstack se explican en los siguientes pasos:

1. **Instalación de los Servidores Chirpstack:** es necesario la descarga de los paquetes relacionados con los prerrequisitos (MQTT broker, Redis y PostgreSQL). Después se descargan los paquetes de los servidores y se configuran para su correcto funcionamiento.

   Una vez instalado y lanzado, podemos acceder a su interfaz web a través de la dirección IP y el puerto 8080.

2. **Creación de la Organización y Perfil de Servicio :** es necesario crear una organización dentro de la estructura Chirpstack. Esta organización incluye distintos perfiles de servicio. Estos incorporan al servidor de red y englobarán a todos los dispositivos, independientemente sean pasarelas o motas.

3. **Creación de la Aplicación Chirpstack:** la aplicación define los dispositivos finales dentro de la estructura.

4. **Creación del Perfil del Dispositivo:** se definene las propiedades que pertenecen a la organización, e.g. tipo de activación, versión LoRaWAN, etc.

5. **Configuración de Dispositivos:** los dispositivos se asocian a una aplicación. En nuestro caso, se han configurado motas que emplean ABP como tipo de activación por lo que se necesitan las claves de sesión NwkSKey y AppSKey y, el identificador del dispositivo DevAddr.

Si se ha configurado correctamente el gateway y los dispositivos, cuando estos se ponen en marcha se puede comprobar si los dispositivos se encuentran funcionando correctamente. Este funcionamiento lo podemos visualizar en las figuras 5.16 y 5.17.

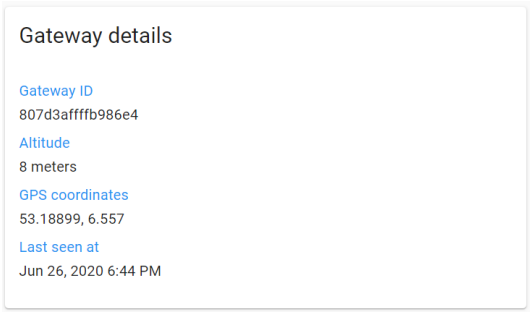


Figura 5.16: Comprobación de Gateway

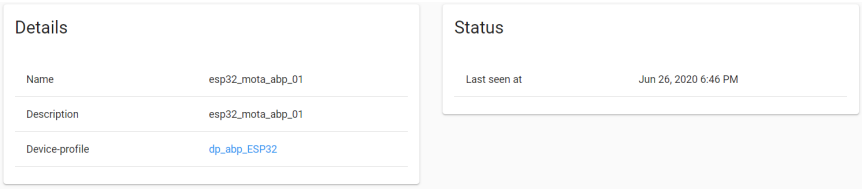


Figura 5.17: Comprobación de la Mota

Si se desea saber de forma más exhaustiva el procedimiento realizado para la instalación y puesta en marcha de los servidores Chirpstack véase el **Anexo B**.

5.6. Implementación y Configuración del Servidor Web

El servidor web se encarga del almacenamiento de los mensajes y usuarios propios de la red TeamUp. Para ello, se implementa de forma que se hace un traspaso de información desde la base de datos propia de Chirpstack a una base de datos privada e implementada sobre PostgreSQL. Posteriormente, si se recibe alguna solicitud desde un usuario a través de la interfaz web, se responde enviando la información correspondiente.

La implementación de las funcionalidades principales que un administrador puede realizar sobre el servidor se han desarrollado en *bash* en distintos archivos que se comentan a continuación:

- **create_table.sh:** este archivo permite al administrador crear la base de datos con una serie de tablas donde se almacenará la información.
- **register_user.sh y register_user_now.sh:** estos archivos permiten al administrador registrar nuevos usuarios dentro del sistema.
- **delete_user.sh:** este archivo permite al administrador borrar usuarios del sistema.
- **show_users.sh:** este archivo permite al administrador visualizar en el terminal los usuarios registrados hasta el momento.
- **show_latest_user_position.sh:** este archivo permite al administrador visualizar en el terminal la última posición de un usuario indicado.

A parte de estas funcionalidades implementadas para el administrador, el servidor permite realizar algunas funciones adicionales. Estas se pueden visualizar a través del terminal (véase figura 5.18) con ayuda del comando:

```
>> python3 server.py --help
```

```
usage: server.py [-h] [--config FILENAME] [--sendtoall MESSAGETOALL]
                 [--todevice DEVEUI] [--touser USER] [--message MESSAGE]

optional arguments:
  -h, --help            show this help message and exit
  --config FILENAME     configuration file
  --sendtoall MESSAGETOALL
                        send message to all connected devices
  --todevice DEVEUI     destination device, --message required
  --touser USER         destination user, --message required
  --message MESSAGE     destination device, --todevice or --touser required
```

Figura 5.18: Funcionalidades Adicionales del Administrador

El servidor se ha implementado basandose en Python. Para ello, se han instalado una serie de librerías adicionales:

- **lighttdp:** esta librería permite la implementación del servidor de forma rápida.
- **php7.3:** este conjunto de librerías son necesarias para la recepción de las solicitudes por parte del navegador web del usuario final.
- **psycopg2:** esta librería permite la conexión con nuestra base de datos Postgres.
- **pycurl:** esta librería permite el uso de Curl, es decir, se encarga de poder usar la API REST del servidor Chirpstack.
- **paho.mqtt.client:** esta librería permite la conexión al servidor MQTT y poder visualizar los mensajes LoRaWAN recibidos. En principio, la API no está pensada para poder ver dichos mensajes.
- **argparse y configparse:** estas librerías se encargan de parsear los parámetros introducidos por línea de comandos.
- **json y base64:** estas librerías permiten la lectura y escritura de archivos con formatos específicos.

La instalación y puesta en marcha del servidor es simple partiendo de tener instalados todos los prerrequisitos. En este momento, simplemente nos falta modificar el archivo de configuración *15-fastcgi-php.conf*.

```
fastcgi.server += ( ".php" =>
        ((
                "socket" => "/var/run/php/php7.3-fpm.sock",
                "broken-scriptfilename" => "enable"
        ))
)
```

El fichero de configuración del servidor *config.ini* incorpora:

- Datos para el uso de la API REST de Chirpstack y el servidor MQTT.
- Datos para la conexión con nuestra base de datos Postgres.

Teniendo el sistema listo, es posible lanzar el servidor (*server.py*). Este servidor implementa métodos relacionados con diversos aspectos:

- **Base de Datos**
  - *openDatabase():* permite abrir las bases de datos, es decir, la base de datos propia y la base de datos de Chirpstack.

- *closeDatabase():* permite cerrar las bases de datos, es decir, la base de datos propia y la base de datos de Chirpstack.
- *getLoggedDevices():* devuelve una lista con los dispositivos que se encuentran logeados en nuestra base de datos.
- *getLoggedUserFromDevice():* devuelve el usuario cuyo dispositivo corresponda con el indicado.
- *getDeviceFromLoggedUser():* devuelve el dispositivo enlazado con un usuario registrado en nuestra base de datos.
- *updateLoggedUser():* se encarga de actualizar la información de un usuario registrado.
- *updateTimeLastSeen():* se encarga de actualizar la última vez que un usuario fue visto por el sistema.
- *updateUserPosition():* se encarga de actualizar la localización de un usuario.
- *insertMQTTMessageOntoDatabase():* se encarga de almancenar los mensajes extraidos mediante MQTT en nuestra base de datos.

- **Chirpstack**

  - *getConnectedDevices():* devuelve la información relacionada con los dispositivos que se encuentran conecetados a Chirpstack.
  - *sendMessageLoRaWAN():* permite el envío de mensajes LoRaWAN a un dispositivo indicado.
  - *sendMessageLoRaWANToAllDevices():* permite el envío de mensajes a todos los dispositivos que se encuentran registrados y conectados a Chirpstack.

- **MQTT**

  - *on_connect():* realiza la conexión de nuestro servidor a la base de datos Chirpstack mediante MQTT.
  - *on_message():* este método alberga la mayor parte de la funcionalidad lógica de nuestro servidor y se ha decidido que la forma más práctica de comprender este funcionamiento es mediante un diagrama de flujo (véase figura 5.19). Aunque el principal cometido del método es la recepción de mensajes MQTT y su posterior almacenamiento.

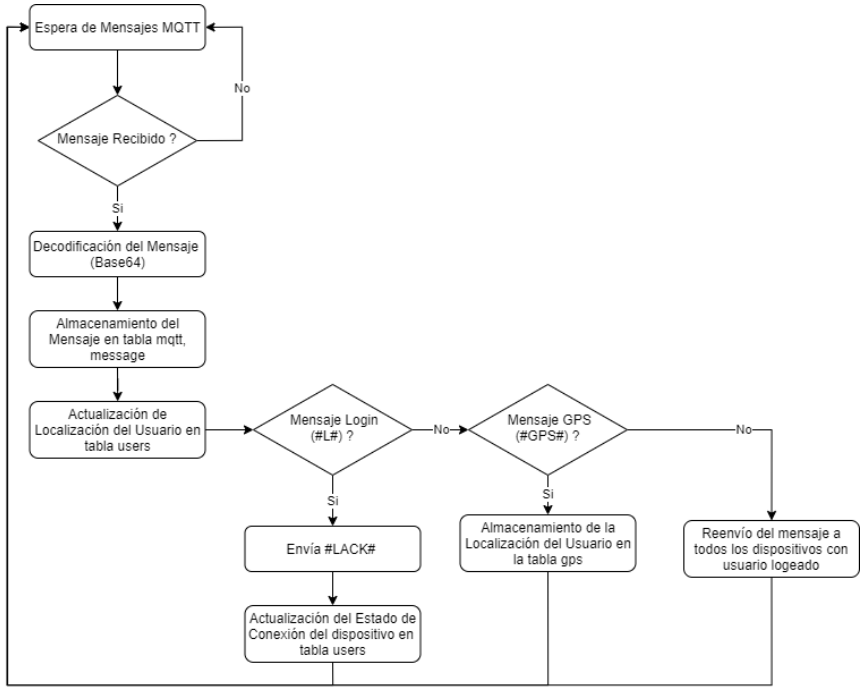


Figura 5.19: Diagrama de Flujo del Servidor

La ejecución de nuestro servidor es muy simple y lineal. Este se queda esperando a recibir mensajes MQTT y actúa en función al tipo de mensaje recibido, actualizando la base de datos o enviando algún mensajes si así se precisa.

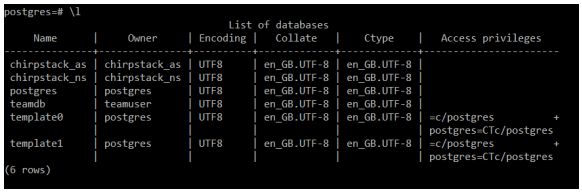
Para acabar, podemos ver la base de datos propia a través del terminal. Para ello, debemos entrar a la base de datos con el siguiente comando:

```
>> sudo -u postgres psql
```

En este punto, se pueden realizar varias operaciones como:

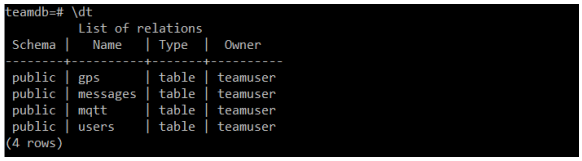
- **Listar bases de datos *(l)*** (véase figura 5.20)
- **Entrar en una base de datos concreta *(c teamdb)***
- **Listar relaciones entre tablas *(dt)*** (véase figura 5.21)
- **Mostrar tablas** (véase figuras 5.22, 5.23 y 5.24)

```
select * from [ users | gps | messages ];
```



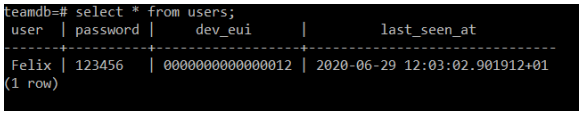
```
postgres=# \l
                                  List of databases
     Name      |     Owner     | Encoding |   Collate   |    Ctype    |   Access privileges
---------------+---------------+----------+-------------+-------------+-----------------------
 chirpstack_as | chirpstack_as | UTF8     | en_GB.UTF-8 | en_GB.UTF-8 |
 chirpstack_ns | chirpstack_ns | UTF8     | en_GB.UTF-8 | en_GB.UTF-8 |
 postgres      | postgres      | UTF8     | en_GB.UTF-8 | en_GB.UTF-8 |
 teamdb        | teamuser      | UTF8     | en_GB.UTF-8 | en_GB.UTF-8 |
 template0     | postgres      | UTF8     | en_GB.UTF-8 | en_GB.UTF-8 | =c/postgres          +
               |               |          |             |             | postgres=CTc/postgres
 template1     | postgres      | UTF8     | en_GB.UTF-8 | en_GB.UTF-8 | =c/postgres          +
               |               |          |             |             | postgres=CTc/postgres
(6 rows)
```

Figura 5.20: Listado de Base de Datos



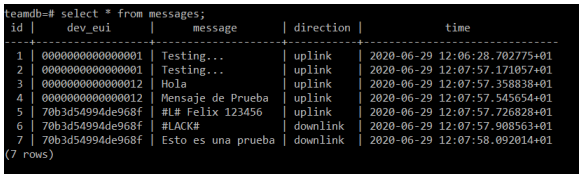
```
teamdb=# \dt
          List of relations
 Schema |   Name   | Type  |  Owner
--------+----------+-------+----------
 public | gps      | table | teamuser
 public | messages | table | teamuser
 public | mqtt     | table | teamuser
 public | users    | table | teamuser
(4 rows)
```

Figura 5.21: Listado de Tablas



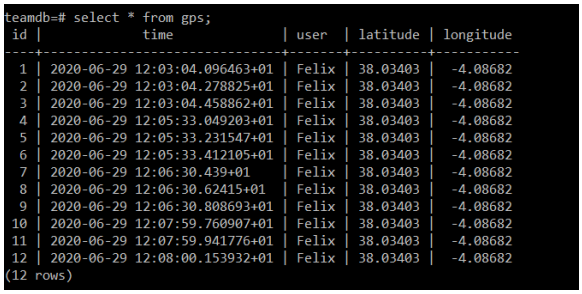
```
teamdb=# select * from users;
 user  | password |     dev_eui      |         last_seen_at
-------+----------+------------------+------------------------------
 Felix | 123456   | 0000000000000012 | 2020-06-29 12:03:02.901912+01
(1 row)
```

Figura 5.22: Tabla Users



```
teamdb=# select * from messages;
 id |     dev_eui      |      message       | direction |           time
----+------------------+--------------------+-----------+-------------------------------
  1 | 0000000000000001 | Testing...         | uplink    | 2020-06-29 12:06:28.702775+01
  2 | 0000000000000001 | Testing...         | uplink    | 2020-06-29 12:07:57.171057+01
  3 | 0000000000000012 | Hola               | uplink    | 2020-06-29 12:07:57.358838+01
  4 | 0000000000000012 | Mensaje de Prueba  | uplink    | 2020-06-29 12:07:57.545654+01
  5 | 70b3d54994de968f | #L# Felix 123456   | uplink    | 2020-06-29 12:07:57.726828+01
  6 | 70b3d54994de968f | #LACK#             | downlink  | 2020-06-29 12:07:57.908563+01
  7 | 70b3d54994de968f | Esto es una prueba | downlink  | 2020-06-29 12:07:58.092014+01
(7 rows)
```

