INTEGRACIÓN DE CONECTIVIDAD LORA EN DISPOSITIVOS ANDROID

LORA CONNECTIVITY INTEGRATION IN ANDROID DEVICES

Yaco Alejandro Santiago Pérez

Trabajo Fin de Master
Máster en Internet de las Cosas

Facultad de Informática
Universidad Complutense de Madrid

Curso: 2020-2021

Convocatoria ordinaria de julio de 2021
Calificación: 9

Director: Francisco D. Igual-Peña

Agradecimientos

Me gustaría agradecer a todas esas personas que han hecho posible este proyecto.

En especial a *Francisco D. Igual-Peña*, que a lo largo de la realización del máster fue quien me aportó los conocimientos y el interés para realizar mi TFM apoyándome en una tecnología de red como Lora.

Y cómo no, también por dedicarme su tiempo y haberme dado la oportunidad de realizar el trabajo bajo su dirección, otorgándome total libertad tanto en la gestión del tiempo como en la realización e implementación del mismo.

Y por último, a los profesores de la *Facultad de Informática* de la *Universidad Complutense de Madrid*, tanto del máster como del grado, por haberme aportado los conocimientos necesarios para realizar con éxito este proyecto.

Le dedico este trabajo a mis padres, por haberme apoyado siempre.
A mi pareja, por los ánimos y por haber compartido esta etapa juntos.
Y a mis amigos, por estar ahí incondicionalmente.

Resumen

Integración De Conectividad Lora En Dispositivos Android

En la actualidad existen zonas o situaciones en las cuales no se dispone de cobertura móvil. Esto nos limita seriamente a la hora de comunicarnos e incluso puede llegar a ser un problema.

Este problema afecta de lleno al mundo del *IOT* a la hora de realizar despliegues, ya que es primordial poder establecer una conexión entre los nodos desplegados. Para dar solución a esta limitación se utilizan las que son llamadas Redes de bajo consumo y largo alcance *(LPWAN)*. Estas redes permiten dotar de conectividad a los nodos desplegados.

Realizando este trabajo se pretende aportar una solución que permita disponer de conectividad inalámbrica en zonas sin cobertura. Mediante la implementación de un dispositivo *IOT* (un chip *ESP32*) con conectividad *LoRa* y el desarrollo de una biblioteca se permite dotar de conectividad *LoRa* a cualquier aplicación para dispositivos *Android*.

Se realizará una implementación real de la solución y se probará en un entorno real.

Palabras clave

IOT, LoRa, LPWAN, ESP32, Arduino, Android, Redes de bajo consumo y área extensa

Abstract

Lora Connectivity Integration In Android Devices

Currently there are areas or situations in which we do not have mobile coverage. This seriously limits us when it comes to communicating and can even become a problem.

This problem fully affects the *IOT* world when making deployments, since it is essential to be able to establish a connection between the deployed nodes. To solve this limitation, Low Power and Long Range Networks *(LPWAN)* are used. These networks provide connectivity to the deployed nodes.

By carrying out this work, it is intended to provide a solution that allows wireless connectivity in areas without coverage. By implementing an *IOT* device (an ESP32 chip) with *LoRa* connectivity and the development of a library, it is possible to provide *LoRa* connectivity to any application for *Android* devices.

A real implementation of the solution will be made and tested in a real environment.

Key words

IOT, LoRa, LPWAN, ESP32, Arduino, Android, Low Power Wide Area Network

Índice general

Índice de figuras

Capítulo 1. ¿Por qué dotar de LoRa a un dispositivo Android?

1.1. LPWAN

¿Qué es?

LPWAN (de las siglas en inglés de «Low Power Wide Area Network») son redes inalámbricas de medio y largo alcance, que pueden llegar a proporcionar varios kilómetros de cobertura [1].

Debido al actual auge en el desarrollo de soluciones del ámbito del Internet de las Cosas se ha extendido e intensificado su uso.

¿Qué ventajas tiene respecto GSM o 4G?

Las redes inalámbricas tradiciones de largo alcance son el *GSM (Global System for Mobile communications)* y *4G (Cuarta generación de tecnologías de telefonía móvil)* y su uso está muy extendido, principalmente en dispositivos de telefonía.

Sin embargo, a la hora de realizar un despliegue *IOT* utilizar una *LPWAN* aporta las siguientes ventajas:

- La más importante es el bajo consumo energético que tienen. Esto es un punto muy positivo dado que en la mayoría de despliegues *IOT* las fuentes de alimentación son muy limitadas.
- No necesitan grandes recursos hardware, ya que están destinadas a dispositivos con una capacidad de memoria y potencia de cálculo limitada y bajo consumo energético.
- Posibilitan tener conectividad inalámbrica en zonas donde no hay cobertura *GSM/4G*.
- Son redes que permiten la implementación de una infraestructura propia, de manera que no dependemos necesariamente de un proveedor de red. Aún así, cada vez más operadores de telefonía ofrecen infraestructura para este tipo de coberturas, como puede ser el caso de las redes *LTE-M* y *NB-IoT*.

¿Qué inconvenientes tiene?

- Proporcionan un ancho de banda muy limitado, del orden de *KB*. Disminuyendo el ancho de banda es como logran coberturas en distancias tan grandes.
- El uso de algunas de estas redes es bajo licencia y requiere de un desembolso económico para poder hacer uso de ellas, como es el caso de las anteriormente mencionadas *LTE-M* y *NB-IoT*.

LoRa

LoRa es una tecnología de modulación de amplio espectro. Esto le permite tolerar ruido y caminos múltiples de señal, mientras mantiene muy bajo el consumo de energía. Para lograr esto se ve sacrificado el ancho de banda, que es muy bajo comparado con otras tecnologías inalámbricas [2].

Para la realización de este proyecto se decidió contar con *LoRa* como tecnología de conexión inalámbrica.

Esta decisión fue tomada en base a que es una *LPWAN* libre de licencia, su señal llega a proporcionar un largo alcance, y en comparación a otras tecnologías inalámbricas tiene un consumo de batería muy bajo.

Como se ha comentado antes, estas características se obtienen sacrificando el ancho de banda, pero esto no supone un problema dado que no se va a requerir un ancho de banda amplio.

En cuanto a coste económico, al ser de licencia libre no supone un desembolso, y a la hora de encontrar hardware que sea compatible, los dispositivos con conectividad *LoRa* tienen un coste menor en comparación a otros con conectividad *LPWAN* como son *NB-IOT* o *Sigfox*.

A continuación se puede observar un gráfico que compara las redes *Sigfox, Wifi, NB-IoT y LoRaWAN* [3].

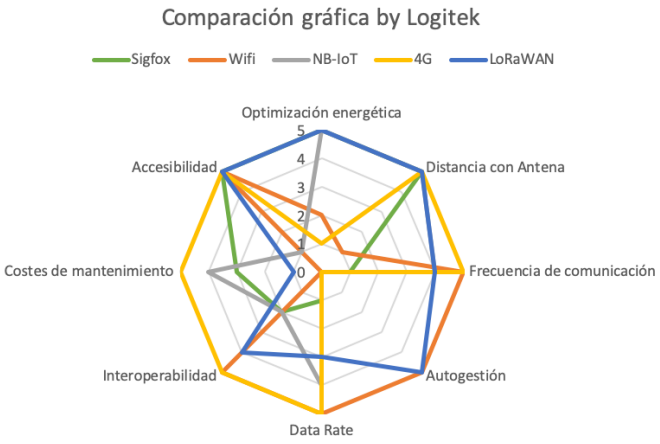


Figura 1.1: Comparativa entre distintas tecnologías inalámbricas.

1.2. Motivación

La motivación principal por la que surge este proyecto es el aportar soluciones a la hora de desarrollar alternativas de conectividad con dispositivos *Android* en zonas en las que no se dispone de cobertura tradicional, especialmente en el entorno rural.

En la actualidad existen zonas en las que, para conseguir cobertura móvil, es necesario acceder a áreas específicas y en ocasiones difíciles de alcanzar. Otros escenarios con problemas de conectividad incluyen zonas agrícolas o al aire libre en la que se realizan actividades deportivas (carreras *outdoor*, excursiones, competiciones en zonas montañosas, ...) Por ultimo, cabe mencionar que muchas veces no es necesario salir de un núcleo urbano para encontrarnos incomunicados, debido a la saturación de la red por eventos multitudinarios como conciertos o festivales.

Por estos motivos se decidió plantear este proyecto para aportar una solución a estos casos mediante el uso de una Red de bajo consumo y área extensa *(LPWAN)*[1]. Y, además, dejar abierta la puerta a que otros desarrolladores puedan aprovechar el trabajo realizado e integrar la solución en su aplicación.

[1]Low Power Wide Area Networks

1.3. Objetivos

El principal objetivo de este proyecto es crear una solución que sirva como base para que otros desarrolladores puedan incorporarla en sus aplicaciones, permitiendo así que su aplicación pueda tener conectividad en entornos donde no hay cobertura tradicional y aporte respuestas a problemas del mundo real como es la desconexión en ciertas zonas rurales de nuestra geografía.

Adicionalmente, desde un punto de vista técnico, este objetivo principal se divide en los siguientes objetivos específicos:

- Crear una solución para que la placa **ESP32** permita establecer una conexión con el dispositivo *Android* y conexiones *LoRa* contra otras placas, además de realizar la gestión y traslado de los mensajes intercambiados de una red a otra.
- Desarrollar una biblioteca *Android* ***(SDK)*** que gestione la conexión con la *ESP32* y todas las operaciones permitidas de una manera sencilla, transparente y eficiente.
- Desarrollar una **aplicación** *Android* final a modo de **Prueba de Concepto** donde se ejemplifica un posible uso de la *SDK* previamente mencionada en una aplicación real.
- **Analizar el funcionamiento** de la solución en un entorno real, y documentar las limitaciones de las conexiones *LoRa* establecidas. De esta forma se conocerían distancias máximas y degradación de la latencia.

1.4. Plan de trabajo

Para la planificación del proyecto, se ha seguido una metodología ágil. Organizándose en 4 grandes tareas denominadas *Épicas*. Son el desarrollo de este proyecto: *ESP32*, *SDK*, *App POC*[2], y finalmente, la redacción de esta memoria.

Estas épicas se han subdividido en tareas acotadas, pequeñas y más enfocadas al desarrollo, que según se identifican y se definen se registran en la lista de tareas por hacer.

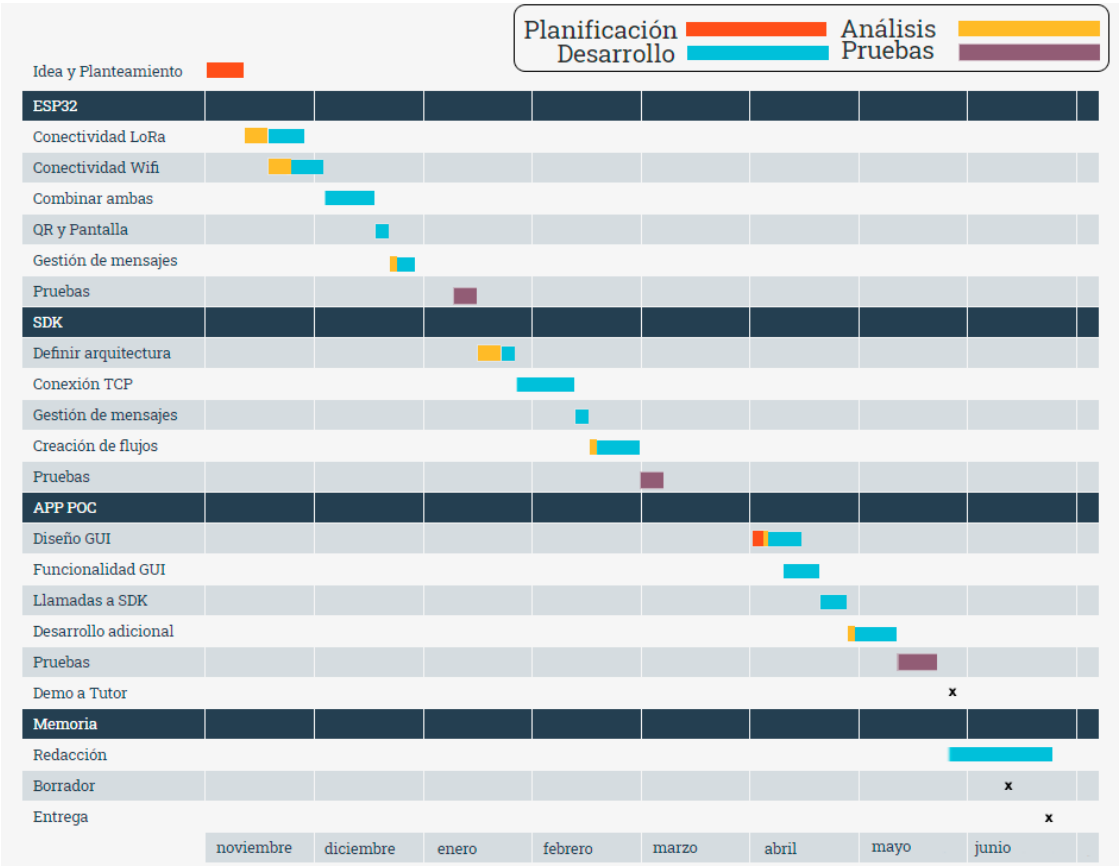


Figura 1.2: Planificación temporal del proyecto.