Figura 5.23: Tabla de Mensajes



```
teamdb=# select * from gps;
 id |             time              | user  | latitude | longitude
----+-------------------------------+-------+----------+-----------
  1 | 2020-06-29 12:03:04.096463+01 | Felix | 38.03403 |  -4.08682
  2 | 2020-06-29 12:03:04.278825+01 | Felix | 38.03403 |  -4.08682
  3 | 2020-06-29 12:03:04.458862+01 | Felix | 38.03403 |  -4.08682
  4 | 2020-06-29 12:05:33.049203+01 | Felix | 38.03403 |  -4.08682
  5 | 2020-06-29 12:05:33.231547+01 | Felix | 38.03403 |  -4.08682
  6 | 2020-06-29 12:05:33.412105+01 | Felix | 38.03403 |  -4.08682
  7 | 2020-06-29 12:06:30.439+01    | Felix | 38.03403 |  -4.08682
  8 | 2020-06-29 12:06:30.62415+01  | Felix | 38.03403 |  -4.08682
  9 | 2020-06-29 12:06:30.808693+01 | Felix | 38.03403 |  -4.08682
 10 | 2020-06-29 12:07:59.760907+01 | Felix | 38.03403 |  -4.08682
 11 | 2020-06-29 12:07:59.941776+01 | Felix | 38.03403 |  -4.08682
 12 | 2020-06-29 12:08:00.153932+01 | Felix | 38.03403 |  -4.08682
(12 rows)
```

Figura 5.24: Tabla de Localizaciones GPS

5.7. Implementación de la Interfaz Web

La interfaz web es una simple visualización de nuestra base de datos Postgres en un navegador. Esta interfaz solicitará información a través de la solicitudes o *queries PHP*. Esta implementación consta de varios archivos:

- **index.php:** presenta la página inicial con el título. Contiene una descripción e incluye un fichero *template_menu.php* que es genérico para todas las páginas.
- **template_menu.php:** este archivo se encarga de generar los iconos y los enlaces del menú superior. Este menú contiene una serie de funcionalidades que comentamos a conntinuación:
  - *Inicio:* muestra únicamente la pantalla principal y el menú.
  - *Registrar usuario:* implementa un formulario (*register_user.php*). Este archivo llama a *register_done.php* cuando se pulsa el botón, pasando el usuario y la clave. Para ello, emplea la librería de Postgres para *PHP* [56]. Además, se encarga de borrar el usuario en caso de que exista y lo añade a la base de datos (*users*).
  - *Borrar usuario:* implementa un formulario (*delete_user.php*) que llama a *delete_done.php* cuando se pulsa el botón. En este momento, se establece una conexión con la base de datos y borra al usuario de la tabla *users*.
  - *Ver usuarios:* en esta página se llama al fichero *db.php*, pasándole el parámetro que hace referencia a la tabla *users*. Este proceso permite la visualización de los usuarios y la información almacenada en la tabla indicada. Esta información se formatea a HTML para la visualización web.
  - *Ver mensajes:* en esta página se llama al fichero *db.php*, pasándole el parámetro que hace referencia a la tabla *messages*. Este proceso permite la visualización de los mensajes almacenados en nuestra base de datos. Esta información se formatea a HTML para la visualización web.
  - *Ver ubicación de usuarios:* en esta página se llama al fichero *db.php*, pasándole el parámetro que hace referencia a la tabla *gps*. Este proceso permite la visualización de las localizaciones GPS de los usuarios logeados en formato de tabla HTML.
  - *Ver ubicación de un usuario en Google Maps:* implementa un formulario (*location_user.php*) que llama a *location_done.php* indicando el nombre del usuario. En este momento, se establece una conexión con la base de datos y busca las últimas coordenadas de dicho usuario en la tabla *gps*.

- *Mandar mensaje a usuario:* en esta página se implementa un formulario (*send_message_to_user.php*) que llama a un archivo llamado *send_message_to_user_done.php*. En este momento se indica el usuario y el mensaje. Posteriormente, se estable la conexión con la base de datos donde se comprueba cúal es el dispositivo asociado al usuario indicado. En caso de estar activo, se envía un mensaje a dicho usuario mediante la API REST.
- *Mandar mensaje a todos los usuarios:* implementa un formulario (*send_message_to_all_users.php*) que llama un archivo llamado *send_message_to_all_users_done.php*. En este momento, se indica el mensaje a transmitir y, al igual que el anterior, envía el mensaje a todos los usuarios activos.

En la figura 5.25 se muestra la pantalla de inicio de la interfaz web.

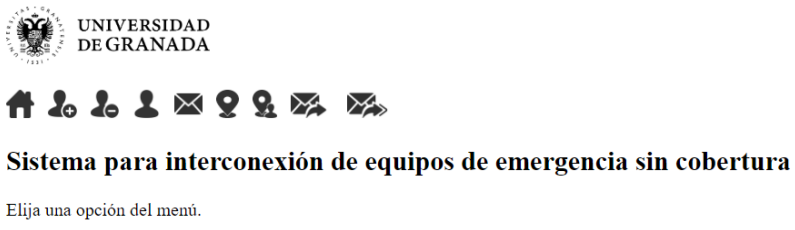


Figura 5.25: Pantalla de Inicio de la Interfaz Web

Capítulo 6

Pruebas

En este capítulo se exponen las pruebas que se han realizado para la comprobación del adecuado funcionamiento de nuestra red TeamUp y el análisis del estado de la misma.

Las pruebas realizadas consisten en la comprobación de conectividad entre dispositivos empleando BLE y LoRaWAN, comprobando a su vez el alcance que puede cubrir esta última tecnología de comunicación. Posteriormente, se realizan las primeras pruebas sobre la aplicación móvil desarrollada y las pruebas de verificación de envío y recepción de mensajes entre los dispositivos que conforman la red. Después, se realizan pruebas de funcionamiento relacionadas con la interfaz web y las acciones posibles por parte del administrador de la red. Para concluir este apartado, se realizan una serie de pruebas finales donde se comprueban tanto el envío como la recepción de mensajes a través de la red completa.

6.1. Conectividad Bluetooth Low Energy

En el desarrollo de la red, los primeros dispositivos implementados han sido las motas y, dado que aún no teníamos forma de probarlas, se ha optado por una comprobación rápida del funcionamiento de las mismas empleando una aplicación móvil desarrollada por Nordic Semiconductor. Esta aplicación se denomina nRF Connect y nos permite realizar una serie de pruebas de depuración previas como conectividad BLE, envío de mensajes, etcétera. Estas pruebas se realizan de forma rápida y sencilla, además de ofrecernos la posibilidad de verificar los avances del proyecto.

La prueba de conectividad con nRF Connect se basa en el escaner de los dispositivos de alrededor y su posterior conexión. Se puede ver el correcto funcionamiento de la conectividad BLE mendiante la figura 6.1:

Posteriormente, se realizo la misma prueba de conectividad con la mota

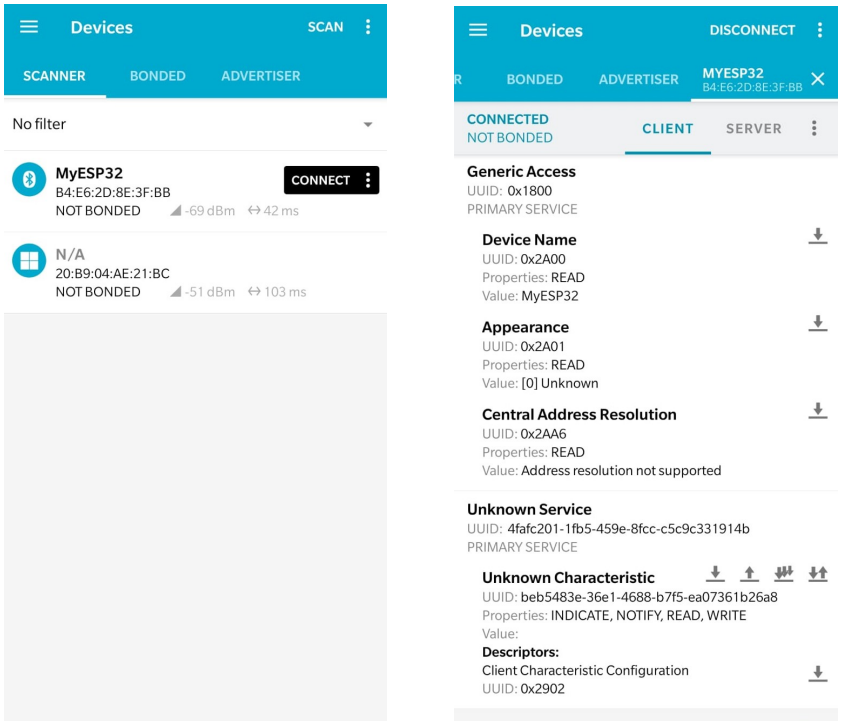


Figura 6.1: Prueba de Conectividad Bluetooth Low Energy

implementada en *Pycom LoPy 4.0*, obteniendo el mismo éxito que en la reflejada anteriormente.

6.2. Cobertura LoRa & LoRaWAN

Las motas, como se ha comentado a lo largo de desarrollo, no sólo incorporan la conectividad BLE, sino que también incluyen conexión vía LoRaWAN. Al comienzo del desarrollo, no se tenía preparada la implementación de los servidores Chirpstack por lo que se realizaron pruebas de conexión y alcance con la tecnología LoRa empleando la infraestructura propia de The Things Network. La infraestructura empleada en la prueba está formada por los servidores públicos TTN [44] y la integración con *TTN Mapper* [48].

Para comprobar el alcance de la cobertura de la tecnología LoRa debemos de seguir los siguientes pasos:

1. Configuración de los dispositivos (mota y gateway) en TTN. Para más información, véase el **Anexo C**.

2. Descarga de la app TTN Mapper en nuestro dispositivo móvil.

3. Enlace del smartphone con la mota. Este enlace se realiza a través de la configuración en la aplicación. Primero, nos registramos con el usuario y contraseña de la cuenta creada en TTN. Posteriormente, debemos indicar la aplicación y el dispositivo al que queremos enlazarnos. En este momento, se mostrarán las características de configuración del enlace como se ven en la figura 6.2 y únicamente tenemos que establecer el enlace.

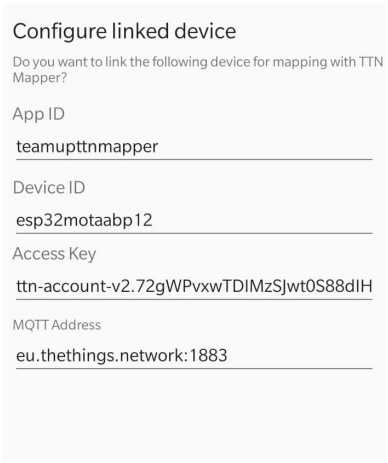


Figura 6.2: Establecimiento del Enlace en TTN Mapper

4. Ruta de prueba. En este momento, se comienzan a enviar a mensajes de forma períodica con varables como la localización del dispositivo, máximo RSSI, máxima SNR, la fecha y hora, ...

5. Comprobación de que los mensajes llegan correctamente al servidor. Esta comprobación se realiza desde la consola TTN en el apartado *traffic* dentro de nuestro gateway, tal y como se muestra en la figura 6.3.

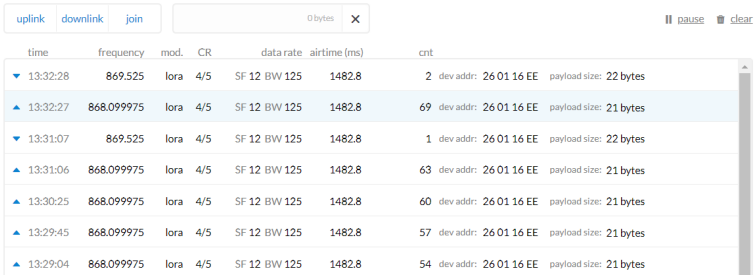


Figura 6.3: Recepción de Mensajes en TTN Gateway

6. Por último, solamente nos queda visualizar el mapa de cobertura generado por *TTN Mapper*. Este mapa, que se muestra en la figura 6.4 presenta un rango de tonos que reflejan la intensidad de la señal LoRa en la zona.

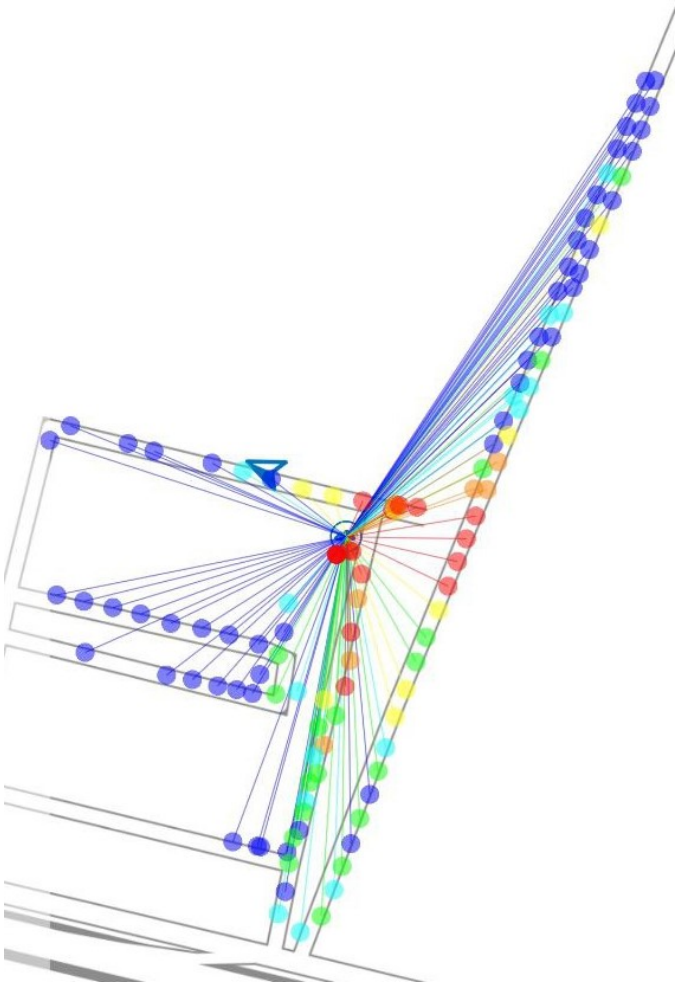


Figura 6.4: Rango de Cobertura LoRaWAN

Se ha comprobado que en la zona donde hay edificaciones la intensidad de la señal disminuye hasta el punto de perderse; mientras que en la zona este, vemos que el rango de cobertura es mayor. Esto se debe a no tener obstáculos apenas. Además, cabe destacar que el gateway en la prueba se localiza dentro de una casa y, por contraste, si está en una zona rural, entonces el rango de cobertura es más amplio.

6.3. TeamUp en zonas con cobertura

TeamUp es la aplicación desarrollada específicamente para el proyecto. Esta mantiene funcionalidades en zonas sin cobertura como enviar mensajes vía BLE una vez inicie sesión el usuario. En esta primera implementación, algunas funcionalidades se han desarrollado a modo de prueba para emplearse en zonas con cobertura, es decir, cualquier sitio donde podemos acceder a Internet (véase figura 6.5). Esta conectividad es necesaria dado que se ha empleado como almacenamiento una base de datos propia de la plataforma *Firebase* de Google.

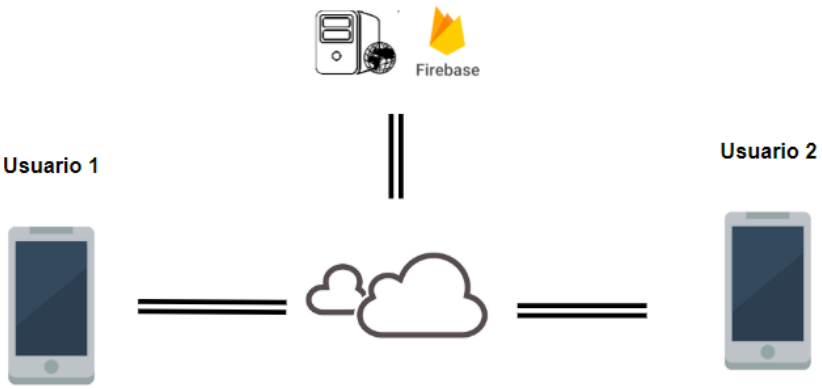


Figura 6.5: Esquema de Red en Zonas con Cobertura

A continuación, se realizan una serie de pruebas relacionadas con cada una de las funcionalidades que incorpora la App TeamUp.

- **Registro de Usuario:** consiste en registrar un nuevo usuario dentro de nuestro sistema. Para ello, lanzamos la aplicación y pulsamos sobre *Registrar*. Esto nos lleva a una pantalla de registro como la que se muestra a continuación e introducimos los campos de nombre, email y contraseña. Esta pantalla podemos verla en la figura 6.6.

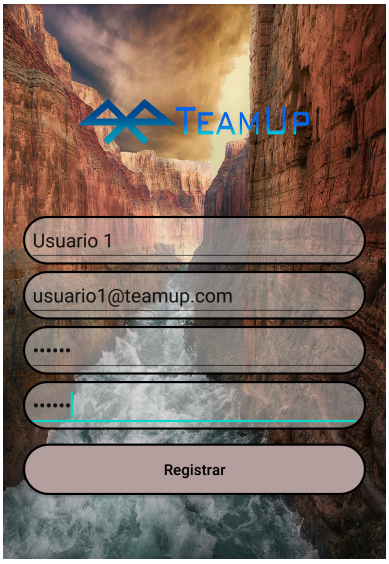


Figura 6.6: Pantalla de Registro de TeamUp

En la figura 6.7 se representa el apartado *Authentication* de *Firebase* y comprobamos que efectivamente aparecen nuestro nuevo usuario.

Identificador	Proveedores	Fecha de creación	Inicio de sesión	UID de usuario ↑
felix@gmail.com	✉	26 may. 2020	24 jun. 2020	Ak8E6PdXHxWk8Sdu3ECsfQXLciL2
jorge@teamup.com	✉	24 jun. 2020	24 jun. 2020	GsUmvAOMZENm2t0zu4nxmAriV...
prueba@gmail.com	✉	10 jun. 2020	10 jun. 2020	m3lGBc3tWVd4ezaLZBInRcj73PX2
usuario1@teamup.com	✉	27 jun. 2020	27 jun. 2020	vRcSvNxmiGRuK3341lEvLrli58J2

Figura 6.7: Authentication en Firebase

Además, se han capturado las tramas emitidas al realizar el registro con *Wireshark* (figura 6.8)y hemos comprobado que esta tranmisión se encuentra encapsulada por el protocolo Transport Layer Security (TLS). Por tanto, es complicado ver como funciona este proceso. Pero podemos intuir el funcionamiento mediante el diagrama de flujo generado por *Flow Graph* de *Wireshark* como se muestra en la figura 6.9.

No.	Time	Source	Destination	Protocol	Length	Info
91	13.946597	192.168.0.108	192.168.0.1	DNS	78	Standard query 0x8b83 AAAA www.googleapis.com
92	18.134216	192.168.0.109	255.255.255.255	UDP	214	49155 → 6667 Len=172
93	18.954928	192.168.0.108	192.168.0.1	DNS	78	Standard query 0x8b83 AAAA www.googleapis.com
94	18.966922	192.168.0.1	192.168.0.108	DNS	106	Standard query response 0x8b83 AAAA www.googleapis.com AAAA 2a00:1450
95	18.971169	192.168.0.108	192.168.0.1	DNS	78	Standard query 0xd906 A www.googleapis.com
96	18.982068	192.168.0.1	192.168.0.108	DNS	190	Standard query response 0xd906 A www.googleapis.com A 216.58.201.170
97	18.990794	192.168.0.108	216.58.201.170	TCP	66	50238 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
98	19.014544	216.58.201.170	192.168.0.108	TCP	66	443 → 50238 [SYN, ACK] Seq=0 Ack=1 Win=60720 Len=0 MSS=1380 SACK_PERM
99	19.014631	192.168.0.108	216.58.201.170	TCP	54	50238 → 443 [ACK] Seq=1 Ack=1 Win=131072 Len=0
100	19.020661	192.168.0.108	216.58.201.170	TLSv1.3	571	Client Hello
101	19.043165	216.58.201.170	192.168.0.108	TCP	54	443 → 50238 [ACK] Seq=1 Ack=518 Win=61952 Len=0
102	19.064828	216.58.201.170	192.168.0.108	TLSv1.3	1484	Server Hello, Change Cipher Spec
103	19.065565	216.58.201.170	192.168.0.108	TCP	1484	443 → 50238 [ACK] Seq=1431 Ack=518 Win=61952 Len=1430 [TCP segment of
104	19.065572	216.58.201.170	192.168.0.108	TLSv1.3	107	Application Data
105	19.065700	192.168.0.108	216.58.201.170	TCP	54	50238 → 443 [ACK] Seq=518 Ack=2914 Win=131072 Len=0
106	19.087945	192.168.0.108	216.58.201.170	TLSv1.3	254	Change Cipher Spec, Application Data
107	19.093971	192.168.0.108	216.58.201.170	TLSv1.3	898	Application Data

Figura 6.8: Captura de Registro con Wireshark

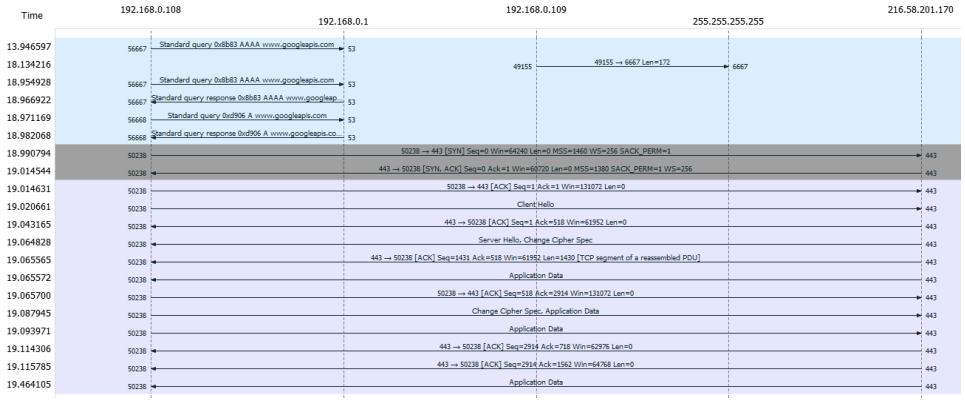


Figura 6.9: Flow Graph del Registro

En las capturas tomadas podemos comprender el funcionamiento del registro que consiste en la búsqueda del servidor de Google mediante DNS, el establecimiento de la conexión con un *3-Handshake* y el envío de la información cifrada.

- **Inicio de Sesión:** consiste en el inicio de sesión de un usuario en la aplicación. Este proceso se realiza a través de la pantalla que se muestra en la figura 6.10.

  Al igual que en el caso anterior, se han capturado los mensajes transmitidos con *Wireshark*.

  En las figuras 6.11 y 6.12 vemos que el comportamiento es similar, pero que las longitudes de los mensajes tranmistidos son variables y más extensos respecto al caso anterior.

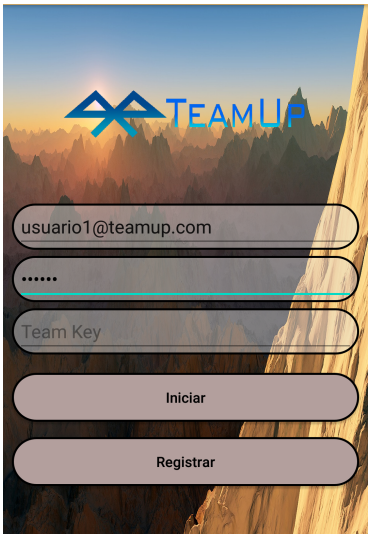
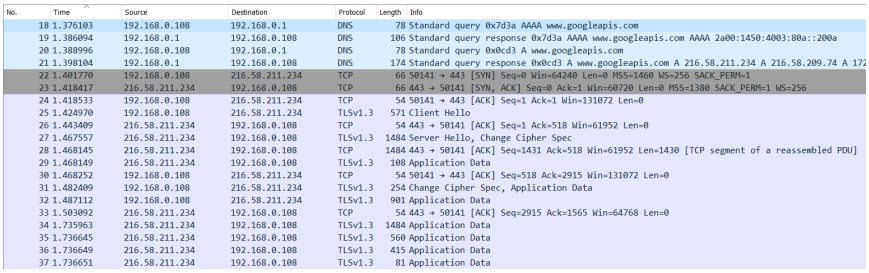


Figura 6.10: Pantalla de Inicio de Sesión de TeamUp



No.	Time	Source	Destination	Protocol	Length	Info
18	1.376103	192.168.0.108	192.168.0.1	DNS	78	Standard query 0x7d3a AAAA www.googleapis.com
19	1.386094	192.168.0.1	192.168.0.108	DNS	106	Standard query response 0x7d3a AAAA www.googleapis.com AAAA 2a00:1450:4003:80a::200a
20	1.388996	192.168.0.108	192.168.0.1	DNS	78	Standard query 0x0cd3 A www.googleapis.com
21	1.398104	192.168.0.1	192.168.0.108	DNS	174	Standard query response 0x0cd3 A www.googleapis.com A 216.58.211.234 A 216.58.209.74 A 172
22	1.401770	192.168.0.108	216.58.211.234	TCP	66	50141 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
23	1.418417	216.58.211.234	192.168.0.108	TCP	66	443 → 50141 [SYN, ACK] Seq=0 Ack=1 Win=60720 Len=0 MSS=1380 SACK_PERM=1 WS=256
24	1.418533	192.168.0.108	216.58.211.234	TCP	54	50141 → 443 [ACK] Seq=1 Ack=1 Win=131072 Len=0
25	1.424970	192.168.0.108	216.58.211.234	TLSv1.3	571	Client Hello
26	1.443409	216.58.211.234	192.168.0.108	TCP	54	443 → 50141 [ACK] Seq=1 Ack=518 Win=61952 Len=0
27	1.467557	216.58.211.234	192.168.0.108	TLSv1.3	1484	Server Hello, Change Cipher Spec
28	1.468145	216.58.211.234	192.168.0.108	TCP	1484	443 → 50141 [ACK] Seq=1431 Ack=518 Win=61952 Len=1430 [TCP segment of a reassembled PDU]
29	1.468149	216.58.211.234	192.168.0.108	TLSv1.3	108	Application Data
30	1.468252	192.168.0.108	216.58.211.234	TCP	54	50141 → 443 [ACK] Seq=518 Ack=2915 Win=131072 Len=0
31	1.482409	192.168.0.108	216.58.211.234	TLSv1.3	254	Change Cipher Spec, Application Data
32	1.487112	192.168.0.108	216.58.211.234	TLSv1.3	901	Application Data
33	1.503092	216.58.211.234	192.168.0.108	TCP	54	443 → 50141 [ACK] Seq=2915 Ack=1565 Win=64768 Len=0
34	1.735963	216.58.211.234	192.168.0.108	TLSv1.3	1484	Application Data
35	1.736645	216.58.211.234	192.168.0.108	TLSv1.3	560	Application Data
36	1.736649	216.58.211.234	192.168.0.108	TLSv1.3	415	Application Data
37	1.736651	216.58.211.234	192.168.0.108	TLSv1.3	81	Application Data

Figura 6.11: Captura de Inicio de Sesión con Wireshark

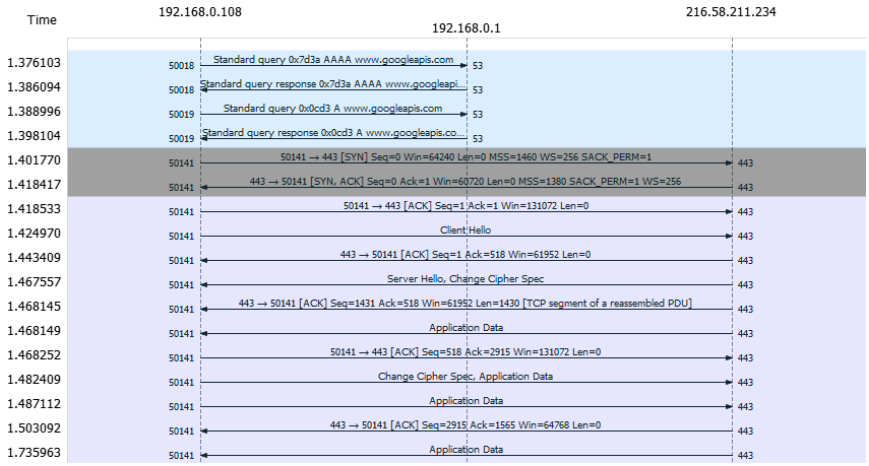


Figura 6.12: Flow Graph del Inicio de Sesión

- **Visualización del Chat:** consiste en el envío de mensajes en zona con cobertura y estos se visualizan en la pantalla de chat como se ve en la figura 6.13.

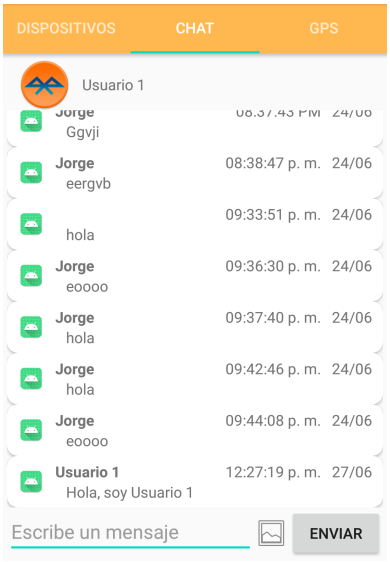


Figura 6.13: Pantalla de Chat de TeamUp

Estos mensajes se almacenan en el nodo *Chat* por defecto y con el formato que se muestra en la figura 6.14.

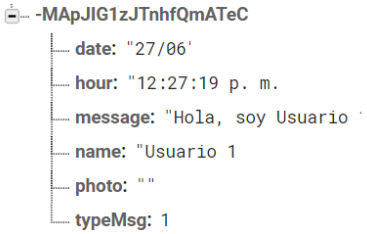


Figura 6.14: Mensaje Almacenado en Firebase

Una vez más se han capturado los mensajes transmitidos con *Wireshark*, obteniendo las figuras 6.15 y 6.16.