Temporalmente está organizado definiendo periodos de tiempo denominados *Sprints* de aproximadamente 15 días. En estos periodos de tiempo se marcaba tener finalizadas un

[2]Proof of concept, o Prueba de concepto

grupo de tareas que componen una funcionalidad, o en su defecto, si no ha sido posible, identificar las causas y volver a analizar las tareas a realizar si fuera necesario.

Adicionalmente, existen ciertas fechas claves denominadas *hitos* que marcan por ejemplo la fecha de finalización del desarrollo, la fecha de demostración del trabajo realizado al tutor, la entrega del borrador al tutor y, finalmente, la fecha de entrega de la memoria definitiva.

A la hora de llevar un registro de las tareas y en qué estado se encuentran, siguiendo los principios de la metodología *Agile*, se ha contado con el apoyo de un tablero *Kanban* de tareas. Concretamente se ha utilizado el *tablero de Github*[3] relacionado con el repositorio del proyecto.

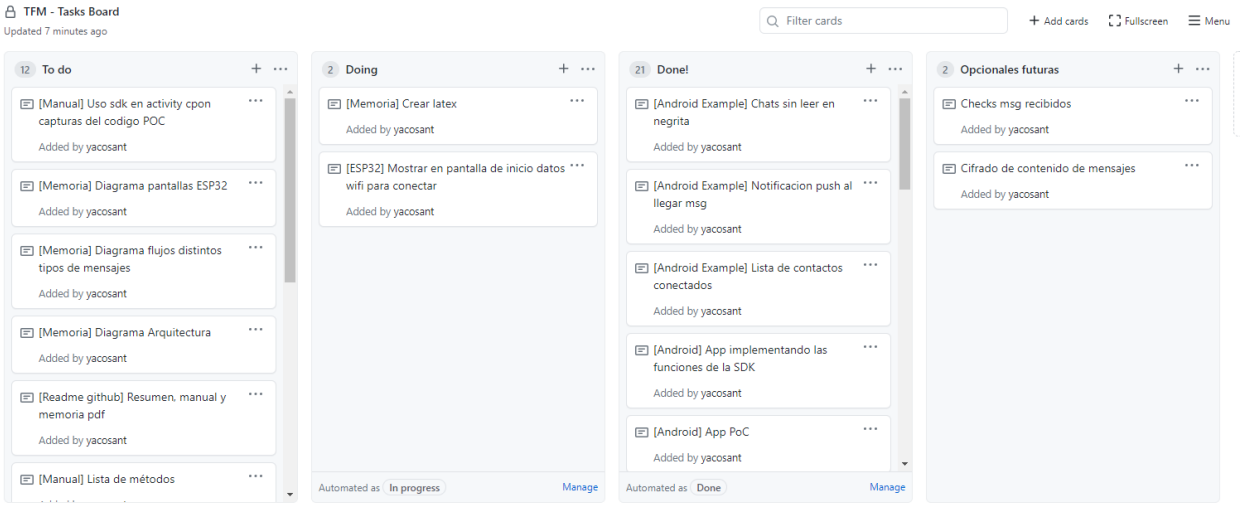


Figura 1.3: Tablero Kanban de Tareas de Github.

En este tablero se ha ido registrando las tareas a realizar *(To Do)*, las que se estaban haciendo *(Doing)*, las terminadas *(Done)* y, por último, las que quedan post puestas para trabajo futuro.

[3]https://github.com/yacosant/TFM/projects/1

1.4.1. Estructura de la memoria

Esta memoria está compuesta por trece capítulos en castellano, dos capítulos en inglés y dos anexos:

- En el **primer** capítulo se expone porqué se ha decidido llevar a cabo este proyecto, cuáles son los objetivos que se persiguen y cómo es el plan de trabajo que se ha seguido.
- En el **segundo** capítulo se desarrolla cuales son las soluciones existentes que traten de dar una solución al problema similar a la nuestra.
- En el **tercer** capítulo se indican cuales han sido las tecnologías y bibliotecas utilizadas y cuales se han descartado para implementar la solución.
- En el **cuarto** capítulo se plantea la arquitectura que se ha ideado e implementado junto con la estructura del dato principal del proyecto.
- En el **quinto** capítulo se presentan cuáles son los casos de uso que definen la funcionalidad del proyecto.
- En el **sexto**, **séptimo** y **octavo** capítulo se explica desde un punto de vista técnico cómo se estructura cada pieza de la solución, y se presenta mediante diagramas *UML*[4] cómo es su funcionamiento.
- En el **noveno** capítulo se exponen otras tres aplicaciones que se pueden llevar a cabo implementando la solución.
- En el **décimo** capítulo se plantean cuáles son las posibles mejoras que se podrían desarrollar en un futuro.

[4]Unified Modeling Language, en castellano Lenguaje unificado de modelado

- En el **undécimo** capítulo se explica el flujo de una ejecución real acompañando mediante fotografías y capturas de pantalla.
- En el **duodécimo** capítulo se recogen e interpretan los resultados obtenidos de realizar pruebas de la solución.
- En el **décimo tercer** capítulo se presentan cuales han sido las conclusiones finales de este trabajo.
- Los capítulos **primero** y **décimo tercero** tienen su versión en inglés donde se introduce el proyecto y se presentan las conclusiones respectivamente.
- En el **anexo 1** se listan las abreviaturas y términos que puedan necesitar definición.
- En el **anexo 2** se indica dónde se puede encontrar el código fuente del proyecto.

Capítulo 2. Estado del arte

En este capítulo se recopilan las distintas soluciones o desarrollos que existen actualmente de características similares a la solución que se presenta con este *TFM*.

Meshtastic

Meshtastic [4] es un proyecto que consiste en una aplicación Android y un compilado para chips dotados de conectividad *LoRa*. Se trata de una **aplicación cerrada**, que hace de app de mensajería, sin posibilidad de integrarla en otras aplicaciones. La conexión entre el dispositivo y el chip se realiza por medio de **Bluetooth Low Energy**.

Pulsartronic: DIY Smartphone LoRa connection

En el portal *Hackster.io* el usuario *Pulsartronic* tiene publicada una entrada [5] donde presenta un proyecto con el que aporta conectividad *LoRa* a un dispositivo Android conectando el chip por medio del **puerto microusb** del móvil. También se trata de una aplicación Android **cerrada**, con una única ventana de **chat común** y una interfaz poco amigable. Para la parte *Lora* requiere la **impresión** de un circuito **PCB** diseñado *ad hoc* para este propósito y su correspondiente ensamblaje.

Adicionalmente, he podido observar que la solución tiene ciertos problemas de

compatibilidad ya que falla en *Android 4.4.2*, según indica un usuario en el *Github*[1] del proyecto.

Skrypt

Skrypt [6] es un proyecto personal presentado en el portal *Hackaday.com* que consiste en el diseño y la implementación de un circuito impreso (**PCB**) que transmita por *LoRa* los mensajes que le llegan mediante **Bluetooth Low Energy**.

Este proyecto lleva tiempo **sin actualizaciones**, y se trata exclusivamente de la parte **hardware**, del cual está disponible el esquema del circuito, pero no está publicado ningún tipo de código.

LoRa-based Device-to-Device Smartphone Communication for Crisis Scenarios

Este proyecto es una publicación [7] donde se presenta la conexión de un dispositivo Android mediante *LoRa* por medio de una *ESP32* para escenarios catastróficos. Se trata de una aplicación de **cerrada** de mensajería instantánea, y permiten establecer la conexión a la *ESP32* por medio de puerto de serie, por medio de *BLE*, o por medio de Wifi.

[1] https://github.com/pulsartronic/USBRFMApp/issues

2.1. Qué aporta esta solución respecto a las demás

La principal diferencia que marca esta solución con respecto a las soluciones que se han comentado anteriormente es la **reusabilidad**.

No se trata de una implementación final con un único objetivo, si no que es una solución generalista que, mediante su uso, permite dotar a cualquier aplicación en Android de la capacidad de comunicación de larga distancia por medio de la red *LoRa*.

Además, al no estar ligada a una implementación concreta del hardware mediante impresiones de circuitos integrados permite su utilización en una amplia variedad de dispositivos *ESP32*, lo cual lo hace más accesible y disminuye mucho el coste destinado al elemento hardware.

En este desarrollo, tanto en la parte del hardware como en la del dispositivo móvil se ha perseguido prestar un servicio a la comunidad de desarrolladores para que puedan hacer una implementación con conectividad de larga distancia de la forma más transparente y sencilla posible.

Capítulo 3. Tecnologías

Para la implementación de la idea inicial de este proyecto se ha tenido que elegir entre distintas tecnologías. En este capítulo se listan tanto tecnologías de comunicación, lenguajes y bibliotecas *software* y elementos *hardware* empleados y descartados.

Estas son las tecnologías que finalmente fueron utilizadas:

3.1. LoRa

Figura 3.1: LoRa

Esta tecnología es el pilar fundamental de este TFM. LoRa[1] es una red *LPWAN*: Low-Power Wide Area Network (red de bajo consumo y área extensa). LoRa cuenta con grandes ventajas para proyectos de *IOT* (Internet of Things), con dispositivos alimentados por baterías.

Utiliza una modulación que se llama ***Chirp Spread Spectrum***, o *CSS*, y se usa en comunicaciones militares y espaciales desde hace décadas.

Su característica más valiosa es que permite un gran alcance: de 10 a 20km en entorno rural y de 3 a 5km en entorno urbano [8].

[1] https://www.semtech.com/lora

3.2. ESP32

La ESP32[2] es una familia de chips *SoC*[3] de bajo coste y bajo consumo de energía, con tecnología Wi-Fi, Bluetooth, doble núcleo de 32bits a (160Mhz-240Mhz).
Pueden ser programadas en varios lenguajes, siendo los principales *C++* y *MicroPython*.

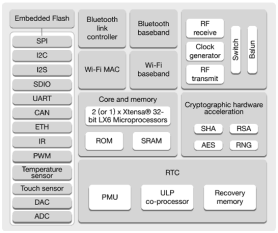


Figura 3.2: Chipset ESP32

3.2.1. TTGO Lora32 Oled v2

Existen numerosos chips que utilizan el SoC ESP32. En mi caso, he decidido utilizar la implementación *TTGO Lora32 Oled v2*, que aporta conectividad *LoRa* y una pequeña pantalla *OLED* de 1 pulgada.

Esta placa tiene integrado el *Transceptor*[4] *LoRa*, en concreto un modelo *Semtech SX1276* [9], que es configurable en cualquiera de sus frecuencias. *868 MHz* para Europa, *915 MHz* para América del Norte o *433 MHz* para Asia.

3.3. Protocolo TCP

TCP: Transmission Control Protocol, es un protocolo que actúa en la capa de transporte del modelo OSI[5].

Aporta la característica de que la comunicación a través de la red sea confiable,

[2]http://esp32.net/
[3]System on a Chip
[4]Dispositivo que cuenta con un transmisor y un receptor
[5]https://www.cloudflare.com/es-es/learning/ddos/glossary/open-systems-interconnection-model-osi/

ya que asegura que los datos que emite el cliente sean recibidos por el servidor sin errores o perdidas y en el mismo orden que fueron emitidos. De esta forma se garantiza que todos los paquetes entre el móvil y la *ESP32* lleguen a su destino.

3.4. Arduino IDE

Figura 3.4: Arduino

Arduino es una compañía y comunidad enfocada en el diseño y creación de placas de desarrollo hardware y software libre para el desarrollo de soluciones que interactúan con el mundo real. En este proyecto se ha contado con *Arduino IDE*[6], como entorno de desarrollo para la codificación, compilación y volcado del desarrollo en la *ESP32*.

La codificación se realizó en *C++* ya que *Arduino* es una adaptación de *C++*, se compila con *GCC*. He utilizado las siguientes bibliotecas de *Arduino* que me han facilitado o permitido ciertas funcionalidades:

3.4.1. Bibliotecas WiFi

Para el uso y la gestión de la conexión Wifi se utilizan las bibliotecas **WiFi.h**[7] y sus asociadas. Permiten la utilización de la antena Wifi, la creación de un punto de acceso Wifi en la *ESP32*, seguridad *WPA2* y gestión de las conexiones establecidas.

3.4.2. Bibliotecas LoRa

Para poder utilizar la antena *LoRa* son necesarios dos bibliotecas:

[6]https://www.arduino.cc/en/software
[7]https://www.arduino.cc/en/Reference/WiFi

- **SPI.h**[8] Empleada para la configuración de los puertos de la antena bajo el protocolo de transferencia *SPI (Serial Peripheral Interface)*.
- **LoRa.h**[9] Permite el envío y recepción de paquetes *LoRa* por medio de la antena.

3.4.3. ArduinoJson

ArduinoJson.h[10] es un biblioteca que facilita la Serialización y Deserialización de cadenas de texto con formato *json* de una manera sencilla y eficiente.

En este desarrollo se emplea para recuperar los campos de los mensaje recibidos por *LoRa* y validar el tipo de mensaje y si es el destinatario de dicho mensaje.

3.4.4. Bibliotecas pantalla OLED

Para poder aprovechar la pantalla oled que integra la *TTGO Lora32 Oled v2* se han utilizado las siguientes bibliotecas:

- **Wire.h**[11] Permite la configuración y comunicación por medio del puerto y protocolo de comunicación serie **I2C** *(Circuito inter-integrado)*.
- **Adafruit_GFX.h**[12] y **Adafruit_SSD1306.h**[13] Librerías desarrolladas por *Adafruit* empleadas para mostrar textos y dibujar en la pantalla.
- **qrcode.h**[14] de *Richard Moore*. Se ha empleado para realizar la transformación de cadena de texto a *array* que representa el código QR a pintar por pantalla. En este

[8] https://www.arduino.cc/en/reference/SPI
[9] https://www.arduino.cc/reference/en/libraries/lora/
[10] https://arduinojson.org/
[11] https://www.arduino.cc/en/reference/wire
[12] https://github.com/adafruit/Adafruit-GFX-Library
[13] https://github.com/adafruit/Adafruit_SSD1306
[14] https://www.arduinolibraries.info/libraries/qr-code

caso, se ha empleado para transformar las credenciales de acceso al punto Wifi de la *ESP32* de manera rápida y sencilla.

3.5. Android

Figura 3.5: Android

Android[15] es el sistema operativo para dispositivos móviles por excelencia. Es de Google y está basado en el *kernel* Linux.

Para poder implementar la *SDK* y, posteriormente, la aplicación *POC* fue necesario utilizar las siguientes herramientas de desarrollo que nos proporcionan desde el equipo de Google:

3.5.1. Android Studio

Se trata del entorno de desarrollo, basado en otro conocido IDE, *IntelliJ*. Permite la edición del código, el diseño de las interfaces y elementos visuales y la compilación o depuración del desarrollo. Adicionalmente, integra el uso de *avd* y *adb* mediante interfaz gráfica. Desarrollar en Android supuso un reto, pero finalmente la experiencia fue muy positiva.

3.5.2. Android Vitual Devices (avd)

El Administrador de Dispositivos Virtuales permite la creación de dispositivos virtuales, eligiendo, entre otras cosas, el dispositivo, las configuraciones del hardware y la

[15]https://developer.android.com/

versión del sistema operativo. De esta manera se puede lanzar la aplicación, realizar pruebas y depurar sin necesidad de conectar ningún dispositivo físico.

Se le dio especial uso a estos dispositivos emulados en la parte inicial del desarrollo, cuando no estaba tan avanzado y aún no existía la necesidad de conectar con la *ESP32*.

3.5.3. Android Debug Bridge (adb)

El ADB es una herramienta que establece una comunicación con un dispositivo y permite realizar una variedad de acciones en él, como instalar y depurar apps, y ejecutar distintos comandos de manera remota. Esta herramienta es la que emplea por detrás Android Studio para realizar la depuración. Tanto en dispositivos virtuales como en dispositivos físicos.

3.6. Java

Java[16] es un lenguaje multiplataforma que permite entre otras cosas, el paradigma de programación orientada a objetos.

Figura 3.6: Java

Es un lenguaje con el cual se ha desarrollado la parte en Android del proyecto.

[16]https://www.java.com/es/

3.7. Github

Figura 3.7: Github

Github[17] es una plataforma de desarrollo colaborativo de *software* y *hosting* de proyectos. La plataforma está basada en Git, por lo cual es una herramienta perfecta para hacer control de versiones. Todas las actualizaciones del código se han ido subiendo a un repositorio (B).

3.8. Tecnologías descartadas

Del mismo modo que en la sección anterior, en esta se van a listar las tecnologías que fueron descartadas a la hora de dar forma a la implementación del proyecto.

3.8.1. Otras LPWAN

Se barajaron otras tecnologías *LPWAN* como ***SigFox*** o ***NB-IOT***. Cada una de estas tienen sus fortalezas y sus debilidades, pero en este caso primaba más la mejor gestión de energía, el menor coste hardware que tenga que desembolsar el usuario final, y un buen rango de cobertura. Finalmente, la seleccionada fue ***LoRa*** al ser la tecnología que mejor se adaptaba a estos requisitos.

3.8.2. Protocolo UDP

A la hora de abrir un socket entre el dispositivo Android y la *ESP32* el protocolo UDP quedó descartado frente al TCP debido a que no está orientado a conexión y no

[17]https://github.com/

garantiza la entrega de los paquetes.

3.8.3. MicroPython

A la hora de codificar el desarrollo de la *ESP32* se decidió utilizar *Arduino* debido a que es un lenguaje compilado que se *flashea* en el chip, es más eficiente, tiene un mayor rendimiento y contaba con las bibliotecas necesarias.

En contraposición, *MicroPython* es un lenguaje interpretado, con una menor eficiencia, el código fuente se almacena en la memoria del dispositivo y ciertos requisitos del desarrollo eran más complejos de implementar o no quedaban totalmente cubiertos.

3.8.4. Kotlin

Kotlin es el nuevo lenguaje de Google para, principalmente, la programación de dispositivos Android. Pese a su creciente popularidad y su aparente flexibilidad, al no contar con experiencia previa y dado que no es el objetivo del Master ni del TFM emplear este lenguaje, fue descartado.

Se prefirió aprovechar los conocimientos disponibles de *Java*. De esta manera se evitaba la posibilidad de incumplir los tiempos marcados e imprevistos que pudieran surgir.

3.8.5. Conectividad Bluetooth con la ESP32

Se evaluaron dos medios para conectar el dispositivo con la *ESP32*: *Bluetooth* o *Wifi*.
En el caso del *Bluetooth*, podía conectarse por medio de conexión en serie, o mediante la lectura y escritura de Tablas Gatt. Pero debido a las limitaciones de tamaño de los mensajes, latencias, complejidad y menor cobertura el *Bluetooth* fue descartado.

Capítulo 4. Arquitectura

En este capítulo se presenta cómo se organiza la arquitectura general del proyecto, en cuanto a conectividad, estructura y funcionalidades.

4.1. Arquitectura global del proyecto

Como se ve reflejado en la figura 4.1, este proyecto, nombrado **LoraConnect** está compuesto por varios dispositivos y desarrollos.

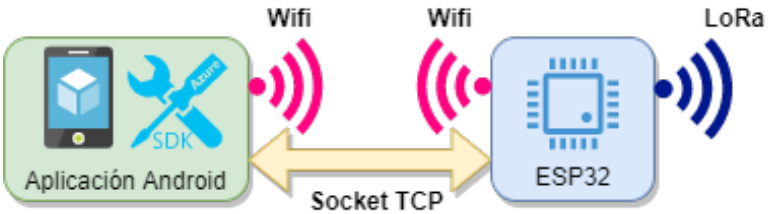


Figura 4.1: Arquitectura Global.

- **Dispositivo Android** donde se ejecuta la aplicación.
  - Aplicación que utiliza el usuario, donde se visualizan y/o envían los mensajes.
  - SDK que se integra dentro de la aplicación para hacer un uso transparente de la conexión con la *ESP32* por medio del *socket TCP* que se establece.
- **ESP32** que realiza la función de puente entre el dispositivo Android y la red *LoRa*.

4.2. Despliegue de la solución

En un entorno real el despliegue estaría compuesto de N conjuntos interconectados entre sí por medio de la red *LoRa*.

Tal y como se puede observar en la figura 4.2, cada uno de los usuarios contarán con un dispositivo móvil *Android* y un dispositivo *ESP32*.

Los dispositivos Android se conectan a través de Wifi y realizan el intercambio de datos por medio de un *socket TCP* contra la *ESP32*. Y esta al mismo tiempo, realiza el intercambio de paquetes con los otros usuarios conectados a través de la red *LoRa*.

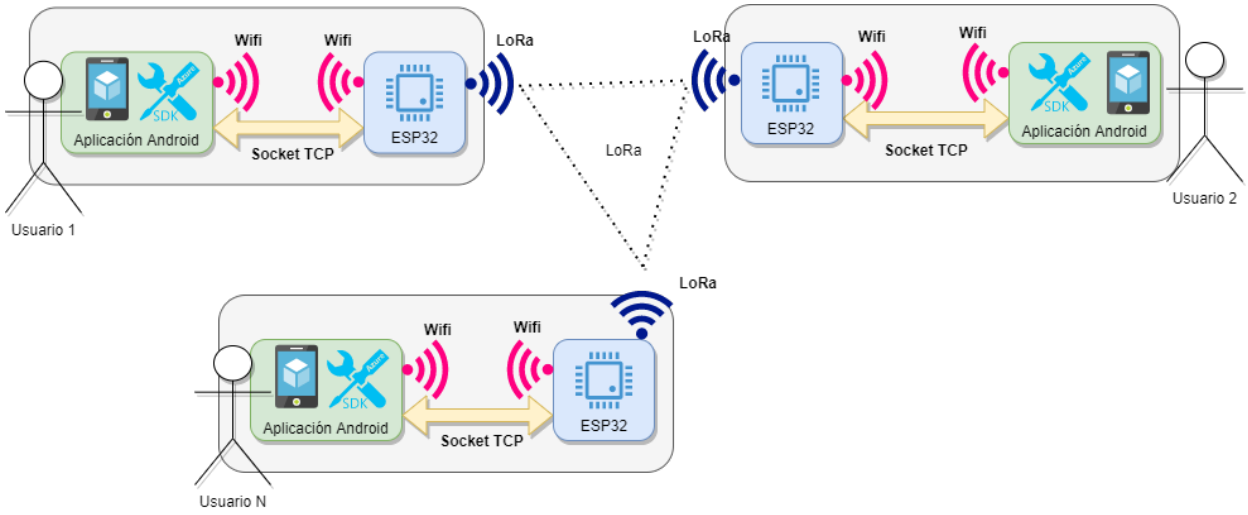


Figura 4.2: Representación de despliegue real.

El tipo y contenido de ese mensaje se origina en la aplicación del usuario origen, en la *SDK* se crea y se envía a la *ESP32*, desde la cual se transmite por medio de *LoRa*. Tras transmitirse, llega a la *ESP32* de destino, que lo recibirá y transmitirá a la *SDK* para que notifique a la aplicación del usuario destino.