No.	Time	Source	Destination	Protocol	Length	Info
40	3.357751	192.168.0.108	192.168.0.1	DNS	79	Standard query 0x025f AAAA play.googleapis.com
41	3.369879	192.168.0.1	192.168.0.108	DNS	107	Standard query response 0x025f AAAA play.googleapis.com AAAA 2a00:1450:4003:801::200a
42	3.371292	192.168.0.108	192.168.0.1	DNS	79	Standard query 0x766e A play.googleapis.com
43	3.378836	192.168.0.1	192.168.0.108	DNS	95	Standard query response 0x766e A play.googleapis.com A 172.217.17.10
44	3.382655	192.168.0.108	172.217.17.10	TCP	66	50331 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
45	3.400014	172.217.17.10	192.168.0.108	TCP	66	443 → 50331 [SYN, ACK] Seq=0 Ack=1 Win=60720 Len=0 MSS=1380 SACK_PERM=1 WS=256
46	3.400139	192.168.0.108	172.217.17.10	TCP	54	50331 → 443 [ACK] Seq=1 Ack=1 Win=131072 Len=0
47	3.407142	192.168.0.108	172.217.17.10	TLSv1.2	235	Client Hello
48	3.422777	172.217.17.10	192.168.0.108	TCP	54	443 → 50331 [ACK] Seq=1 Ack=182 Win=61952 Len=0
49	3.443221	172.217.17.10	192.168.0.108	TLSv1.2	1484	Server Hello
50	3.443640	172.217.17.10	192.168.0.108	TLSv1.2	1471	Certificate, Server Key Exchange, Server Hello Done
51	3.443724	192.168.0.108	172.217.17.10	TCP	54	50331 → 443 [ACK] Seq=182 Ack=2848 Win=131072 Len=0
52	3.480460	192.168.0.108	172.217.17.10	TLSv1.2	147	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
53	3.481149	192.168.0.108	172.217.17.10	TLSv1.2	294	Application Data
54	3.495794	172.217.17.10	192.168.0.108	TLSv1.2	346	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
55	3.502243	172.217.17.10	192.168.0.108	TCP	54	443 → 50331 [ACK] Seq=3140 Ack=515 Win=62976 Len=0
56	3.527184	172.217.17.10	192.168.0.108	TLSv1.2	1220	Application Data
57	3.527267	192.168.0.108	172.217.17.10	TCP	54	50331 → 443 [ACK] Seq=515 Ack=4306 Win=131072 Len=0
58	3.527496	172.217.17.10	192.168.0.108	TLSv1.2	139	Application Data
59	3.527500	172.217.17.10	192.168.0.108	TLSv1.2	88	Application Data
60	3.527580	192.168.0.108	172.217.17.10	TCP	54	50331 → 443 [ACK] Seq=515 Ack=4425 Win=130816 Len=0
61	3.537232	192.168.0.108	172.217.17.10	TLSv1.2	1004	Application Data
62	3.559152	172.217.17.10	192.168.0.108	TCP	54	443 → 50331 [ACK] Seq=4425 Ack=1465 Win=64768 Len=0
63	3.580658	172.217.17.10	192.168.0.108	TLSv1.2	915	Application Data
64	3.581259	172.217.17.10	192.168.0.108	TLSv1.2	193	Application Data

Figura 6.15: Captura del Envío de Mensaje Online

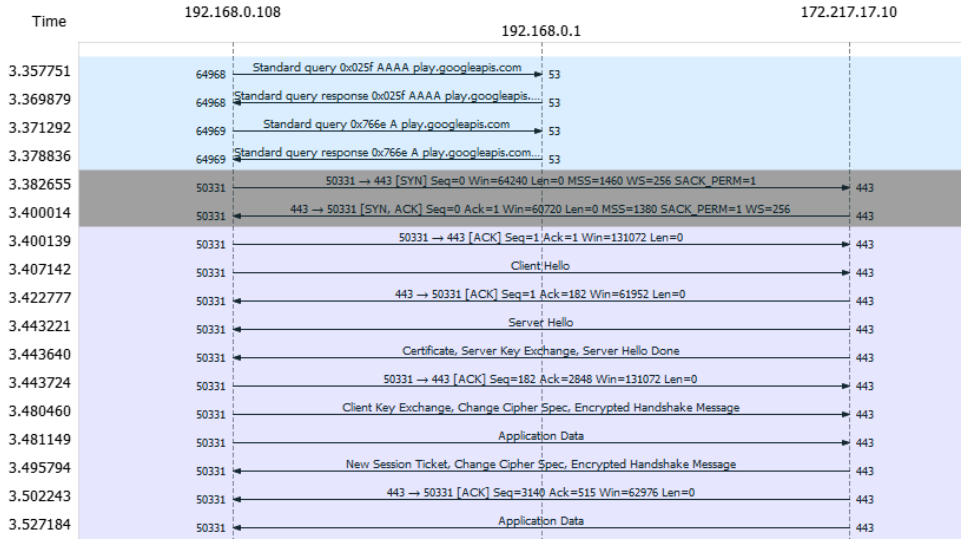


Figura 6.16: Flow Graph del Envío de Mensaje Online

6.4. Envío de mensajes vía BLE

En este apartado se realiza la prueba de conexión y envío de mensajes vía BLE, es decir, se realizan envíos de mensajes de texto desde el dispositivo móvil a la mota. Este proceso se ha realizado empleando dos aplicaciones distintas.

6.4.1. nRF Connect

Primero, se ha empleado la apliación nRF Connect para comprobar que la recepción de los mensajes BLE se reciban correctamente. Al abrir la aplicación, debemos conectarnos como se hizo previamente y posteriormente podemos enviar mensajes a través del servicio que implementa la mota. Enviado el mensaje, podemos ver si este ha llegado a la mota enlazada mediante el monitor serie.

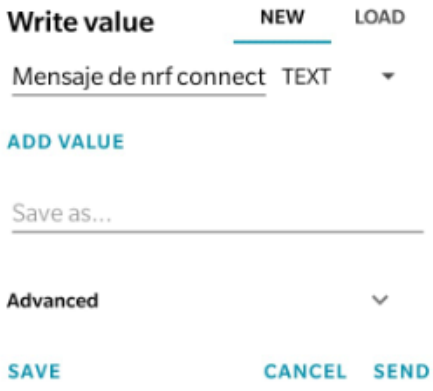


Figura 6.17: Envío de Mensaje desde nRF Connect

```
Startup (ESP32 Chip ID: B93F8E2DE6B4)
Node TTGO #12, ABP v1.0
Waiting a client connection to notify...
Using only one channel
Packet queued
270651: EV_TXCOMPLETE (includes waiting for RX windows)
Received using BLE: Mensaje de nrf connect
Time between last two packets: 41014 msec
LoRaWAN message sent
Packet queued
2743113: EV_TXCOMPLETE (includes waiting for RX windows)
```

Figura 6.18: Recepción del Mensaje nRF Connect

En las figuras 6.17 y 6.18, vemos la efectiva transmisión del mensaje emitido por la aplicación nRF Connect y su posterior reenvío mediante LoRaWAN.

6.4.2. TeamUp

A continuación, se ha realizado la misma prueba pero desde la aplicación desarrollada específicamente para el proyecto (*TeamUp*). El funcionamiento es idéntico al explicado para nRF Connect. Así, primero se realiza la conexión con la mota y, posteriormente, envía mensajes mediante BLE. Este proceso se realiza desde la pantalla que podemos observar en la figura 6.19.

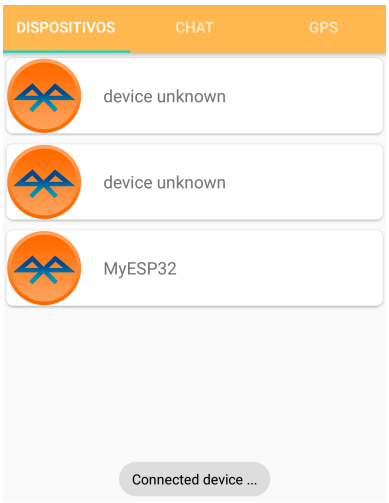


Figura 6.19: Envío de Mensaje desde TeamUp

Por defecto, si la localización del dispositivo se encuentra activa se envía un mensaje específico de localización. A parte, podemos realizar el envío de los mensajes de texto en cualquier momento, tal y como se muestra en la figura 6.20.

```
Received using BLE: #GPS# 38.03404 -4.087042
Time between last two packets: 154768 msec
LoRaWAN message sent
Packet queued
9853085: EV_TXCOMPLETE (includes waiting for RX windows)
Received using BLE: Mensaje desde TeamUp
Time between last two packets: 151844 msec
LoRaWAN message sent
Packet queued
19342769: EV_TXCOMPLETE (includes waiting for RX windows)
Received using BLE: #GPS# 38.034042 -4.086793
Time between last two packets: 28149 msec
LoRaWAN message sent
Packet queued
```

Figura 6.20: Recepción de Mensajes TeamUp

6.5. Envío de mensajes vía LoRaWAN

En este apartado se realiza la prueba de conexión y envío de mensajes vía LoRaWAN entre las motas y los servidores Chirpstack. En esta conexión, la mota es fundamental dado que se encarga de la retransmisión de los mensajes como se puede ver en las figuras expuestan con anterioridad.

Partiendo de que el envío de la mota es correcto, podemos seguir el paso de los mensajes a través de los distintos componentes de la red.

- **Gateway:** Los gateway utilizados (*single channel gateway* implementado sobre una mota ESP32) incorporan una página web propia donde se almacena un historial con los mensajes recibidos (véase figura 6.21). Estos mensajes se pasan al bridge incorporado en la Raspberry Pi.

Message History

Time	Node		Channel	SF	pRSSI
Saturday 27-6-2020 17:00:53	00 00 00 12	0	868100000	7	-73
Saturday 27-6-2020 16:57:28	00 00 00 12	0	868100000	7	-75
Saturday 27-6-2020 16:53:32	00 00 00 12	0	868100000	7	-71

Figura 6.21: Recepción de Mensajes en Gateway

- **Servidor de Red:** El bridge incorporado pasa el mensaje al servidor de red Chirpstack. Hasta este momento se mantiene la integridad del mensaje debido al funcionamiento de la red LoRaWAN y aún se mantiene el mensaje cifrado. En la figura 6.22 se ven algunos mensajes desde la consola Chirpstack a la que accedemos con un navegador indicando la url: `http:192.168.0.25:8080`, además de necesitar el usuario y la clave. Los mensajes (tramas LoRaWAN) de este servidor los encontramos en el apartado *Live LoRaWAN Frame* del *Gateway.*

- **Servidor de Aplicación:** El servidor de aplicación almacena las tramas y las descifra empleando la clave AppKey. En este servidor podemos visualizar de nuevo las tramas LoRaWAN y, además, se permite ver el mensaje a nivel de aplicación, como se observa en la figura 6.23.

  El mensaje se encuentra reflejado en la variable *data*, pero este por defecto en Chirpstack se almacena en Base64. Por lo tanto, si seleccionamos un mensaje y lo decodificamos, se muestra el mensaje.

```
"I0dQUyMgMzguMDM0MDQgLTQuMDg3MDQy" --> #GPS# 38.03404 -4.087042
"TWVuc2FqZSBkZXNkZSBUZWFtVXA="     --> Mensaje desde TeamUp
```

Figura 6.22: Tramas LoRaWAN en el Servidor de Red Chirpstack

Figura 6.23: Mensajes en el Servidor de Aplicación Chirpstack

6.6. Funcionalidades de Interfaz Web

La interfaz web del administrador de la red TeamUp implementa una serie de funcionalidades (véase el **Apartado 5.7**). En este apartado, se

trata de mostrar estas funcionalidades mediante el uso de un navegador. Las alternativas se seleccionan sobre una barra que se muestra en la figura 6.24.

Figura 6.24: Funcionalidades de la Interfaz Web

- *Inicio:* nos lleva a la página principal que se ve en la figura 6.25.

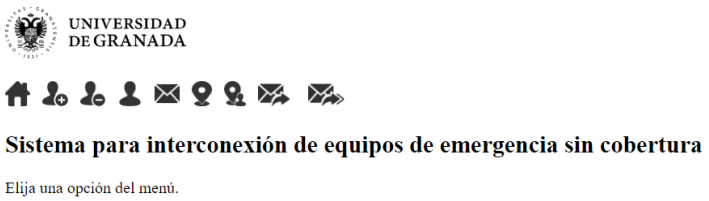


Figura 6.25: Inicio (Home)

- *Registrar usuario:* nos permite el registro de nuevos usuarios como se muestra en la figura 6.26.

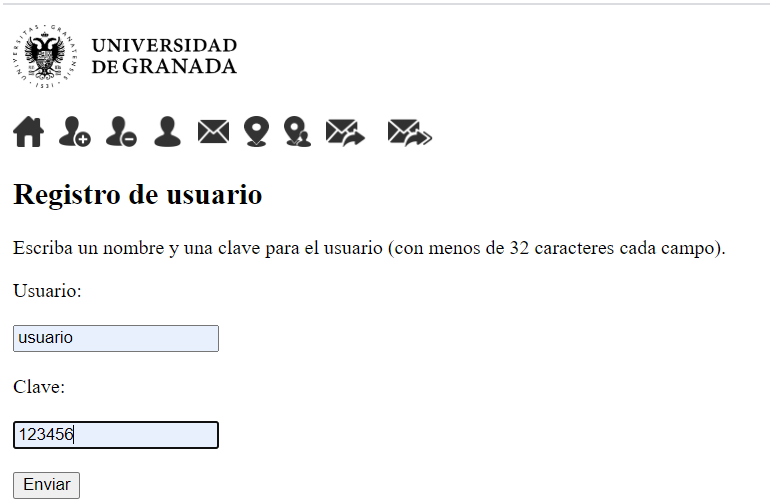


Figura 6.26: Registro de Usuario

En la figura 6.27 se muestra en pantalla una confirmación del registro del nuevo usuario.

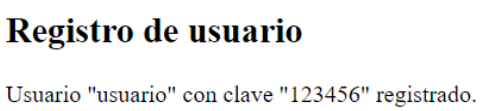


Figura 6.27: Confirmación del Registro de Usuario

Estos cambios los podemos visualizar mediante la funcionalidad *Ver Usuarios* donde se muestra una tabla con los usuarios registrados. Además, se observa que los usuarios nuevos no tienen asignados ningún dispositivo hasta que se conectan con él por primera vez. Esto podemos observarlo en la figura 6.28.

Figura 6.28: Confirmación en Tabla de Usuarios del Registro de Usuario

- *Borrar usuario:* nos permite borrar a un usuario, como muestra se realiza la eliminación del usuario creado anteriormente. Esta funcionalidad se muestra en la figura 6.29.

Figura 6.29: Borrar Usuario

Al realizar la operación, se muestra un mensaje de comprobación como se observa en la figura 6.30.

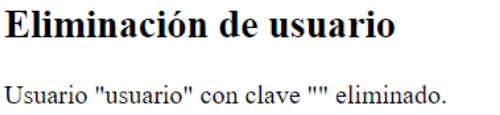


Figura 6.30: Confirmación de Borrado de Usuario

Por último, se puede realizar la comprobación mediante la tabla de usuarios que se muestra en la figura 6.31.

Usuarios registrados en el sistema:

user	dev_eui	last_seen_at
Felix	0000000000000012	2020-06-29 12:03:02.901912+01

Figura 6.31: Confirmación en Tabla de Usuarios

- *Ver usuarios:* nos permite visualizar los usuarios registrados en el sistema y si estos están enlazados a algún dispositivo. Se muestra en la figura 6.31.
- *Ver mensajes:* nos permite visualizar los mensajes enviados y recibidos por nuestro sistema de comunicaciones. Estos mensajes se pueden ver en la figura 6.32.

Registro de los mensajes recibidos por el servidor de aplicación LoRaWAN:

id	dev_eui	message	direction	time
1	0000000000000001	Testing...	uplink	2020-06-29 12:06:28.702775+01
2	0000000000000001	Testing...	uplink	2020-06-29 12:07:57.171057+01
3	0000000000000012	Hola	uplink	2020-06-29 12:07:57.358838+01
4	0000000000000012	Mensaje de Prueba	uplink	2020-06-29 12:07:57.545654+01
5	70b3d54994de968f	#L# Felix 123456	uplink	2020-06-29 12:07:57.726828+01
6	70b3d54994de968f	#LACK#	downlink	2020-06-29 12:07:57.908563+01
7	70b3d54994de968f	Esto es una prueba	downlink	2020-06-29 12:07:58.092014+01

Figura 6.32: Visualización de la Tabla de Mensajes

- *Ver ubicación de usuarios:* nos permite comprobar la ubicación de los usuarios mediante una tabla, como se muestra en la figura 6.33.

Registro de posición de los usuarios registrados:

id	time	user	latitude	longitude
1	2020-06-29 12:03:04.096463+01	Felix	38.03403	-4.08682
2	2020-06-29 12:03:04.278825+01	Felix	38.03403	-4.08682
3	2020-06-29 12:03:04.458862+01	Felix	38.03403	-4.08682
4	2020-06-29 12:05:33.049203+01	Felix	38.03403	-4.08682
5	2020-06-29 12:05:33.231547+01	Felix	38.03403	-4.08682
6	2020-06-29 12:05:33.412105+01	Felix	38.03403	-4.08682
7	2020-06-29 12:06:30.439+01	Felix	38.03403	-4.08682
8	2020-06-29 12:06:30.62415+01	Felix	38.03403	-4.08682
9	2020-06-29 12:06:30.808693+01	Felix	38.03403	-4.08682
10	2020-06-29 12:07:59.760907+01	Felix	38.03403	-4.08682
11	2020-06-29 12:07:59.941776+01	Felix	38.03403	-4.08682
12	2020-06-29 12:08:00.153932+01	Felix	38.03403	-4.08682

Figura 6.33: Visualización de la Tabla de Localización

- *Mandar mensaje a usuario:* nos permite el envío de mensajes desde la interfaz web por parte de un administrador. Estos mensajes se pueden enviar de forma individual a un usuario o a todos los usuarios registrados.