4.3. Estructura del mensaje

A lo largo de todo esta ejecución en la que intervienen distintos componentes hay algo que es común en todos ellos. Esto es **el mensaje**.
Esta entidad es trasversal a los distintos componentes y su estructura está compuesta por los siguientes campos:

- **op** (short): Este campo indica cuál es el tipo del mensaje.
- **timestamp** (String): Este campo contiene el Timestamp en el que se creó el mensaje en formato *Unix Epoch* (*ms* desde la medianoche del 01/01/1970).
- **origen** (String): Este campo indica el nombre de usuario de quién envía el mensaje.
- **destino** (String): Este campo indica el nombre de usuario al que va dirigido el mensaje.
- **mensaje** (String): Contenido del mensaje. Este campo es opcional según el tipo.

Tipos de mensaje

Para diferenciar entre las distintas funciones que puede tener un mensaje se utiliza el campo ***op***. Este campo puede tomar 5 valores:

- **[0] Mensaje**: Este tipo indica que el mensaje lleva un contenido, un origen y un destino concreto. Son los únicos mensajes que se lanzan bajo la decisión del usuario.
- **[1] Test**: Tipo de mensaje creado con el objetivo de poder realizar pruebas automáticas con el objetivo de medir la latencia de la red, registrar los resultados y posteriormente recuperarlos para analizarlos. Se indica origen, destino y contenido.

- **[2] Hello ACK**: Este tipo de mensaje es el que los usuarios conectados responden automáticamente al origen del mensaje de *Tipo Hello* para informarle de que ellos también están en linea. Este mensaje tiene origen y destino concreto. Sin contenido.

- **[3] Hello**: Este tipo de mensaje se envía automáticamente con el objetivo de indicar que se acaba de conectar a la red. Tiene un origen concreto pero no un destino concreto *(broadcast)* con el objetivo de llegar a todos los usuarios conectados. No lleva contenido.

- **[4] Bye**: Tipo de mensaje que se envía automáticamente cuando el usuario se desconecta de la red. Consta de origen pero no de destino *(broadcast)* con el objetivo es notificara todos los usuarios conectados de la desconexión. No porta contenido.

El envío tanto por Wifi a la *ESP32*, como por *LoRa* al destinatario final se realiza por medio de un texto. Para transformar la estructura de mensaje a un texto se convierte en un *json* minimizado con la intención de reducir al mínimo la longitud del texto a enviar y aprovechar al máximo el ancho de banda. Este *json* tiene el aspecto indicado en el cuadro 4.1.

```
{
    "op":0,
    "o":"Yaco",
    "d":"Fran",
    "t":"1624622400000",
    "msg":"Mensaje de Ejemplo"
}
```

Cuadro 4.1: Ejemplo de *Json minimizado* generado de un Mensaje.

Capítulo 5. Casos de uso

En este capítulo se presentan los distintos casos de uso que se presentan a la hora de utilizar la *SDK* y ser implementados en la aplicación final.

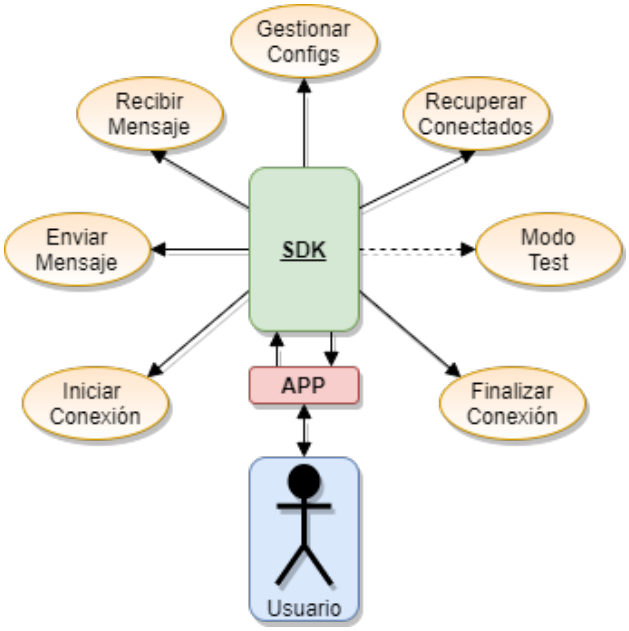


Figura 5.1: Casos de Uso.

5.1. Caso de uso: Conexión

En este caso lo que se hace es establecer la conexión del dispositivo *Android* contra el chip *ESP32*. Esta conexión se realiza por medio de un *socket TCP* punto a punto.

Tras abrir la conexión contra la ESP32, se envía un mensaje a modo de *broadcast* de tipo *HELLO*, de forma que se notifica a todos los usuarios que nos hemos conectado y quedamos a la espera de su respuesta mediante un mensaje de tipo *HELLO_ACK* para descubrir quienes están conectados.

5.2. Caso de uso: Envío de mensaje

Una vez se ha establecido la conexión es posible realizar el envío de mensajes. Los mensajes que se envían desde el dispositivo *Android* utilizan el método expuesto por la *SDK*. Mediante este método se envía a través del *socket TCP* a la *ESP32* y desde ella da el salto a la red *LoRa*.

5.3. Caso de uso: Recepción de mensaje

De la misma manera que expone la funcionalidad de enviar mensajes, la *SDK* también gestiona la recepción de mensajes y lo notifica a la aplicación.
Al recibir un mensaje por medio de la red *LoRa* la *ESP32* evalúa si va dirigido a nosotros, y de ser así se transmitirá el mensaje al dispositivo *Android* por medio del *socket TCP*.

5.4. Caso de uso: Recuperación de usuarios conectados

Un dato interesante e importante para la gestión de las acciones disponibles en la Aplicación *Android* es consultar a la *SDK* qué usuarios están disponibles, esto nos permite el saber si un usuario está conectado actualmente y podemos enviarle un mensaje.
Esta funcionalidad devuelve la lista de usuarios conectados.

5.5. Caso de uso: Gestión de configuraciones

Se ofrecen la opción de definir configuraciones, como un nombre de usuario, con garantías de persistencia, y la opción de poder recuperarlo siempre que se quiera.

Tal y como se ha comentado en la recepción de mensajes, la redirección de los mensajes de la red *LoRa* a los dispositivos *Android* se realiza en base a los nombres de usuarios. También gestiona el *flag* que indica si se encuentra el modo de *test* activado.

5.6. Caso de uso: Desconexión

Del mismo modo que se ofrece la posibilidad de establecer la conexión, se ofrece el poder realizar la desconexión de la red.
Mediante esta funcionalidad la *SDK* lanza un mensaje de tipo *BYE* a modo de *broadcast* que es transmitido a la red *LoRa* para notificar nuestra desconexión al resto de usuarios y, posteriormente, terminar la conexión *TCP* contra la *ESP32*.

5.7. Caso de uso: Modo Test

Esta funcionalidad de Modo Test no está orientada al uso normal de la SDK, pero se ha creado de cara a poder evaluar los resultados ofrecidos por la solución.

Su objetivo consiste en realizar una batería automatizada de envío de mensajes de tipo *TEST* para realizar el cálculo de las latencias de la red en distintos escenarios y con distintas distancias entre dispositivos.

De esta forma se obtienen datos precisos sobre el rendimiento de la solución y se pueden interpretar, como se ha hecho en al sección de *Resultados* (12).

Capítulo 6. Desarrollo ESP32

A lo largo de este capítulo se detalla todo lo referente al desarrollo de la parte de la solución referente a la *ESP32*.

6.1. Propósito del desarrollo

Este componente del proyecto es el encargado de realizar el envío y la recepción de mensajes por medio de la red **LoRa**. Estos mensajes serán enviados desde o hacia el dispositivo móvil *Android* que estará conectado a la *ESP32*.

Para llevar a cabo estas tareas el código cuenta con las siguientes funciones, tipo de dato y bucle principal.

6.2. Funciones y Datos

- ***void setup():*** Es la función encargada de inicializar todo lo necesario cuando se enciende. Realiza las llamadas a todas las funciones de *setup* de cada componente. Esta función es nativa de *Arduino* y se ejecuta al arrancar.
- ***void setup_ Serial():*** Realiza la inicialización de la comunicación por el puerto de serie contra la consola de logs de Arduino IDE. Se configura a una velocidad de transferencia de *115200 baudios* (*bits/s*).

- ***void setup_ Display():*** Establece las configuraciones de la pantalla oled, configurando los pines de la pantalla.

- ***void setup_ Wifi():*** En esta función se levanta el punto de acceso Wifi y se arranca el servidor encargado de gestionar la conexión con el cliente que se conecte.

- ***void setup_ Lora():*** Configura los pines de la antena, la conexión por *SPI* y establece en qué frecuencia se va a trabajar. La utilizada es *868Mhz*, la permitida en Europa.

- ***void drawQR():***Es la función encargada de llamar a la biblioteca *qr.h*, obteniendo como resultado un *array* que pinta por pantalla con mediante 2 bucles.

- ***void printLcd(char* text, int size, bool clear, int x, int y):*** Función creada para poder pintar por pantalla con las características que mejor vienen en cada caso. Para ello se le pasa el texto a pintar, y como argumentos opcionales, el tamaño que se le quiere poner al texto, se indica si se quiere borrar lo que está pintado ya en pantalla, y las coordenadas donde colocar el principio del texto.

- ***void deserializeObject(String json):*** Función que recibe un mensaje en formato *String* y hace uso de la biblioteca *ArduinoJson.h* para obtener un *StaticJsonDocument* que posteriormente se parsea dentro de un objeto *Mensaje* con los diferentes atributos.

6.2.1. Clase Mensaje

Esta clase está definida en el archivo *Utils.h* e implementada en el *Utils.cpp*. Esta clase tiene los campos referentes a un mensaje y las siguientes funciones:

- ***void Mensaje::parse(StaticJsonDocument<300>doc):*** Obtiene del *StaticJsonDocument* cada uno de los campos del mensaje recibidos.

- ***getters:*** Recuperan los valores de cada uno de los atributos.

6.3. Bucle Principal

Los desarrollos en *Arduino* tienen 2 funciones necesarias nativas. Una es ***setup()***, vista previamente y otra es ***loop()***. En la función *loop()* se ejecuta en bucle y en ella es donde transcurre la lógica principal.

Para un mejor entendimiento se presenta un diagrama de flujo en la figura 6.1.

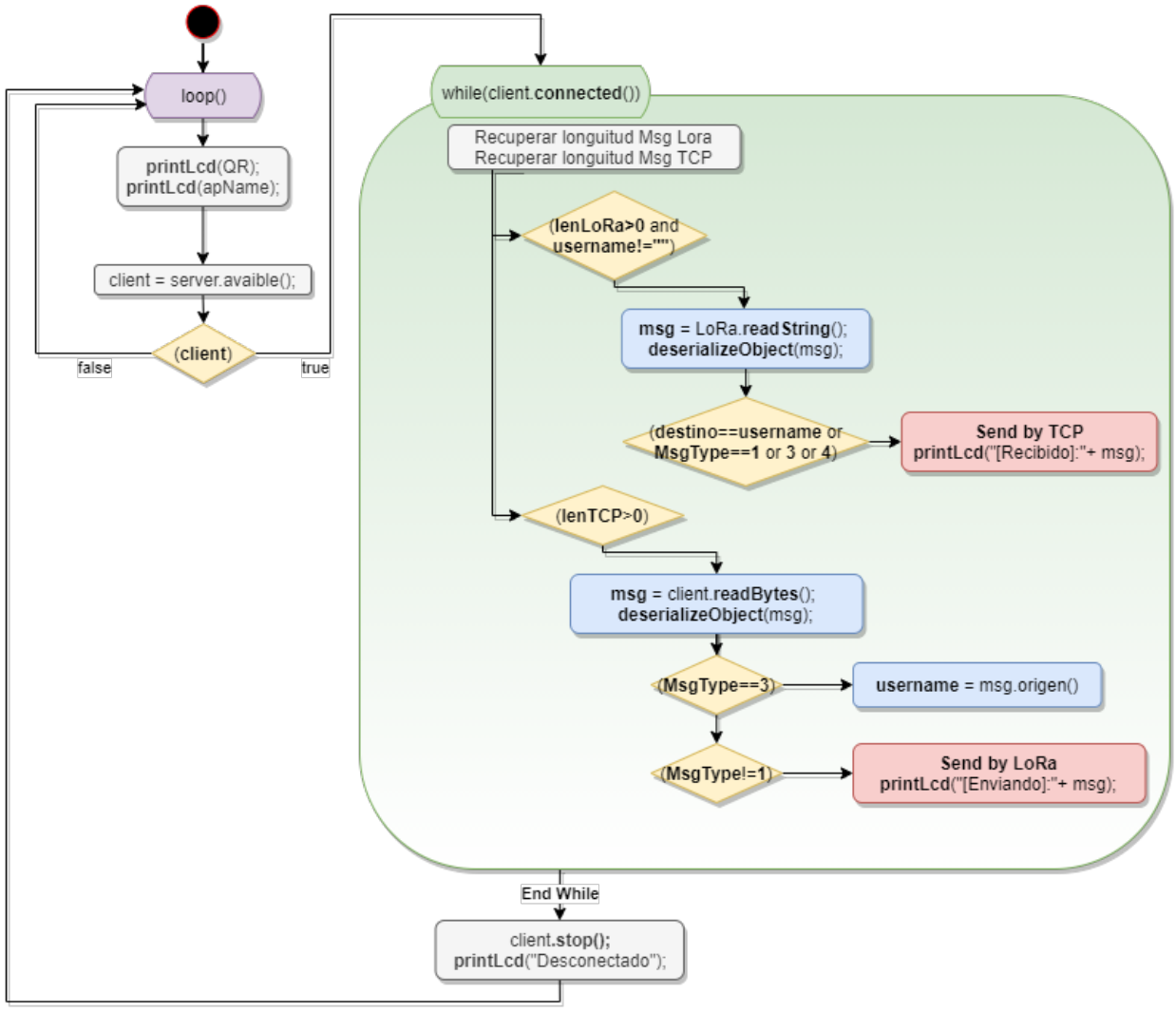


Figura 6.1: Ejecución del Bucle Principal.

Capítulo 7. Desarrollo SDK

7.1. Propósito del desarrollo

Este componente es la biblioteca que se encarga de actuar como intermediario entre la aplicación Android final y la *ESP32*. Su propósito es el establecer la conexión con el chip por medio de un *socket TCP*, gestionarla, y ofrecer de la forma más transparente posible la capacidad del envío y al recepción de mensajes a la aplicación final que la integre.

7.2. Arquitectura

La Figura 7.1 muestra, esquemáticamente, la arquitectura de la biblioteca, cuyos componentes se detallan a continuación:

- Paquete **controller:** Conformado por la clase *ControllerConfig* encargada de gestionar lo relacionado a las configuraciones como el nombre de usuario o el flag del modo test, la clase *ControllerTcp* con la que se realiza la conexión a la *ESP32*, y la clase *ControllerTest* que gestiona las pruebas para obtener estadísticas de las latencias.
- Paquete **data:** Contiene el modelo de dato *Message* con el cual se trabaja.
- Paquete **utils:** Recoge las utilidades como son las constantes definidas en *Constants*.
- Clase **LoraConnect:** Es la clase principal que expone todos los métodos de la *SDK*.

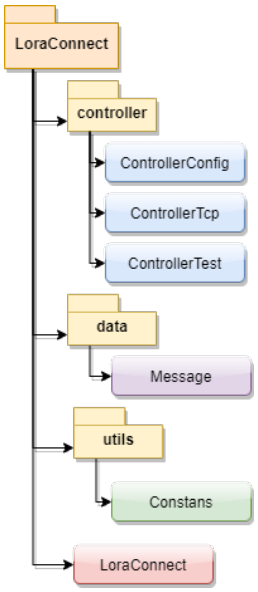


Figura 7.1: SDK: Paquetería del proyecto.

7.3. Métodos expuestos por la SDK

Todos los métodos son estáticos, de forma que se pueden invocar desde cualquier parte del proyecto donde se use la SDK sin instanciar ningún objeto de la clase *LoraConnect*. Esto también permite que la conexión sea única y no se establezca una conexión por cada objeto que se instancie.

Los métodos expuestos por la *SDK* son los siguientes:

- ***void init(Context context):*** Es la función que se invoca en primer lugar. Su funcionalidad consiste en inicializar la instancia de la *SDK*, pasándole el contexto de la aplicación Android para poder interactuar con ella.
- ***void connectToESP32(OnMessageReceived listener):*** Una vez hecha la inicialización con el método anterior, se puede establecer la conexión con la *ESP32*. Para ello se invoca a este método pasándole el *listener* de tipo *OnMessageReceived* que se ha definido en la aplicación con el objetivo de notificarle la llegada de mensajes.

Al invocar a esta función, tras establecer la conexión, se envía automáticamente un mensaje de tipo *Hello* para notificar la unión a la red *LoRa*.

- ***String getUsername():*** Este método recupera el nombre de usuario que tenemos configurado. Si es la primera vez que se invoca a este método, recupera la información de la configuración persistente. Este *username* es el que figurará cuando enviemos mensajes, y será el que tendrán que utilizar los otros usuarios para hacernos llegar un mensaje.

- ***void setUsername(String username):*** Este método sería utilizado a la hora de establecer o actualizar el nombre de usuario que se quiere utilizar. Se almacena en la configuración persistente.

- ***Message sendMessage(String destination, String msg):*** Este método es el que se debe llamar si se quiere enviar un mensaje. Se le debe de pasar el destinatario y el contenido del mensaje a enviar. Los demás campos como *origen*, *timestamp*, y *tipo* de mensaje se setearán automáticamente de modo que sea lo más transparente y sencillo.

- ***ArrayList<String>getConnected():*** Llamando a este método se obtiene la lista de los usuarios que se encuentran conectados actualmente. Esta lista se gestiona y se va rellenando y vaciando sola según van llegando los mensajes de tipo *Hello* o de *Bye*.

- ***Message makeTest(String destination, String params):*** Este no es un método con un objetivo funcional como tal. La creación de este método y todo el flujo que desencadena su llamada surge de la necesidad de tener una manera metódica de conocer cuál es el índice de paquetes perdidos y cuales son las latencias de los mensajes.

  Este método recibe el destinatario contra quien iniciar el test y unos parámetros: duración del test y frecuencia de envío de los mensajes expresados en segundos y separados por un *;*.

Tras su acabar su ejecución se vuelcan en un archivo *.txt* almacenado en el dispositivo *Android* de destino con las latencias obtenidas y al latencia media.

- ***void disconnect():*** Este método es el que se ha de llamar cuando llegue el momento en el cual el usuario quiera cerrar la aplicación o desconectarse. Con esta simple llamada, antes de cerrar la conexión con la *ESP32*, se genera el envío automático de un mensaje de tipo *Bye* con el cual se va a notificar que se abandona la red *LoRa*.

7.4. Diagramas

7.4.1. Diagrama de clases

Mediante el diagrama de la figura 7.2 se representan las clases y sus respectivos atributos y métodos que conforman el código de la *SDK*.

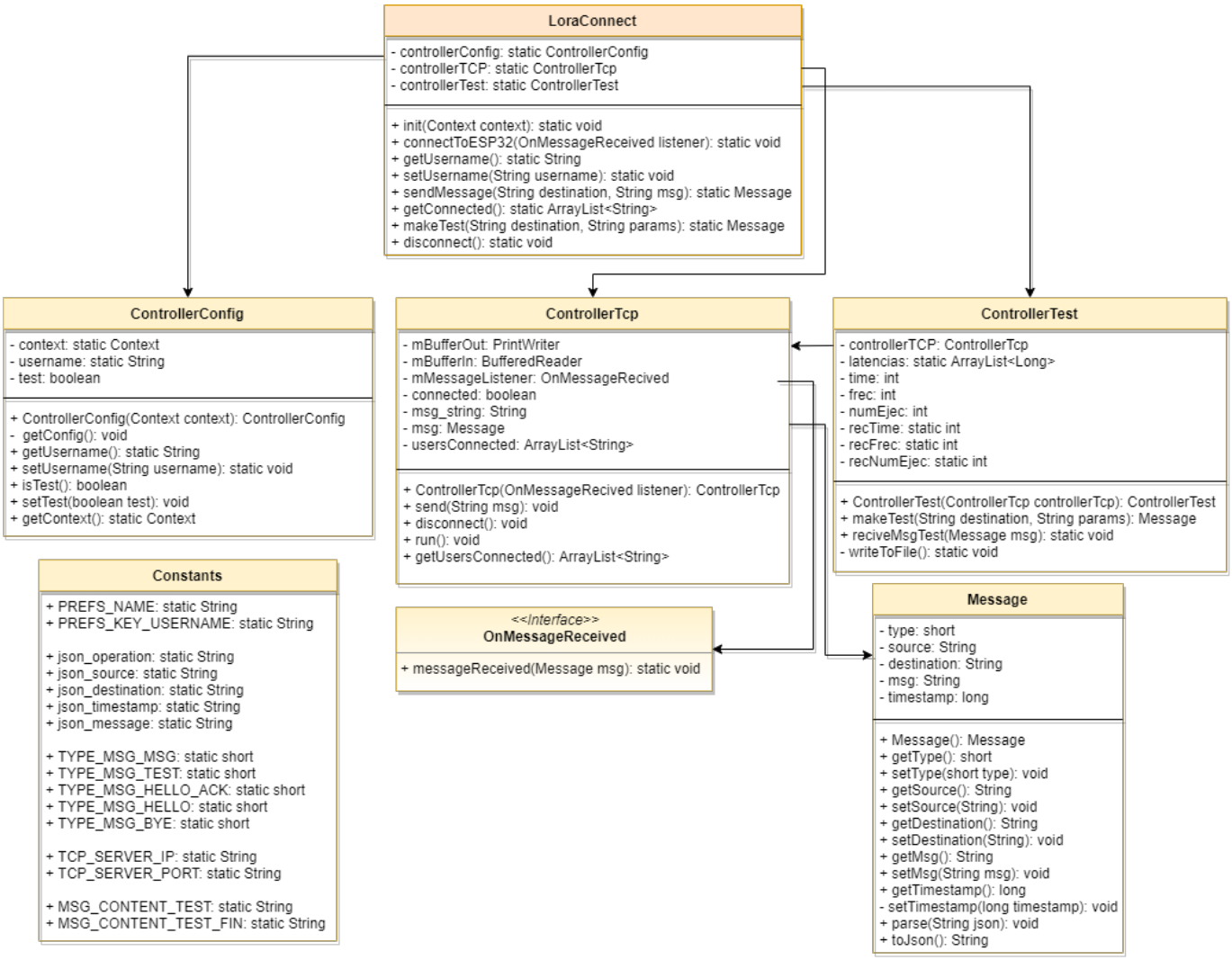


Figura 7.2: Clases del proyecto de la SDK.

7.4.2. Diagramas de Flujo

A lo largo de esta sección se va a explicar con la ayuda de diagramas de flujo y sus anotaciones el funcionamiento de cada una de los métodos expuestos por la *SDK* que se han descrito previamente en el apartado 7.3.

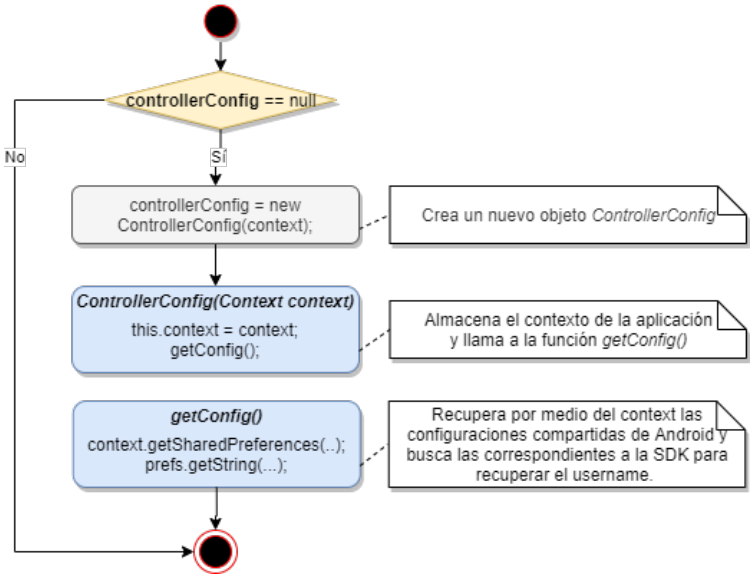


Figura 7.3: Flujo **init(Context context)**.