  Primero accedemos a la interfaz web y a la página que permite el envío de mensajes. Se indica el usuario al que queremos enviar el mensaje y el mensaje en sí, como se muestra en la figura 6.34. Se supone que el usuario en cuestión se encuentra registrado en el sistema en este instante.

Figura 6.34: Envío de Mensajes desde Interfaz Web

Unos instantes después, podemos comprobar que el mensaje ha sido recibido por el usuario en su dispositivo como se muestra en la figura 6.35.

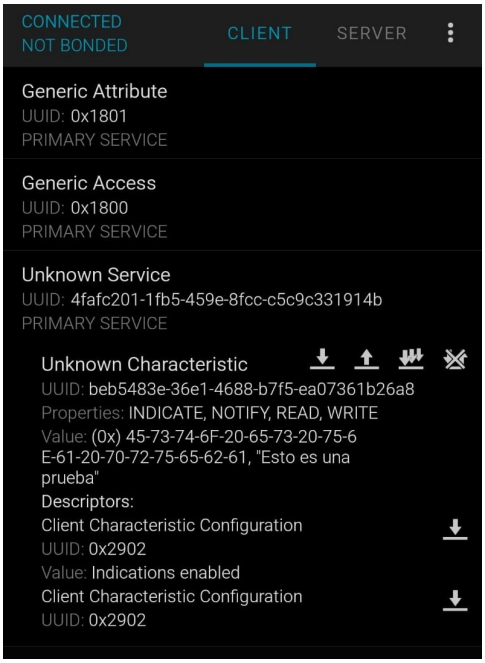


Figura 6.35: Recepción del Mensaje enviado desde Interfaz Web

Por último, se puede realizar una comprobación mediante la visualización de los mensajes almacenados en la red. Esta comprobación se muestra en la interfaz web de la figura 6.36.

Registro de los mensajes recibidos por el servidor de aplicación LoRaWAN:

id	dev_eui	message	time	direction
1	70b3d54994de968f	#L# Felix 123456	2020-07-03 22:26:36.542257+01	uplink
2	70b3d54994de968f	#LACK#	2020-07-03 22:26:37.339966+01	downlink
3	70b3d54994de968f	Esto es una prueba	2020-07-03 22:27:30.233449+01	downlink

Figura 6.36: Comprobación de Mensajes enviado desde Interfaz Web

6.7. Inicio de Sesión del Usuario al Sistema

Esta prueba explican los pasos seguidos para el inicio de sesión de un usuario al sistema en su totalidad la red TeamUp.

1. **Puesta en marcha del Servidor:** se inicia el servidor Python que se encarga del traspaso de los mensajes de una base de datos a otra, y de la visualización web. En la figura 6.37 se muestra la puesta en marcha del servidor.

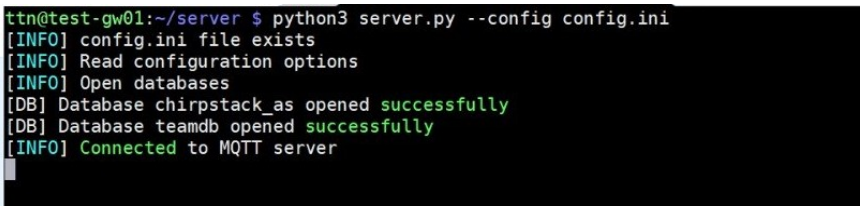


Figura 6.37: Puesta en Marcha del Servidor

2. **Conexión LoRaWAN:** se espera a que la mota establezca la conexión con nuestro gateway LoRaWAN. Este paso se puede observar en la figura 6.38.

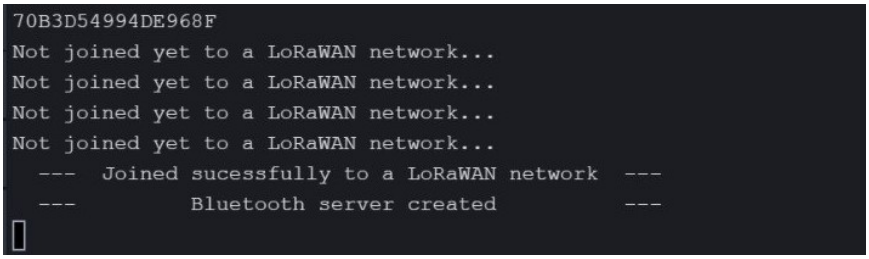


Figura 6.38: Conexión LoRaWAN Mota-Gateway

3. **Conexión BLE:** se establece la conexión BLE entre el dispositivo y la mota. En este caso, se ha empleado la aplicación *nRF Connect* de forma idética a la explicada anteriormente.

4. **Envío de mensaje Login:** el mensaje de inicio de sesión o login necesita un formato específico.

   ```
   #L# [usuario] [contraseña]
   ```

   Este mensaje se encuentra guardado en el fichero *login2* en *nRF Connect* para poder repetir las pruebas sin necesidad de tener que introducirlo en cada intento. Además, podemos ver el mensaje enviado en la aplicación en la figura 6.39.

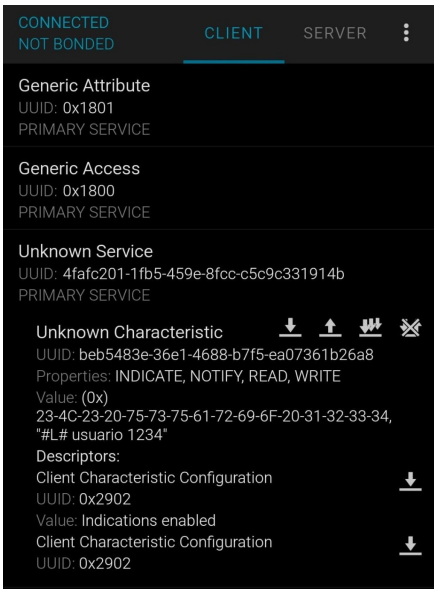


Figura 6.39: Envío de mensaje Login

5. **Recepción del mensaje Login ACK:** al segundo de la transmisión del mensaje Login, se recibe un mensaje de confirmación con el formato:

   ```
   #LACK#
   ```

   En la figura 6.40 se muestra la recepción del mensaje *Login ACK*.

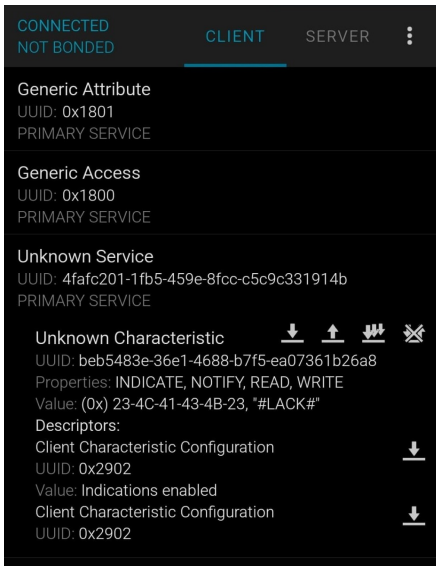


Figura 6.40: Recepción del mensaje Login ACK

6. **Mensajes en la Mota:** en la mota (*LoPy 4.0*) se observa que se recibe el mensaje de Login (*#L#*) por BLE y se manda por LoRaWAN; y el mensaje Login ACK (*#LACK#*) por LoRaWAN y lo retransmite por BLE. Estos mensajes se muestran en la figura 6.41.

```
  ---  Joined sucessfully to a LoRaWAN network  ---
  ---          Bluetooth server created          ---
  ---            BT client connected             ---
BT write request with value = b'#L# usuario 1234'
Message sent through LoRaWAN = b'#L# usuario 1234'
LoRaWAN message received on port=1, size=6, payload=#LACK#
BT message sent with payload=#LACK#
```

Figura 6.41: Mensajes en la Mota

7. **Mensajes en el Servidor:** se observan los mismos mensajes en el servidor lanzado para comprobar que el traspaso de información entre las bases de datos se ha efectuado correctamente. La figura 6.42 presenta los mensajes vistos desde el servidor.

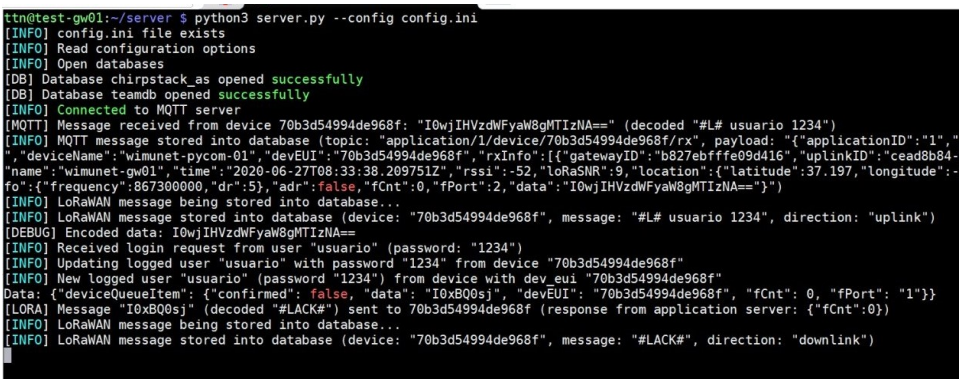


Figura 6.42: Mensajes en el Servidor

8. **Mensajes en Interfaz Web:** se visualizan los mensajes desde el punto de vista del administrador de la red (véase figura 6.43).

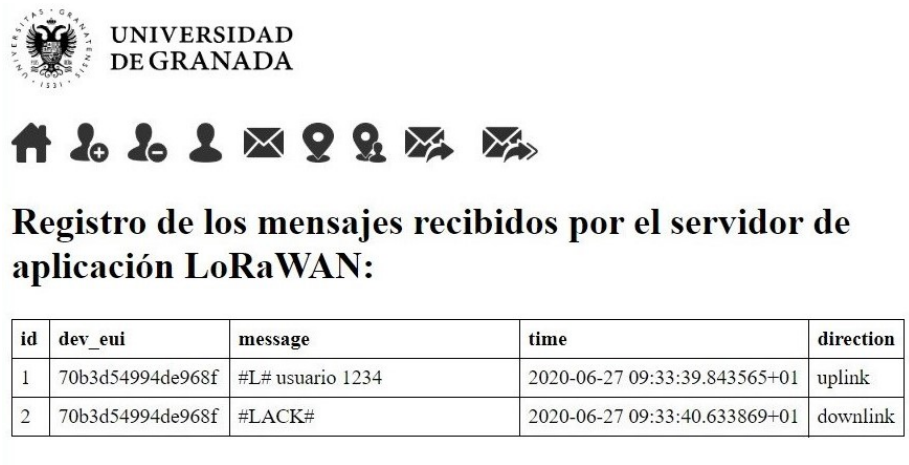


Figura 6.43: Mensajes en el Interfaz Web

6.8. Recepción de Mensajes en el Usuario

Esta prueba refleja la recepción de mensajes desde el punto de vista del usuario. Hasta el momento, se han realizado pruebas de envío de mensajes empleando las aplicaciones móviles. En esta prueba, se realiza el envío del mensaje desde el terminal, aunque de se ha implementado esta funcionalidad en la interfaz web del administrador, y se muestra la recepción del mensaje. Los pasos seguidos en esta demostración son:

1. **Lanzar el Servidor:** se inicia el servidor Python que se encarga del traspaso de los mensajes de una base de datos a otra, y de la visualización web.

2. **Conexión LoRaWAN y BLE:** se espera a que la mota establezca la conexión con nuestro gateway LoRaWAN y se realiza la conexión BLE entre el smartphone y la mota utilizando *nRF Connect*.

3. **Registro del Usuario:** se realiza el registro de la misma forma que se explicó en el apartado anterior.

4. **Envío del Mensaje desde terminal:** se realiza el envío de mensajes desde la terminal empleando el siguiente comando:

   ```
   python3 server.py --touser usuario --message "Probando..."
   ```

   En la figura 6.44 se muestra el envío del mensaje desde el terminal.

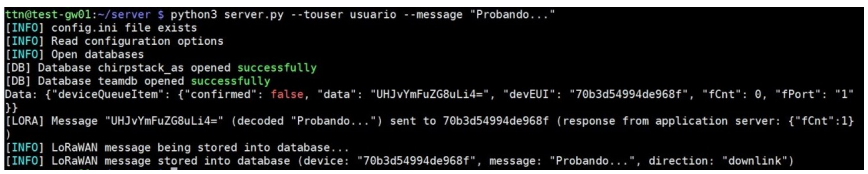


Figura 6.44: Envío del Mensaje desde terminal

5. **Comprobación en la Mota:** se comprueba que el mensaje ha sido recibido en el mota *LoPy 4.0* y esta lo retransmite vía BLE al dispositivo móvil. Esto se puede apreciar en la figura 6.45.

6. **Comprobación en el Móvil:** se comprueba que el mensaje llega de forma correcta al dispositivo móvil mediante la comunicación BLE, como se muestra en la figura 6.46.

7. **Comprobación del Administrador:** obviamente, estos mensajes también se almacenan en nuestra base de datos y se pueden visualizar en el navegador (véase figura 6.47).

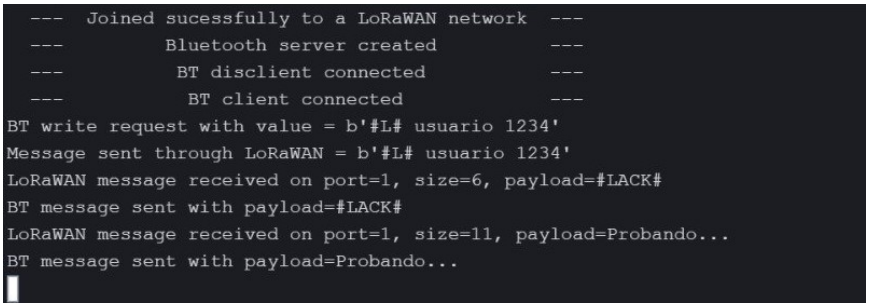


Figura 6.45: Comprobación en la Mota

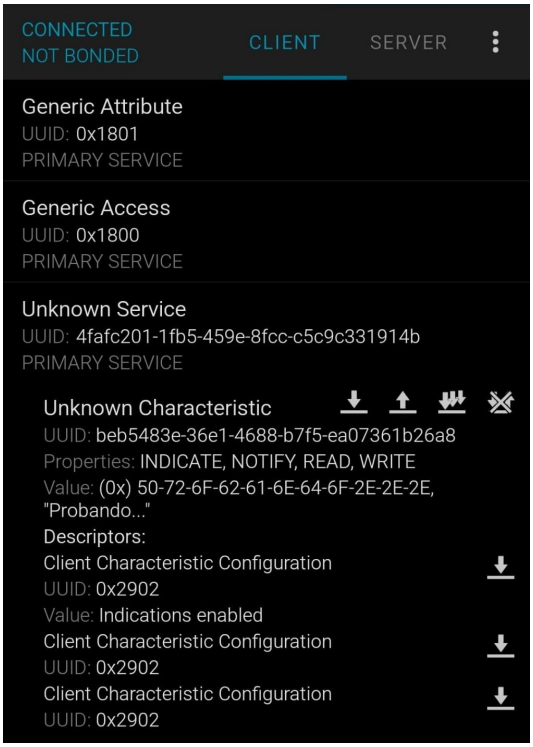


Figura 6.46: Comprobación en el Móvil

64	70b3d54994de968f	Probando...	2020-06-27 09:58:44.633613+01	downlink

Figura 6.47: Comprobación del Administrador

6.9. Conclusiones de las Pruebas

En este último apartado del capítulo, se exponen las conclusiones obtenidas en cada una de las pruebas realizadas y explicadas anteriormente.

1. **Conectividad Bluetooth Low Energy:** esta prueba ha permitido comprobar si la implementación de la mota respecto a la comunicación BLE funciona como se esperaba. Aunque únicamente se comprueba la conectividad en esta prueba y no el envío de mensajes.

2. **Cobertura LoRa & LoRaWAN:** esta prueba ha permitido comprobar si la infraestructura Chirpstack y la comunicación LoRaWAN funcionan correctamente. Además, se ha comprobado el alcance de la comuncicación LoRaWAN para conocer el rango de cobertura que podemos generar en zonas determinadas.

3. **TeamUp en zonas con conbertura:** estas pruebas muestran el funcionamiento principal de la aplicación desarrollada y nos permiten comprobar que el funcionamiento y visualización de la misma está acorde con los requisitos que se plantearon a resolver, es decir, registro, inicio de sesión, envío de mensajes y localización, ...

4. **Envío de mensajes vía BLE:** estas pruebas han permitido comprobar el funcionamiento de la comunicación BLE. No sólo el registro de usuarios sino también el envío de mensajes desde el smartphono y su posterior recepción en la mota. Estas pruebas se realizaron empleando dos aplicaciones móviles: *nRF Connect* para depurar nuestro código y ver el funcionamiento de recepción de las motas y *TeamUp* para verificación del funcionamiento del envío de mensajes desde nuestra aplicación y su posterior recepción.

5. **Envío de mensajes vía LoRaWAN:** esta prueba ha permitido comprobar si el reenvío de mensajes vía LoRaWAN desde la mota a los servidores ha sido eficiente y, si se realiza el posterior almacenamiento de estos mensajes en las bases de datos.

6. **Funcionalidades de Interfaz Web:** se han realizado una serie de pruebas relacionadas con las acciones de administración del sistema. Se han comprobado que funcionan correctamente las funciones implementadas como son:

   - Registrar y borrar usuarios del sistema.
   - Visualizar los usuarios registrados.
   - Visualizar los mensajes enviados y recibidos en el sistema.
   - Visualizar las ubicaciones de los ususarios activos en el sistema.
   - Enviar mensajes a un usuario específico o a todos los usuarios activos en el sistema.

7. **Inicio de Sesión del Usuario en el Sistema:** esta prueba se ha realizado con el fin de comprobar el correcto funcionamiento de la red

en su conjunto. En la prueba se ha comprobado el envío de mensajes de inicio de sesión desde el dispositivo móvil a los servidores, pasando por los nodos intermedios de la red. Como se ha mencionado, los mensajes de incio de sesión como su correspondiente *acknowledgement* se han realizado con exito y se han visualizado en las diferentes etapas de la red.

8. **Recepción de Mensajes en el Usuario:** esta prueba, al igual que la anterior, se ha realizado con el fin de comprobar el rendimiento de la red en su totalidad. En comparación, en esta prueba el envío de mensajes se realiza desde la parte del administrador de la red y no desde el usuario. Este envío ha sido satisfactorio y, de igual forma que en la prueba anterior, se han podido comprobar el seguimiento del mensaje desde su creación hasta su recepción, pasando por todos los nodos intermedios de la red.

Capítulo 7

Conclusiones y Trabajos Futuros

En este capítulo se exponen las conclusiones obtenidas a lo largo del proyecto y las líneas de trabajos futuros a realizar entorno al mismo.

7.1. Conclusiones

Este proyecto ha supuesto bastantes dificultades debidas a la complejidad del mismo. Esta complejidad se debe al problema a resolver que consiste en la interconexión de un equipo de emergencias en zonas sin cobertura.

Principalmente, se realizó un estudio del tecnologías y capacidades de las mismas que podíamos emplear en zonas sin cobertura. Las tecnologías seleccionadas finalmente fueron BLE, LoRa y LoRaWAN. Todas ellas tienen gran importancia en el mundo del Internet de las Cosas (IoT). Posteriormente, se realizó un diseño de la red a implementar que esta formada por un conjunto de elementos (dispositivo móvil, mota, gateway , servidores e interfaz web). Todos estos elementos forman la red y permiten realizar todos los escenarios de pruebas que se idearon y comprobaron.

Previas a esas pruebas, se realizaron una serie de implementaciones de funcionamiento sobre los dispositivos mencionados anteriormente. Cada dispositivo implica unas capacidades distintas, lo que nos llevó a realizar las implementaciones usando diversos entornos de programación y lenguajes. Estas implementaciones se explican en el **Capítulo 5** y se subdividen según el dispositivo a emplear. Además, cabe destacar, el desarrollo de una aplicación totalmente funcional con la red creada.

Una vez se integraron todos los desarrollos y se montó la red completa, se han realizado un conjunto de pruebas exitosas que se pueden comprobar en el **Capítulo 6**. El objetivo de estas pruebas es la verificación del correcto funcionamiento de la red y las diferentes funcionalidades implementadas, para así ver si puede suplir el problema inicial. Como se ha comprobado con anterioridad, la red diseñada es perfecta para zonas sin cobertura dado que emplea tecnologías suficientes para albergar un amplio rango. Además, supone una gran ventaja emplear estas tecnologías debido a su extensa información online que facilitó la implementación y uso de la misma. Por otro lado, se comprueba que los usuarios pueden enviar y recibir mensajes a través de esta red, lo que está ampliamente comprobado y se puede afirmar que la red es 100 % operativa.

7.2. Trabajos Futuros

A pesar de tener en pleno funcionamiento la red diseñada, cabe comentar que existen posibles mejoras o ideas a desarrollar aún. Estas se reflejan a continuación:

- Mejora del diseño de la interfaz web.
- Implementación y comprobación de la ruta de un usuario desde la interfaz web, empleando *Google Maps*.
- Diseño de una red mallada (MANET) mediante Wi-Fi como vía complementaria a la red que se encuentra en funcionamiento, de forma que los miembros del equipo de emergencias empleen esta estructura de red en caso de encontrarse cerca unos de otros.
- Análisis del rendimiento, capacidad y coste energético asociado a las tecnologías empleadas en la red actual.

Bibliografía

[1] Kevin Ashton. *"That 'Internet of Things' Thing", RFID Journal*. 1999.

[2] Gartner. *Ïnternet of Things Defined - Tech Definitions by Gartner"*. URL: `https://www.gartner.com/en/information-technology/glossary/internet-of-things` (visitado 14-04-2020).

[3] IoT-Analytics. *"State of the IoT 2018: Number of IoT devices now at 7B"*. URL: `https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/` (visitado 14-04-2020).

[4] Outernet. *"Lantern, el gadget de Outernet que te da acceso a Internet libre desde el espacio usando wifi"*. URL: `http://www.compartirwifi.com/blog/lantern-el-gadget-de-outernet-que-te-da-acceso-a-internet-libre-desde-el-espacio-usando-wifi/` (visitado 14-04-2020).

[5] *Official Page of Othernet*. URL: `https://othernet.is/` (visitado 14-04-2020).

[6] *The Commotion Wireless Project*. URL: `https://commotionwireless.net/` (visitado 14-04-2020).

[7] *Commotion Construction Kit*. URL: `https://commotionwireless.net/docs/cck/` (visitado 14-04-2020).

[8] *SPOT X Globalstar*. URL: `https://www.globalstar.com/es-la/products/personnel-safety/spotx` (visitado 14-04-2020).

[9] Uepaa Safety App Lösung. *"Die clevere App-Lösung für Alleinarbeiter mit Totmannfunktion. Leichtgewichtige Smartphone App mit 24/7 Notrufzentrale"*. URL: `https://safety.uepaa.ch/de/` (visitado 14-04-2020).

[10] *Official Page of ETH Zurich*. URL: `https://ethz.ch/en.html` (visitado 14-04-2020).

[11] *Official Page of Swisscom*. URL: `https://www.swisscom.ch/en/about.html` (visitado 14-04-2020).

[12] *Official Page of Rega WEINMANN.* URL: https://www.weinmann-emergency.com/es/soluciones/ccsv/rega-reference/ (visitado 14-04-2020).

[13] Alberto Ceberio. *Aplicación para emergencias en la montaña sin cobertura de telefonía móvil.* URL: https://pararpensaractuarblog.wordpress.com/2015/06/15/aplicacion-para-emergencias-en-la-montana-sin-cobertura-de-telefonia-movil/ (visitado 14-04-2020).

[14] *Official Page of Beartooth.* URL: https://beartooth.com/ (visitado 14-04-2020).

[15] Greg Barragan. *Bandas ISM.* URL: http://bandasism.blogspot.com/2013/07/bandas-ism.html (visitado 14-04-2020).

[16] *Official Page of CEPT.* URL: https://www.cept.org/ (visitado 14-04-2020).

[17] ETSI. *"Welcome to the World of Standards!".* URL: https://www.etsi.org/ (visitado 14-04-2020).

[18] Aprendiendo Arduino. *Banda ISM.* URL: https://aprendiendoarduino.wordpress.com/tag/banda-ism/ (visitado 14-04-2020).

[19] Bluetooth Technology. *Learn About Bluetooth.* URL: https://www.bluetooth.com/learn-about-bluetooth/ (visitado 14-04-2020).

[20] Bluetooth Technology. *Radio Versions.* URL: https://www.bluetooth.com/learn-about-bluetooth/bluetooth-technology/radio-versions/ (visitado 14-04-2020).

[21] IEEE. *"The world's largest technical professional organization dedicated to advancing technology for the benefit of humanity.".* URL: https://www.ieee.org/ (visitado 14-04-2020).

[22] Katie Wilson Stephen Wilson & Ezio Biglieri. *Academic Press Library in Mobile and Wireless Communications - 1st Edition.* Academic Press, 27th July 2016.

[23] Bluetooth Technology. *Core Specifications.* URL: https://www.bluetooth.com/specifications/bluetooth-core-specification/ (visitado 14-04-2020).

[24] Kevin Townsend Carles Cufí Akiba & Robert Davidson. *Getting Started with Bluetooth Low Energy.* O'Reilly Media, Inc., May 2014.

[25] Texas Instruments. *CC2540 and CC2541 Bluetooth low energy Software Developer's.* URL: http://www.ti.com/lit/ug/swru271g/swru271g.pdf.

[26] Libelium. *Bluetooth Low Energy Networking Guide.* URL: http://www.libelium.com/downloads/documentation/bluetooth-low-energy-networking_guide.pdf.

[27] *Modulación SSFH*. URL: https://www.circuitdesign.de/products/tech_info/Modulation/modulation_SS.asp.

[28] Bluetooth SIG. *Specification of the Bluetooth System, Covered Core Package version 4.2*.

[29] Bluetooth Technology. *GATT Bluetooth*. URL: https://www.bluetooth.com/specifications/archived-specifications/.

[30] *Official Page of LoRa Alliance*.

[31] *Documentation of LoRaWAN*. URL: https://lora-alliance.org/about-lorawan.

[32] *Pila de Potocolos LoRa*. URL: https://www.aprendiendoarduino.com/tag/lorawan/.

[33] Semtech. *Semtech LoRa Technology Overview*. URL: https://www.semtech.com/lora (visitado 14-04-2020).

[34] Technical Marketing Workgroup 1.0. *A technical overview of LoRa and LoRaWAN*. URL: https://www.tuv.com/media/corporate/products_1/electronic_components_and_lasers/TUeV_Rheinland_Overview_LoRa_and_LoRaWANtmp.pdf.

[35] *Documentation of LoRaWAN*. URL: https://lora.readthedocs.io/en/latest/.

[36] *Official Page of LoRaWAN*. URL: https://lorawan.es/.

[37] The Things Network. *Documentation of Spreading Factor, The Things Network*. URL: https://www.thethingsnetwork.org/article/how-spreading-factor-affects-lorawan-device-battery-life.

[38] The Things Network. *Documentation of Classes of Devices*. URL: https://www.thethingsnetwork.org/docs/lorawan/classes.html.

[39] Jorge Navarro Ortiz. *Apuntes Redes Inalámbricas y Movilidad*.

[40] The Things Network. *Documentation of Architecture of LoRaWAN*. URL: https://www.thethingsnetwork.org/docs/lorawan/architecture.html.

[41] The Things Network. *Documentation of Activation*. URL: https://www.thethingsnetwork.org/docs/lorawan/addressing.html.

[42] Jorge Navarro-Ortiz y col. "Integration of LoRaWAN and 4G/5G for the Industrial Internet of Things". En: *IEEE Communications Magazine* 56.2 (feb. de 2018). Conference Name: IEEE Communications Magazine, págs. 60-67. ISSN: 1558-1896. DOI: 10.1109/MCOM.2018.1700625.

[43] Alfaiot. *Comparation Chirpstack vs The Things Network*. URL: https://alfaiot.com/blog/ultimas-noticias-2/post/loraserver-chirpstack-vs-the-things-network-ttn-14.

[44] *Official Page of The Things Network*. URL: https://www.thethingsnetwork.org/.

[45] *Download Page of Arduino IDE*. URL: https://www.arduino.cc/en/Main/Software.

[46] *Download Page of Android Studio*. URL: https://developer.android.com/studio?hl=es-419.

[47] *Getting Started Chirpstack*. URL: https://www.chirpstack.io/guides/debian-ubuntu/.

[48] *Official Page of TTN Mapper*. URL: https://ttnmapper.org/.

[49] *Download Page nRF Connect by Nordic Semiconductor*. URL: https://www.nordicsemi.com/Software-and-tools/Development-Tools/nRF-Connect-for-mobile.

[50] *Official Page of Visual Studio Code*. URL: https://code.visualstudio.com/.

[51] *Official Page of PlatformIO*. URL: https://platformio.org/.

[52] *Official Page of Wireshark*. URL: https://www.wireshark.org/.

[53] Pycom. *API Reference for Bluetooth*. URL: https://docs.pycom.io/firmwareapi/pycom/network/bluetooth/gattscharacteristic/.

[54] Pycom. *API Reference for LoRa*. URL: https://docs.pycom.io/firmwareapi/pycom/network/lora/.

[55] Pycom. *API Reference for Nano Gateway*. URL: https://docs.pycom.io/tutorials/lora/lorawan-nano-gateway/.

[56] *Manual Postgres PHP*. URL: https://docs.pycom.io/tutorials/lora/lorawan-nano-gateway/.

[57] *Getting Started The Things Network*. URL: https://www.thethingsnetwork.org/docs/.

Apédices

Apéndice A

Manual de Usuario de TeamUp

En este archivo se explica el funcionamiento de la aplicación **TeamUp**. Esta aplicación está implementada en dos idiomas: inglés y castellano. Lo primero que debemos de hacer es comprobar que estén activas las funciones de localización y conexión Bluetooth en nuestro smartphone.

Al lanzar la aplicación, se obtendrá el idioma utilizado por defecto en el disposivo móvil. Posteriormente, se muestra la pantalla *Splash Screen* durante 2 segundos.

Después de la espera, se lanza una pantalla de login o inicio de sesión de usuario. En esta pantalla se tienen tres campos a rellenar:

- **E-mail:** especificar el correo electrónico con el que se realizo el registro de usuario.
- **Contraseña:** confirmación del usuario a través de una clave.
- **Team Key (Extra):** es una clave que diferencia los chats grupales que a los que acceder. Un grupo específico generará su clave que conoceran todos los miembros del mismo.

  En caso contrario, se debe acceder a la aplicación sin rellenarlo y, automáticamente, se accederá a un chat grupal general.

Además de la opción de iniciar sesión, en esta pantala, tenemos un botón para acceder al registro de usuario.

En la pantalla de registro de usuario, tendremos una serie de campos a rellenar para hacer efectivo el registro de un nuevo usuario.

- **Nombre:** identificación del usuario dentro de la aplicación.

- **E-mail:** especifica el correo electrónico con el que se registra el nuevo usuario.
- **Contraseña:** indica la clave de acceso correspondiente al usuario.
- **Constraseña Repetida:** repetición de la clave de acceso para asegurar que el usuario no introduzca erroneamente la clave.

En la figura A.1 se muestra la pantalla descrita anteriormente.