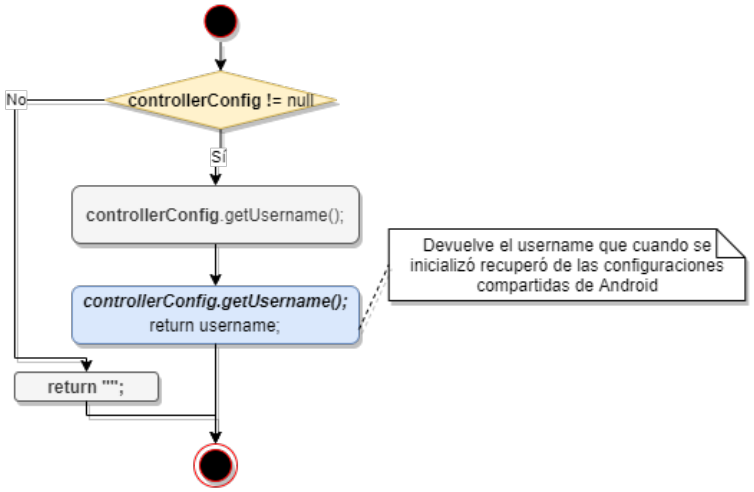


Figura 7.4: Flujo **getUsername()**.

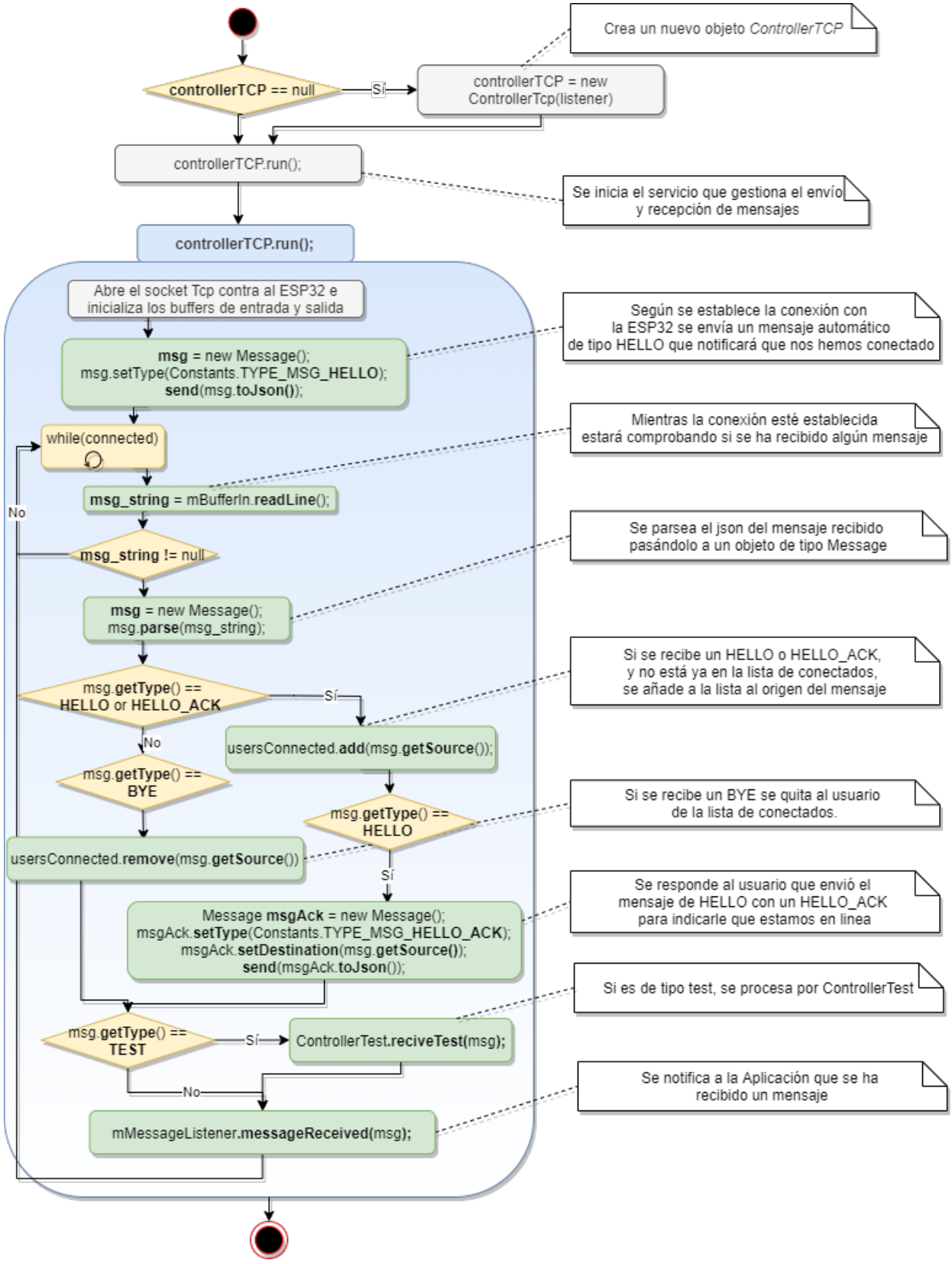


Figura 7.5: Flujo **connectToESP32(OnMessageReceived listener)**.

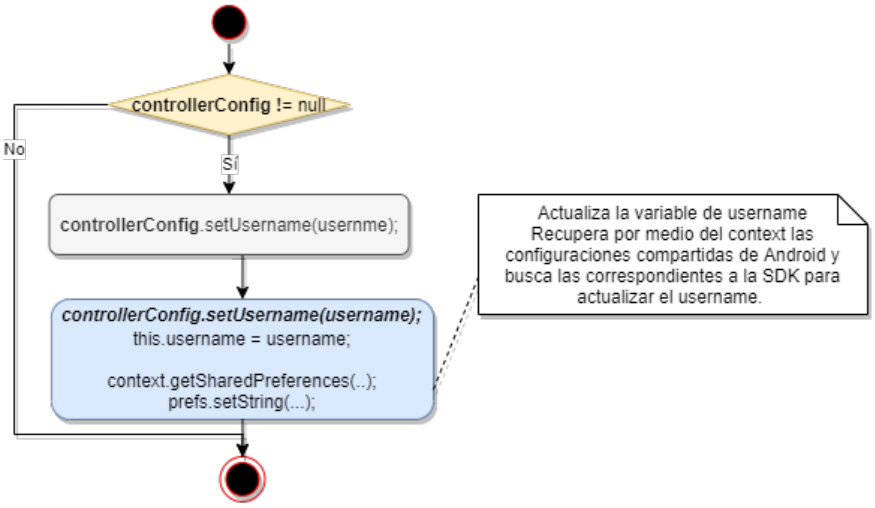


Figura 7.6: Flujo **setUsername(String username)**.

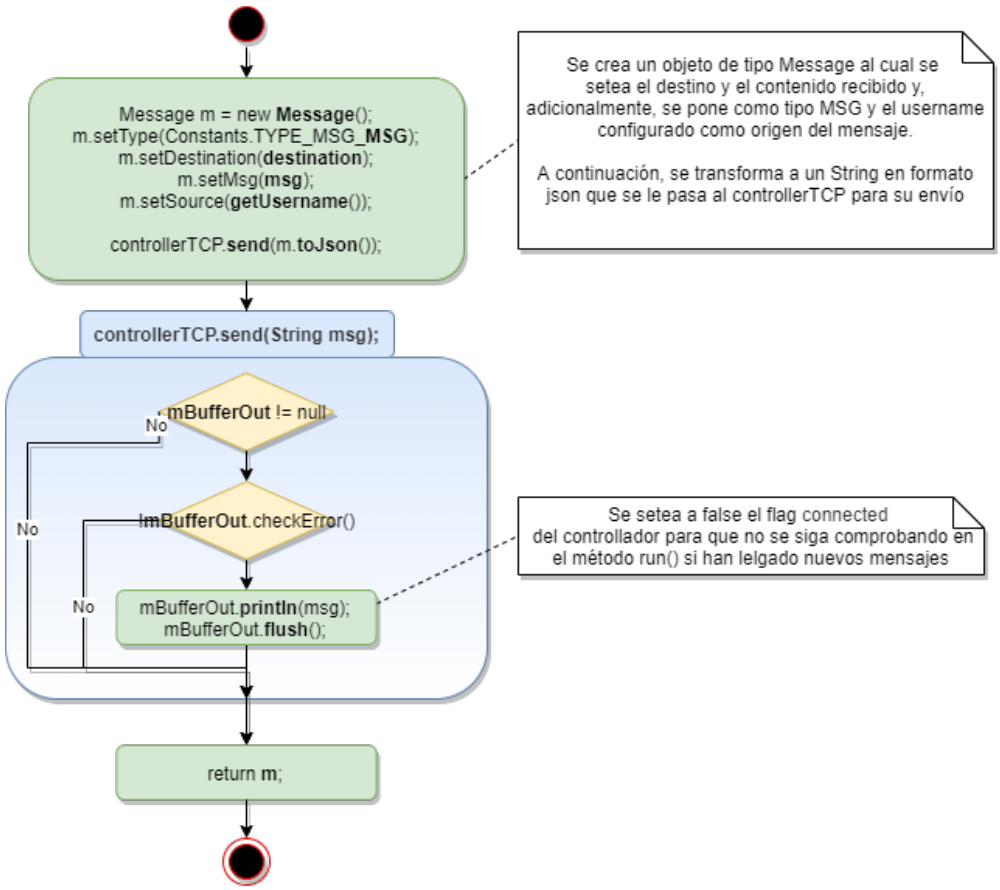


Figura 7.7: Flujo **sendMessage(String destination, String msg)**.

A continuación, en la figura 7.8 se explica el flujo seguido en el dispositivo emisor a la hora de realizar un test.

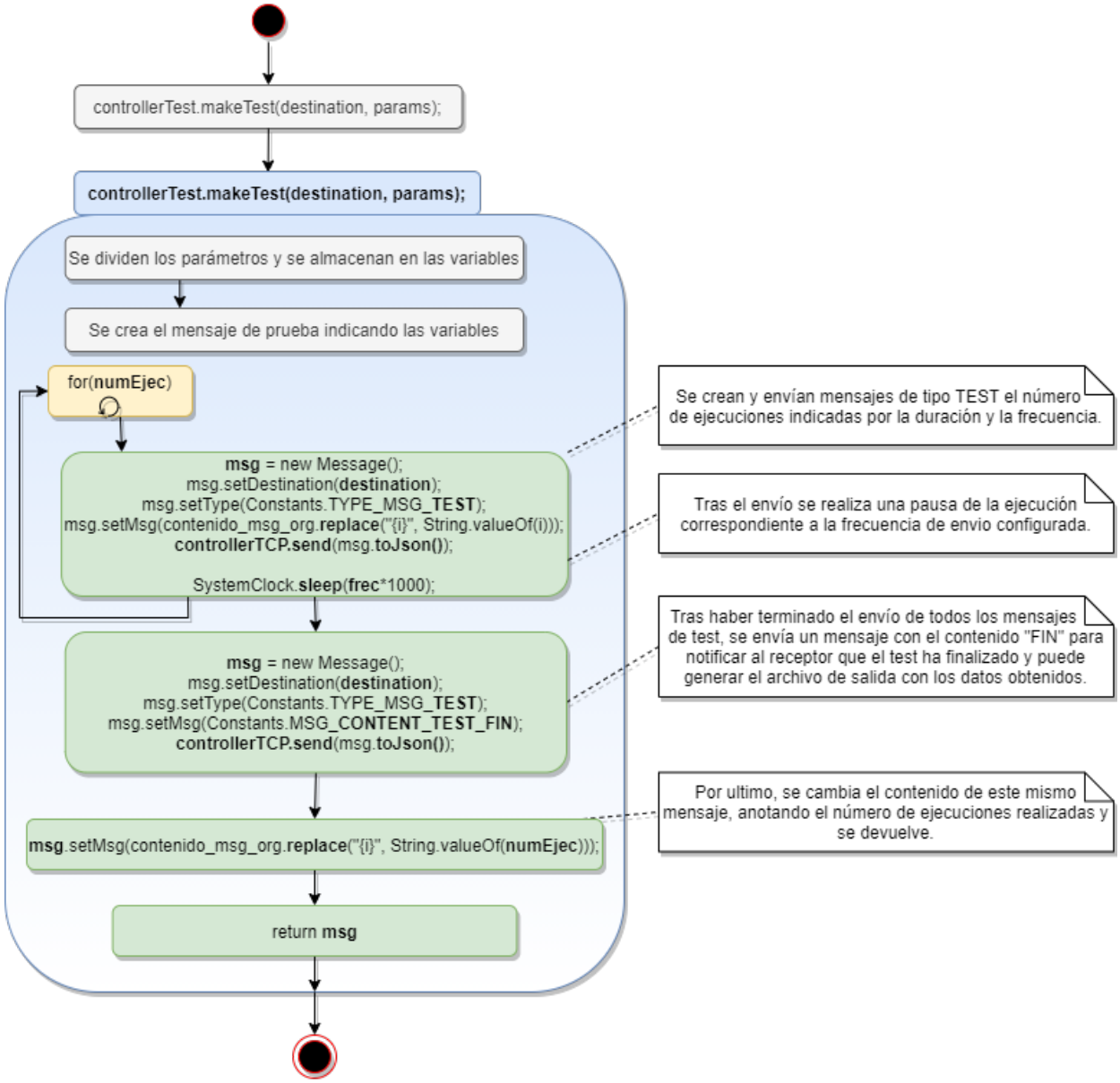


Figura 7.8: Flujo **makeTest(String destination, String params)**.