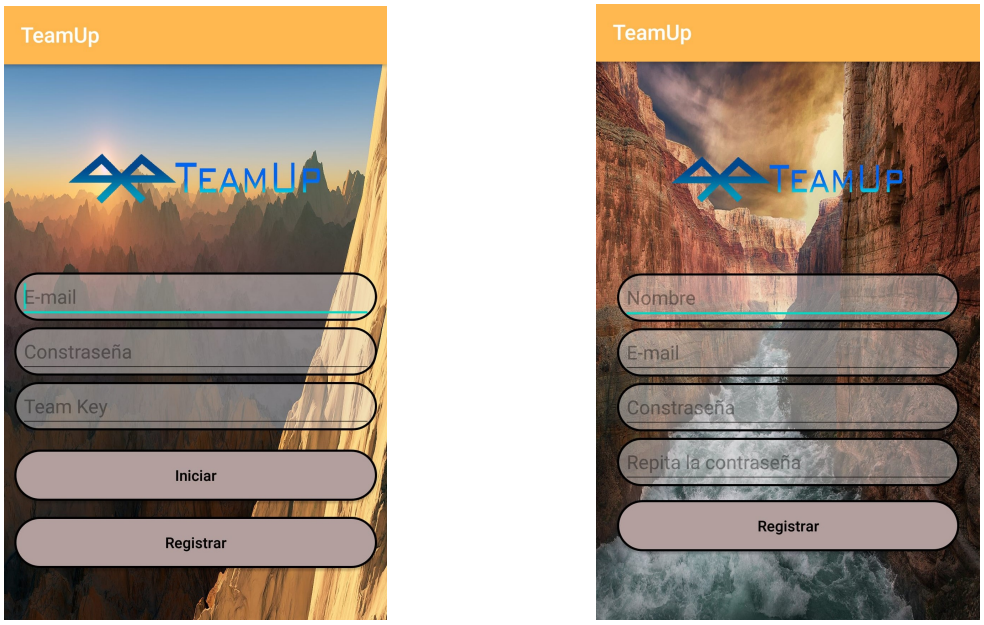


Figura A.1: Pantalla de Inicio de Sesión y Registro

Una vez se ha registrado el usuario, podemos acceder a la aplicación desde la pantalla de inicio de sesión con nuestro correo electrónico y contraseña.

En el interior de la aplicación podemos distinguir tres partes claramente, entre las que podemos navegar indistintamente deslizando con el dedo o pulsando sobre los botones de la parte superior de la pantalla.

1. **Dispositivos:** en esta pantalla se verán los dispositivos Ble de nuestro alrededor. Podemos conectarnos a cualquiera de ellos pulsando sobre su nombre y se confirma la conexión con un mensaje *"Dispositivo conectado ..."*.

   Además, esta pantalla nos ofrece la posibilidad de enviar mensajes vía Intenet en caso de tener conectividad Wi-Fi o telefónica. Y permite el envío a nuestro sistema privado en caso contrario, empleando Bluetooth Low Energy. Esta pantalla se muestra en la figura A.2.

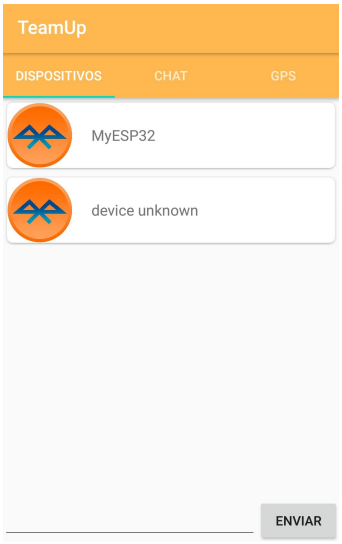


Figura A.2: Pantalla de Dispositivos Ble

2. **Chat:** en esta pantalla se visualizan los mensajes enviados al chat grupal. Además, se permite el envío de mensajes de texto e imagenes en caso de tener conectividad a Internet.

3. **GPS:** en esta pantalla se ven las coordenadas geográficas del usuario (longitud y latitud) y una visualización de la localización sobre un mapa.

   En la figura A.3 se muestran las pantallas de chat y GPS.

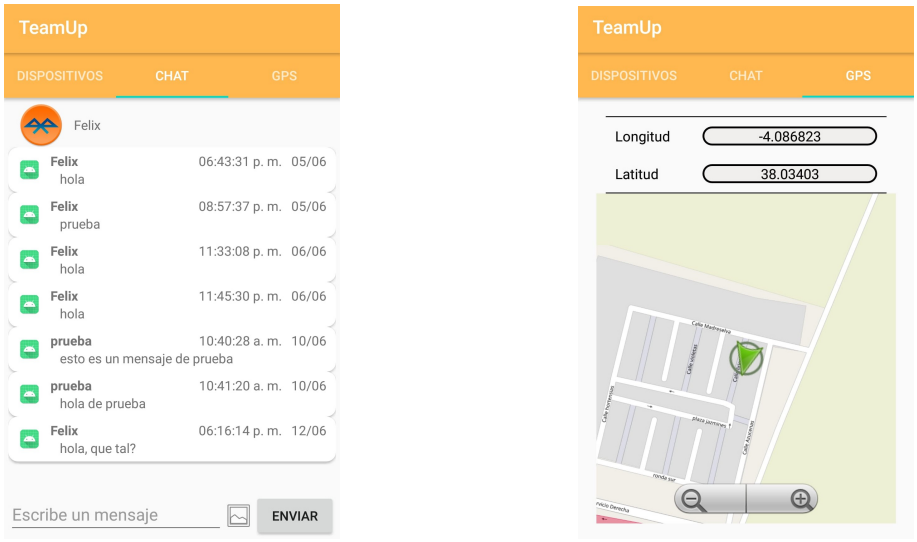


Figura A.3: Pantalla de Chat y localizacion GPS

Apéndice B

Instalación y Configuración del servidor Chirpstack

B.1. Introducción

Este documento muestra la instalación de los servidores de red y aplicación Chirpstack y la respectiva configuración del gateway y la mota para este servidor [47].

B.2. Instalación de los servidores Chirpstack

La instalación se ha realizado sobre un sistema operativo Raspbian Buster. Antes de comenzar con la instalación, se deben instalar las dependencias necesarias para el correcto funcionamiento de los servidores.

- **MQTT broker (Mosquitto):** Es una implementación del protocolo MQTT y permite la publicación de información en tópicos o temas y la subscripción a dichos tópicos.
- **Redis:** Es una base de datos interna que permite almacenar información temporal.
- **PostgresSQL:** Es una base de datos de largo tiempo empleada para almacenar recursos.

Estas dependencias se instalan con los siguientes comandos:

```
>> sudo apt install mosquitto mosquitto-clients
>> sudo apt install redis-server redis-tools
>> sudo apt install postgresql
```

A continuación, se configura la base de datos PosgreSQL y los usuarios. Primero entramos a PostgreSQL y creamos los roles y las bases de datos de servidor de red y aplicación.

```
>> sudo -u postgres psql
create role chirpstack_as with login password 'dbpassword';
create role chirpstack_ns with login password 'dbpassword';
create database chirpstack_as with owner chirpstack_as;
create database chirpstack_ns with owner chirpstack_ns;
```

El siguiente paso es modificar la base de datos del servidor de aplicación incluyendo dos extensiones y nos salimos de PostgreSQL.

```
\c chirpstack_as
create extension pg_trgm;
create extension hstore;
\q
```

Ahora se configura el repositorio software compatible con nuestro sistema. Para ello, se instala un gestor de directorios, se establece una clave para el nuevo repositorio y se añade creando un nuevo archivo.

```
>> sudo apt install apt-transport-https dirmngr
>> sudo apt-key adv --keyserver keyserver.ubuntu.com
    --recv-keys 1CE2AFD36DBCCA00
>> sudo echo "deb https://artifacts.chirpstack.io/
    packages/3.x/deb stable main" | sudo tee /etc/
    apt/sources.list.d/chirpstack.list
```

A continuación, se instala el *Gateway Bridge* de Chirpstack. Aunque este elemento se puede instalar en la misma máquina que los servidores o en otra distanciada. Primero, se instala el paquete y, luego, se inicia el servicio.

```
>> sudo apt install chirpstack-gateway-bridge
>> sudo systemctl start chirpstack-gateway-bridge
>> sudo systemctl enable chirpstack-gateway-bridge
```

Posteriormente, se procede a la instalación de los servidores de red y aplicación de Chripstack.

Servidor de Red Chirpstack

1. Se instala el paquete del servidor de red.

   ```
   >> sudo apt install chirpstack-network-server
   ```

2. Se modifica el archivo de configuración dependiendo de la banda de frecuencias empleada. En nuestro caso, se configurará para la banda EU868.

```
[general]
log_level=4

[postgresql]
dsn="postgres://chirpstack_ns:dbpassword@localhost/
    chirpstack_ns?sslmode=disable"

[network_server]
net_id="000000"

[network_server.band]
name="EU_863_870"

[[network_server.network_settings.extra_channels]]
frequency=867100000
min_dr=0
max_dr=5

[[network_server.network_settings.extra_channels]]
frequency=867300000
min_dr=0
max_dr=5

[[network_server.network_settings.extra_channels]]
frequency=867500000
min_dr=0
max_dr=5

[[network_server.network_settings.extra_channels]]
frequency=867700000
min_dr=0
max_dr=5

[[network_server.network_settings.extra_channels]]
frequency=867900000
min_dr=0
max_dr=5
```

En este punto, se reinicia el servidor de red y se establece para iniciarse al arrancar el sistema.

```
>> sudo systemctl start chirpstack-network-server
>> sudo systemctl enable chirpstack-network-server
```

Servidor de Aplicación de Chirpstack

1. Se instala el paquete del servidor de aplicación.

   ```
   >> sudo apt install chirpstack-application-server
   ```

2. Se modifica el archivo de configuración.

```
[general]
log_level=4

[postgresql]
dsn="postgres://chirpstack_as:dbpassword@localhost/
    chirpstack_as?sslmode=disable"

[application_server.external_api]
jwt_secret="***"
```

El secreto JWT se sobtiene siguiendo los siguientes pasos:

a) Generación de un secreto en Base64, se ha empleado el siguiente comando:

```
openssl rand -base64 32
```

b) Generación del JWT mediante la página `https://jwt.io/`. En esta página se genera el JWT completando una serie de campos y empleando el secreto generado con anterioridad. En la figura B.1 se muestra la creación del secreto JWT.

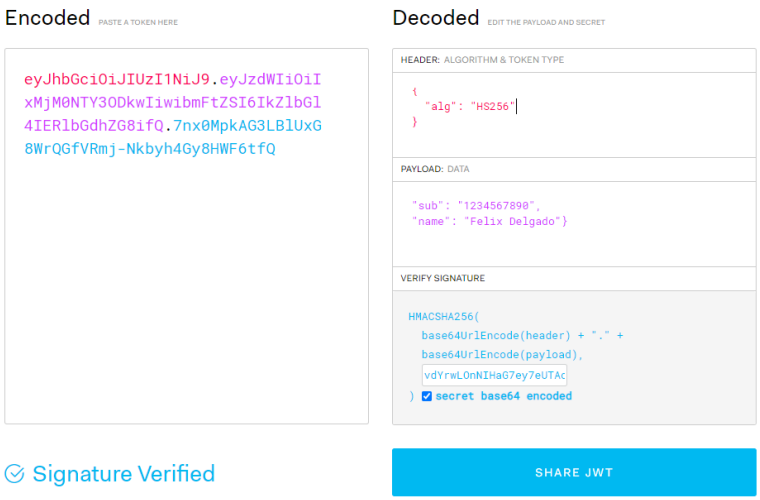


Figura B.1: Generación del secreto JWT

Por último, se reinicia el servidor de aplicación y se establece para iniciarse al arrancar el sistema.

```
>> sudo systemctl start chirpstack-application-server
>> sudo systemctl enable chirpstack-application-server
```

B.3. Configuración de los servidores Chirpstack

En este apartado se comenta el procedimiento seguido para la configuración de los elementos de red en el servidor Chirpstack. Para poder realizar la configuración, accedemos a la consola a través de un navegador con *localhost:8080*. Se solicitarán usuario y contraseña que por defecto es: *admin*.

B.3.1. Configuración del Gateway

El gateway o pasarela es el elemento intermedio entre la mota y los servidores Chirpstack. Esta pasarela se ha implementado sobre dispositivos ESP32, siendo una pasarela monocanal. Los aspectos más relevantes de la configuración de este dispositivo se encuentran en el **Capítulo 5**. Respecto a los servidores Chirpstack, se debe de declarar donde se ubica el servidor a partir de su dirección IP.

```
1 #define _TTNPORT 1700
2 #define _TTNSERVER "192.168.0.25"
```

Posteriormente, el procedimiento consiste en registrar la pasarela a través de la consola web de Chirpstack. Previo a este paso, se necesita crear una organización y un perfil de servicio como se muestra en la figura B.2.

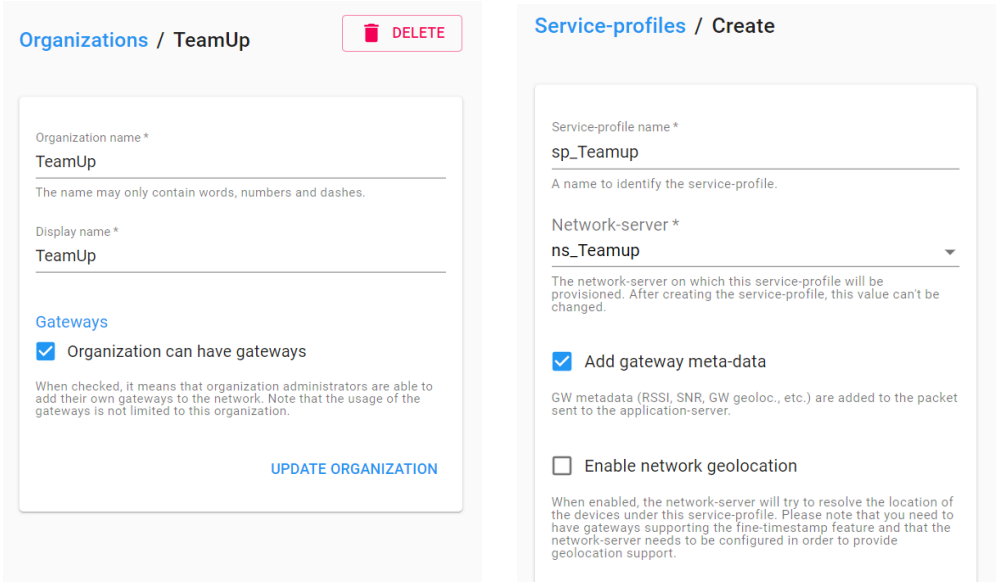


Figura B.2: Creación de Organización y Perfil en Chirpstack

Ahora si podemos introducir nuestra pasarela desde la pestaña de *Gateway*. Es imprescindible no confundirse al introducir el *Gateway ID* que identifica unívocamente a nuestra pasarela e indicar el perfil de servicio que emplea. Este paso se muestra en la figura B.3.

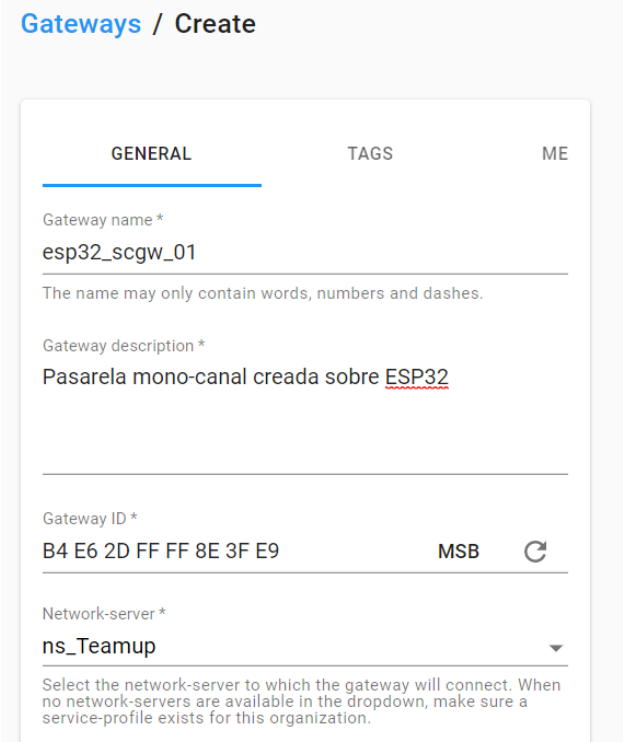


Figura B.3: Creación de Pasarela en Chirpstack

B.3.2. Configuración de la Aplicación y Dispositivos

En esta subsección, se indican los pasos a seguir para la configuración de los dispositivos en nuestros servidores. Primero, debemos de crear nuestra aplicación rellenando el registro de la misma e indicando el perfil de servicio enlazado, como se muestra en la figura B.4.

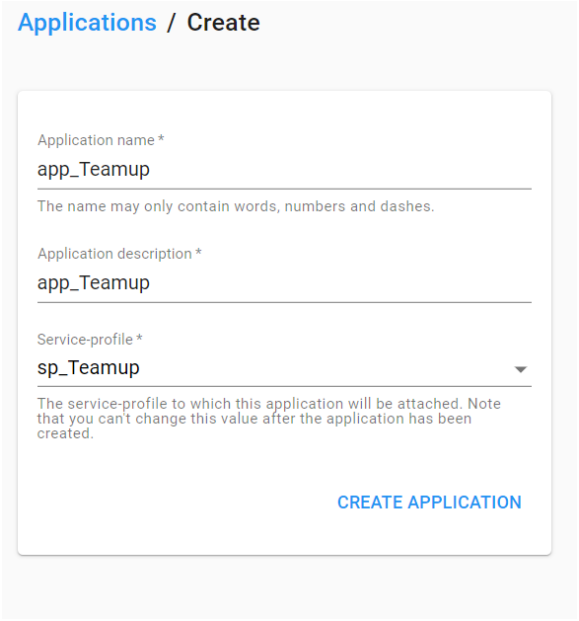


Figura B.4: Configuración de Aplicación en Chirpstack

Una vez creada la aplicación, se define un perfil de dispositivo que emplearemos de forma general a todos los dispositivos de nuestra red. Este perfil emplea la versión de LoRaWAN 1.0.0 y un máximo EIRP de 14.

A continuación, se crea el registro de los dispositivos en el apartado *Application/app_TeamUp/Devices* (véase figura B.5). Al registrar un nuevo dispositivo se deben de indicar el perfil de dispositivo y el identificador EUI. Además, debemos de comprobar el tipo de activación empleado. En nuestro caso, se ha empleado la activación ABP. Esta activación debe configurarse directamente sobre el dispositivo en la consola web y se indica el *Device Address*, el NwkSKey y el AppSKey.

Es importante deshabilitar el frame-counter, lo cual empeorará la seguridad de nuestra comunicación pero tendremos menos problemas a la hora de tomar las muestras.

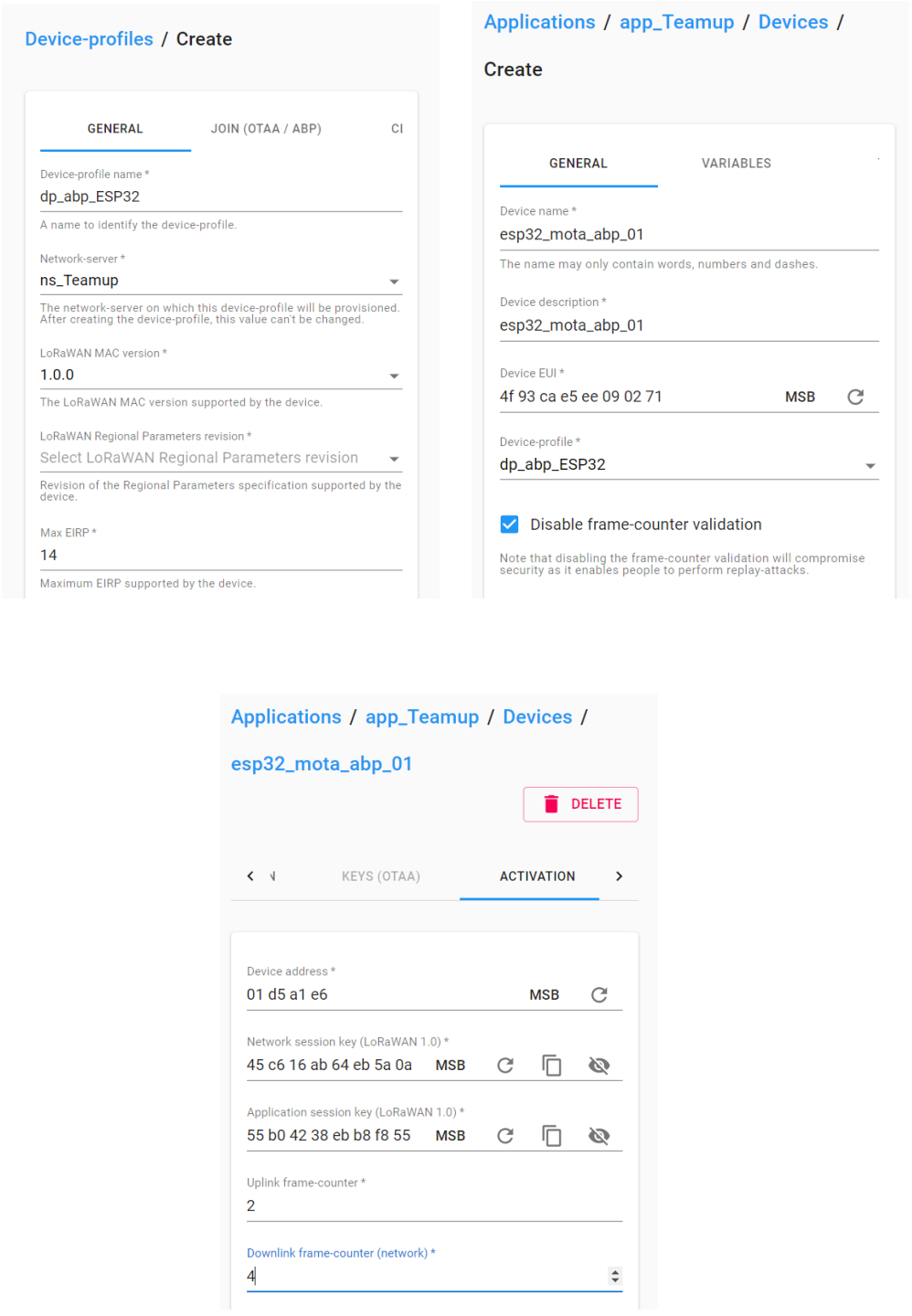


Figura B.5: Configuración de Dispositivos en Chirpstack

Apéndice C

Configuración del servidor TTN

C.1. Introducción

Este documento muestra la configuración del gateway y la mota en el servidor TTN [57].

C.2. Configuración

En este apartado se comenta el procedimiento seguido para la configuración de los elementos de red en el servidor TTN. Esto quiere decir que los elementos serán reconocidos por el servidor de la infraestructura a nivel mundial. Para poder realizar la configuración, es necesario las creación previa de un usuario de la plataforma TTN.

C.2.1. Configuración del Gateway

El gateway o pasarela es el elemento intermedio entre la mota y los servidores TTN. Esta pasarela se ha implementado sobre dispositivos ESP32, siendo una pasarela monocanal. A continuación, se comenta la diferencia en el código respecto a la implementación de la pasarela del **Anexo D**.

El único cambio respecto a la implementación citada anteriormente, es la declaración del servidor TTN. En este caso, nos conectamos a los servidores desplegados por TTN en el continente europeo.

```
1 #define _TTNPORT 1700
2 #define _TTNSERVER "router.eu.thehings.network"
```

Posteriormente, el procedimiento consiste en registrar la pasarela en la plataforma TTN. Para ello, entramos en *Console/Gateway/Register Gateway*. A continuación, se rellenan los valores del registro. Para poder introducir correctamente el EUI o ID del gateway, tenemos que seleccionar la opción *I'm using the legacy packet forwarder*. Además, incluimos una breve descripción, la frecuencia en la que se trabaja (868.1 MHz) y la localización GPS. Este procedimiento se puede ver claramente en la figura C.1.

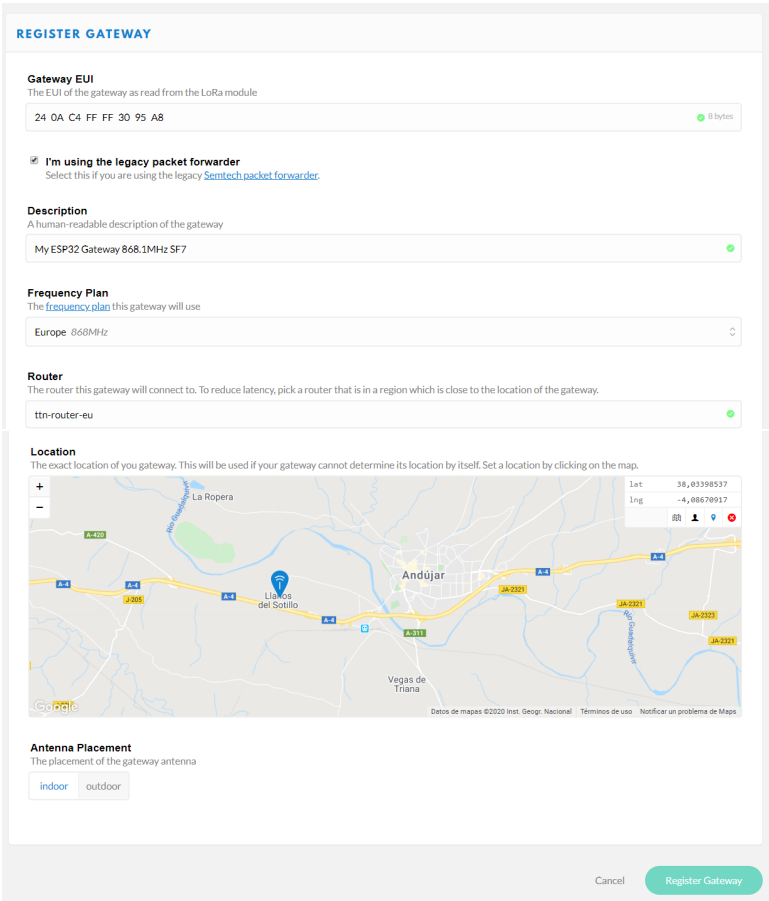


Figura C.1: Registro del Gateway en The Things Network

Por último, debemos comprobar si nuestra gateway esta correctamente implementada. La pasarela se debe ver activa en la misma consola, tal y como se aprecia en la figura C.2.

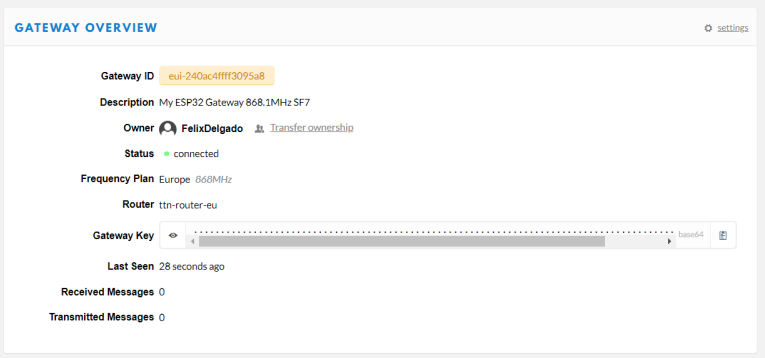


Figura C.2: Comprobación Gateway Conectado en The Things Network

C.2.2. Configuración de la Aplicación y Dispositivos

Ahora, se procede a la configuración de los dispositivos dentro de una aplicacion en los servidores TTN. Para ello, accedemos a *Console/Applications/Add Aplication* y creamos a apicación como se muestra en la figura C.3.

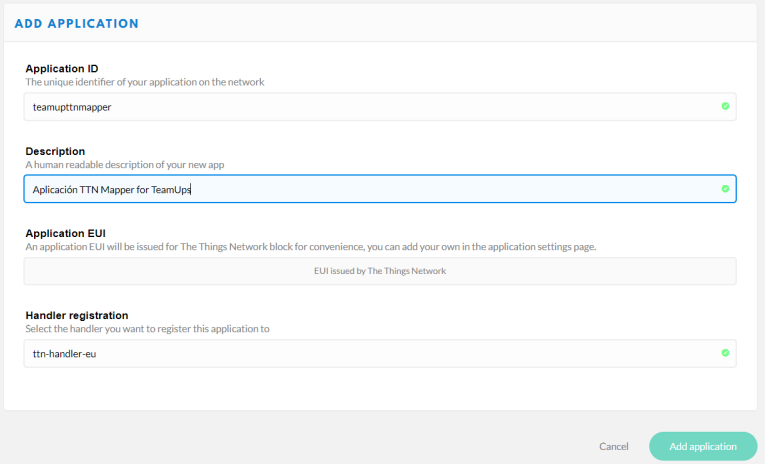


Figura C.3: Creación de Aplicación en The Things Network

Una vez creada la aplicación se va a definir el dispositivo perteneciente a la aplicación. Este paso se realiza desde nuestra aplicación en *Devices/Register Device* (véase figura C.4). En este punto, es importante no equivocarnos al incluir el valor de *App Key* y, puesto que el *Device Address* es definido por la plataforma, debemos modificarlo en nuestro código de la mota para que coincidan.

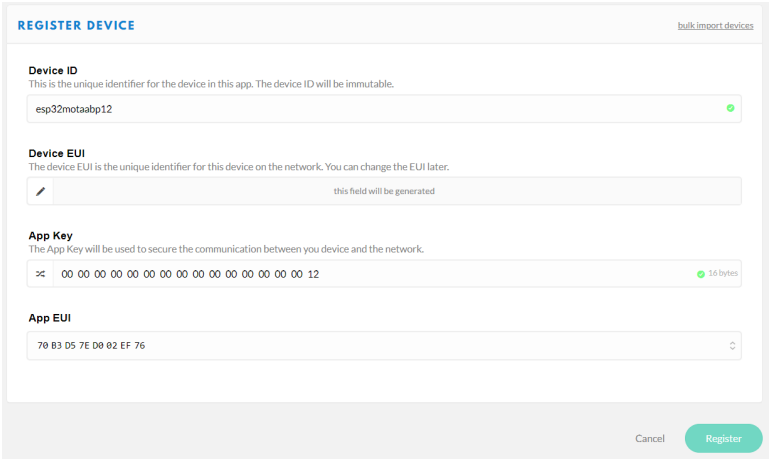


Figura C.4: Registro de Dispositivo en The Things Network

Para acabar con la configuración, en mi caso, he de cambiar el tipo de activación. Debido a que por defecto, el tipo de activación implementado por la plataforma es OTAA y mi dispositivo mantiene una activación ABP. Este cambio se produce sobre la configuración del dispositivo en *Settings* que se muestra en la figura C.5.

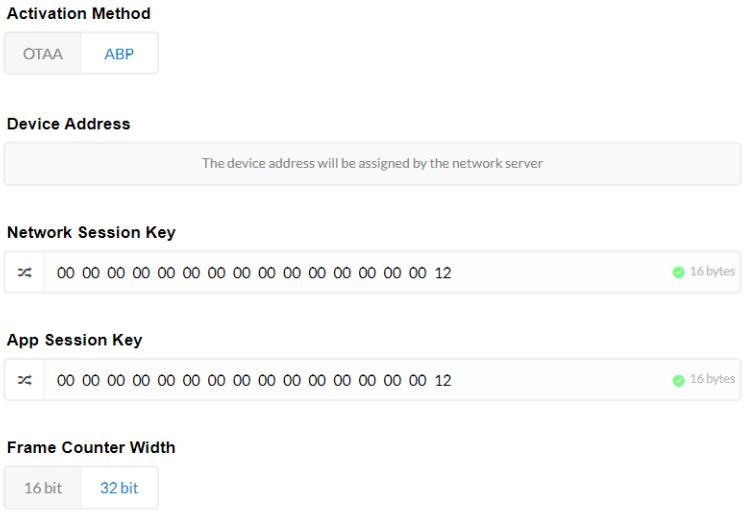


Figura C.5: Modo de Activación del Dispositivo en The Things Network