Cuando se ejecute un test, un usuario emisor lanzará los mensajes tal y como se ha indicado en la figura 7.8 y otro usuario recibirá esos mensajes como se indica en la figura 7.9 y evaluará los tiempos de cada mensaje.

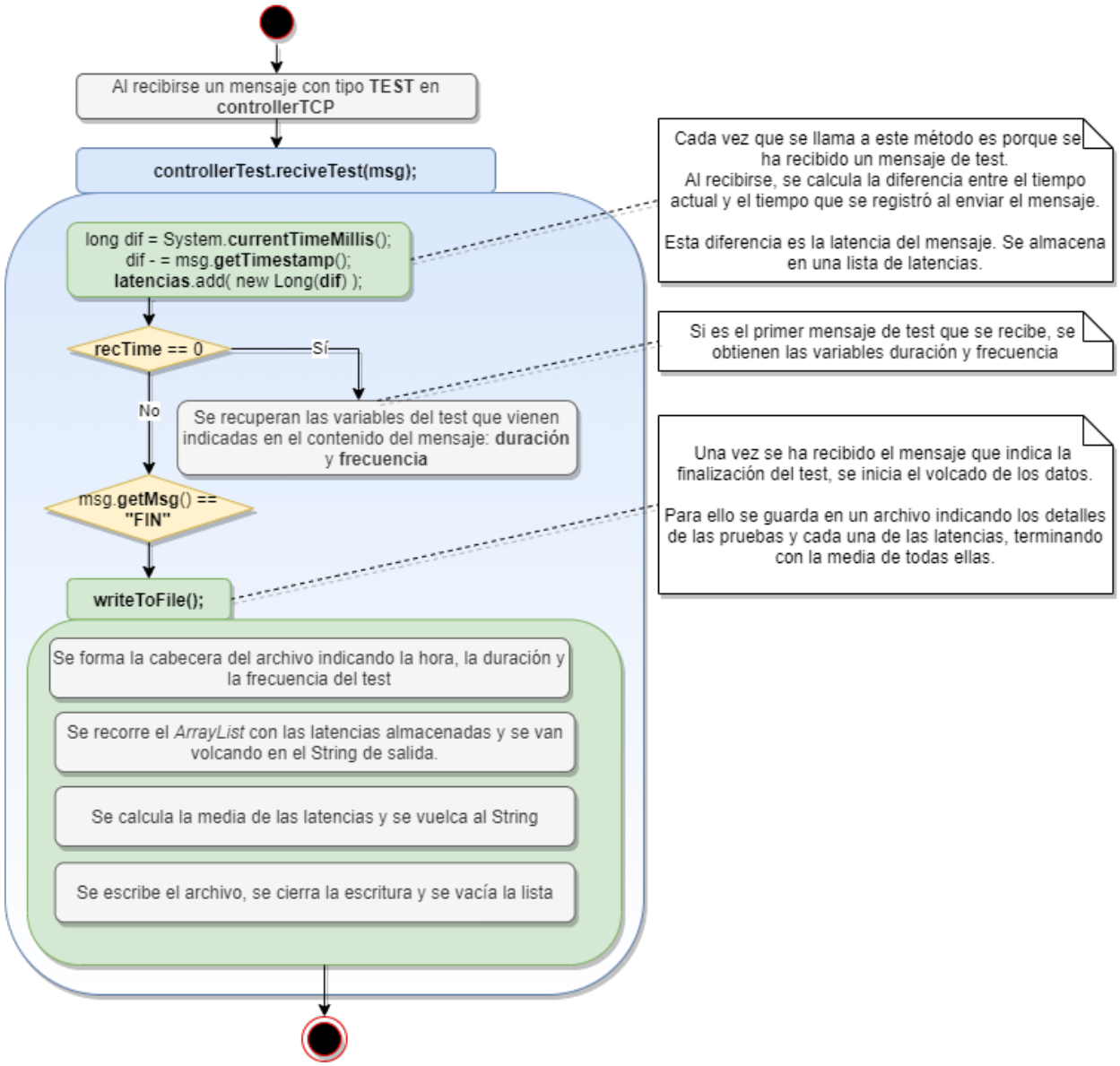


Figura 7.9: Flujo **reciveTest(Message msg)**.

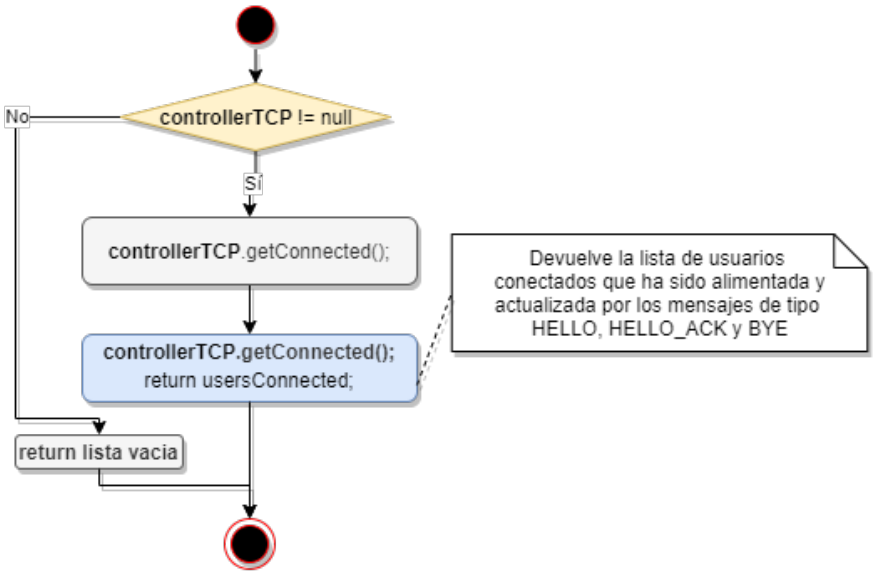


Figura 7.10: Flujo **getConnected()**.

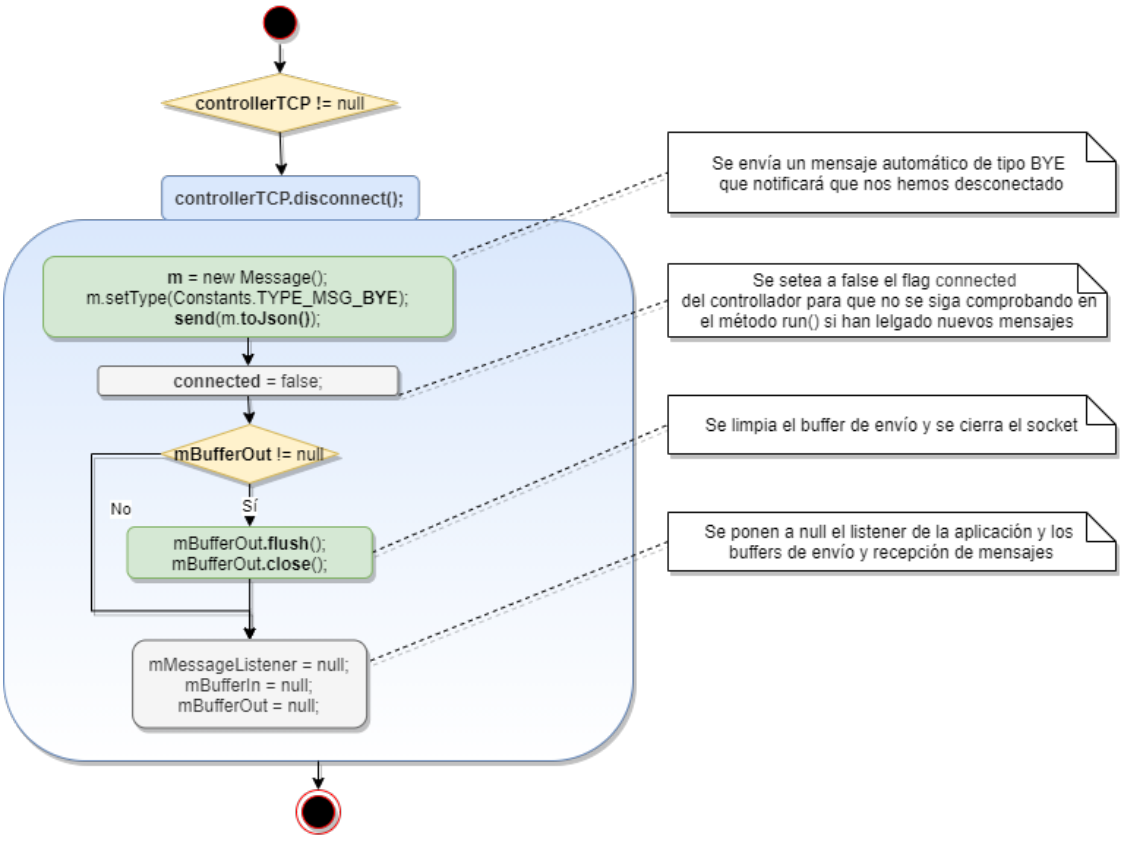


Figura 7.11: Flujo **disconnect()**.

Capítulo 8. Desarrollo de una prueba de concepto

8.1. Propósito del desarrollo

Este desarrollo es una **Prueba de Concepto**. El propósito de este desarrollo es poder mostrar cómo sería la implementación de la solución, y ejemplificar de una forma práctica el funcionamiento de los desarrollos previos. A efectos prácticos es una **aplicación Android completa**.

8.2. Arquitectura

La figura 8.1 muestra, de manera esquemática, la arquitectura completa de la aplicación propuesta. Se detallan a continuación sus principales componentes:

- Paquete **controller:** En él se ubica el *ControllerChat*, que es la clase encargada de realizar la gestión de los chats.
- Paquete **data:** Contiene el modelo de dato *Chat*, un modelo de datos donde se almacenan los distintos mensajes enviados y recibidos para cada uno de los usuarios.
- Clases **Adapter:** *ChatListAdapter, ContactsListAdapter, MsgInChatAdapter*. Son las clases encargadas de la transformación de las listas de tipos de datos como *Chat* o

Message a elementos visuales compatibles con la parte gráfica de la aplicación.

- Clases **Fragment:** Son las clases encargadas de controlar las operaciones que se realizan sobre la interfaz gráfica. Recogen las acciones a realizar cuando se acciona algún *listener* o las funciones que se lanzan al crearse el elemento gráfico que tiene asociado.

- Clase **MainActivity:** Actividad principal en la cual se inicia la aplicación y dentro de la cual se realiza la primera llamada a la *SDK*: la llamada a *init(Context context)*.

- Paquete de la **GUI:** Dentro del cual se encuentran otros subpaquetes como: *drawable* que agrupa los elementos que se pintan, *layout* que agrupa las definiciones gráficas de las pantallas, *menu* donde se define la apariencia del menú desplegable, y *values* donde se definen los valores de textos y otros elementos.

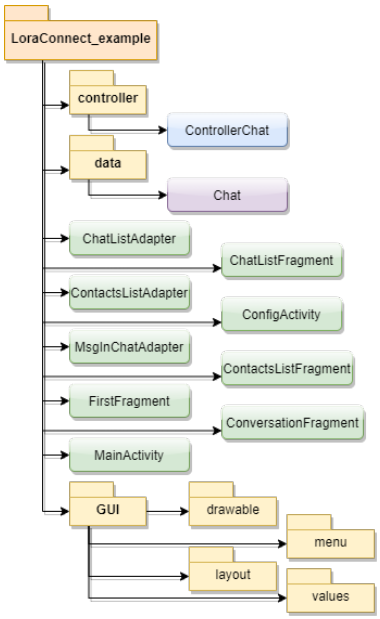


Figura 8.1: POC: Paquetería del proyecto.

8.3. Diagramas

8.3.1. Diagrama de clases

Mediante el diagrama de la figura 8.2 se representan las clases y sus respectivos atributos y métodos que conforman el código de la *POC*.

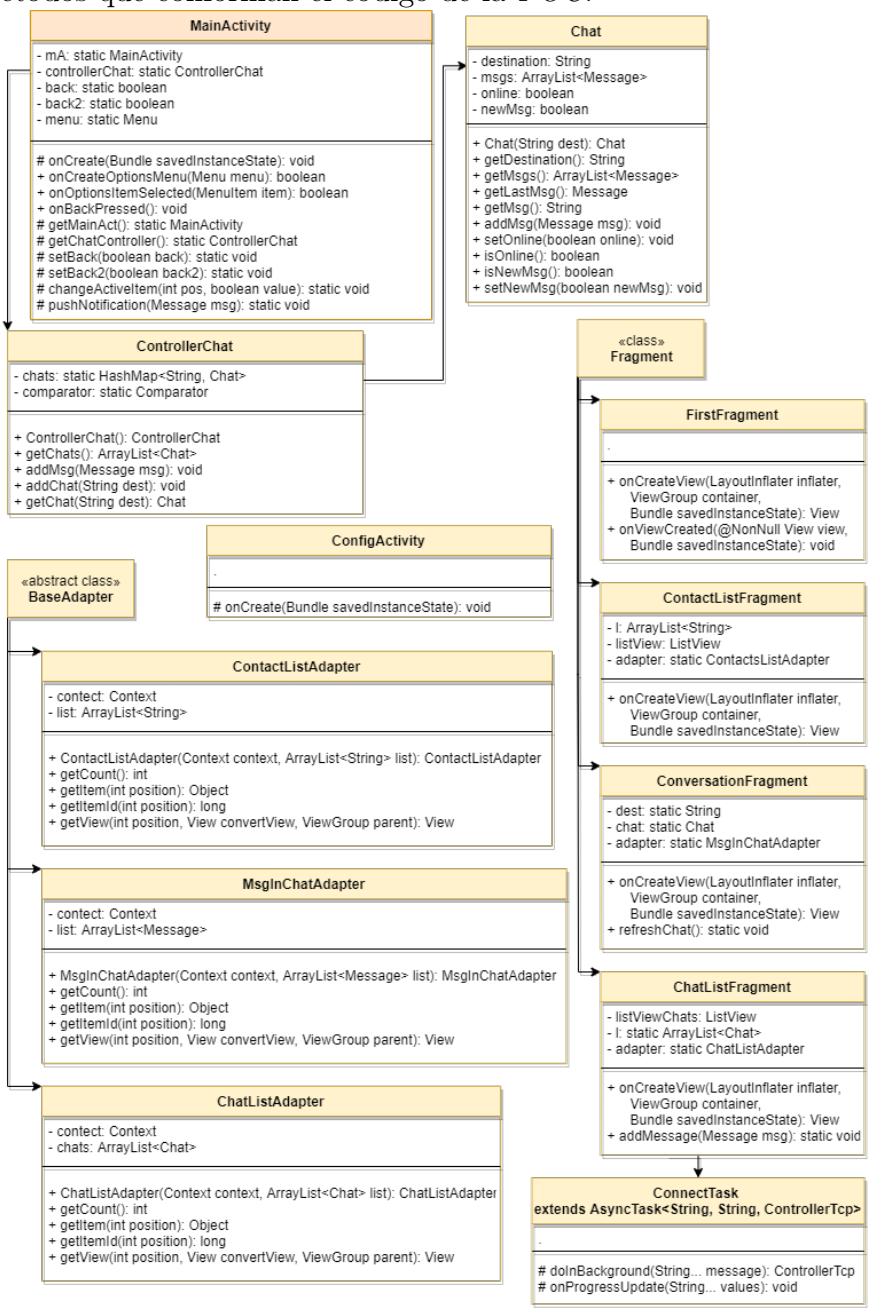


Figura 8.2: Clases del proyecto.

8.3.2. Diagramas de Secuencia

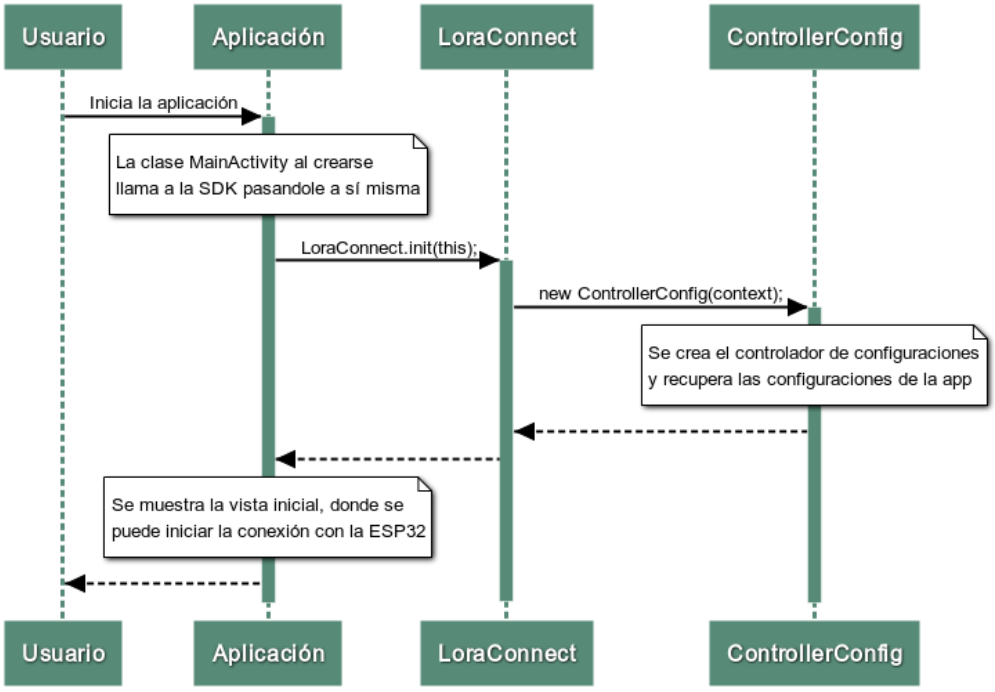


Figura 8.3: Secuencia: **Inicialización** de la SDK.

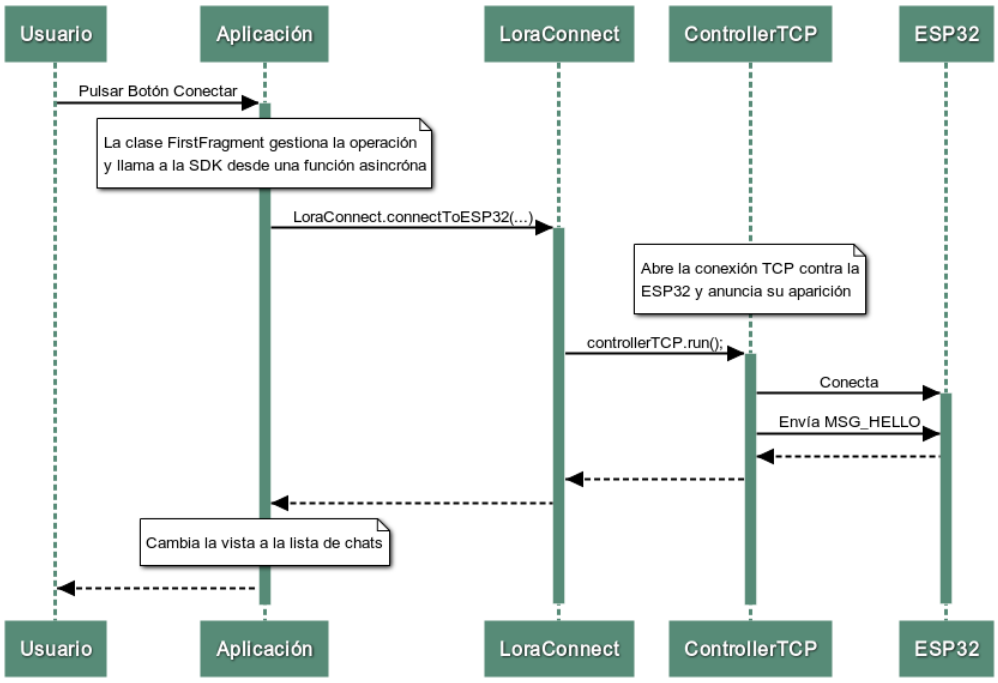


Figura 8.4: Secuencia: **Conexión** a la ESP32.

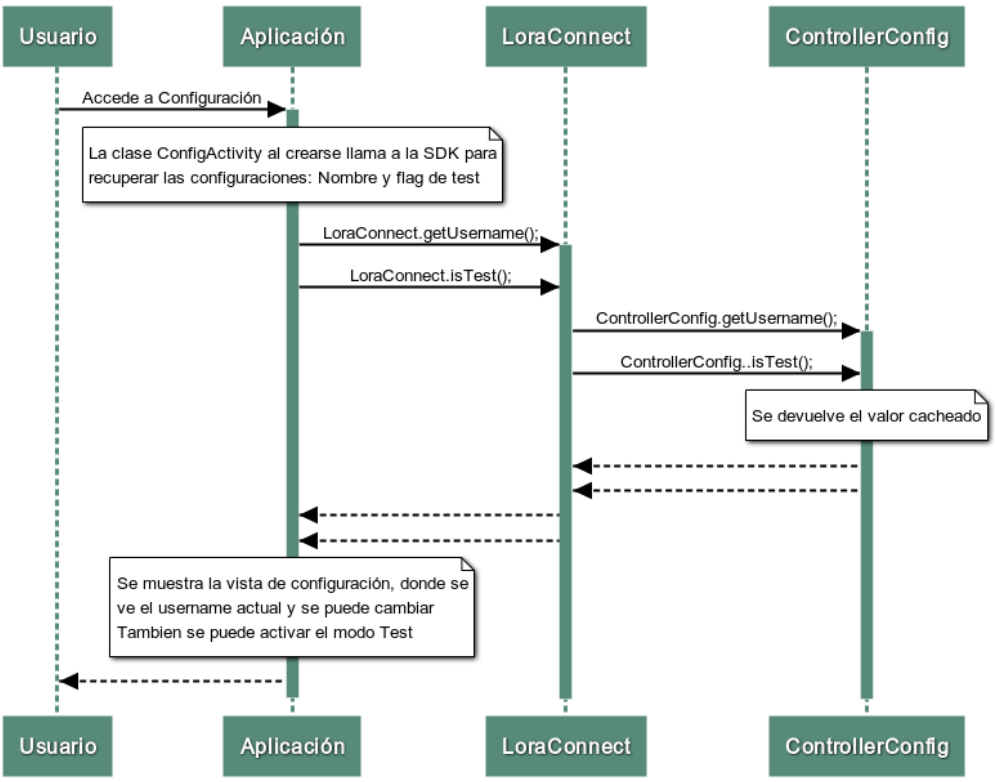


Figura 8.5: Secuencia: Gestión de **configuraciones**.

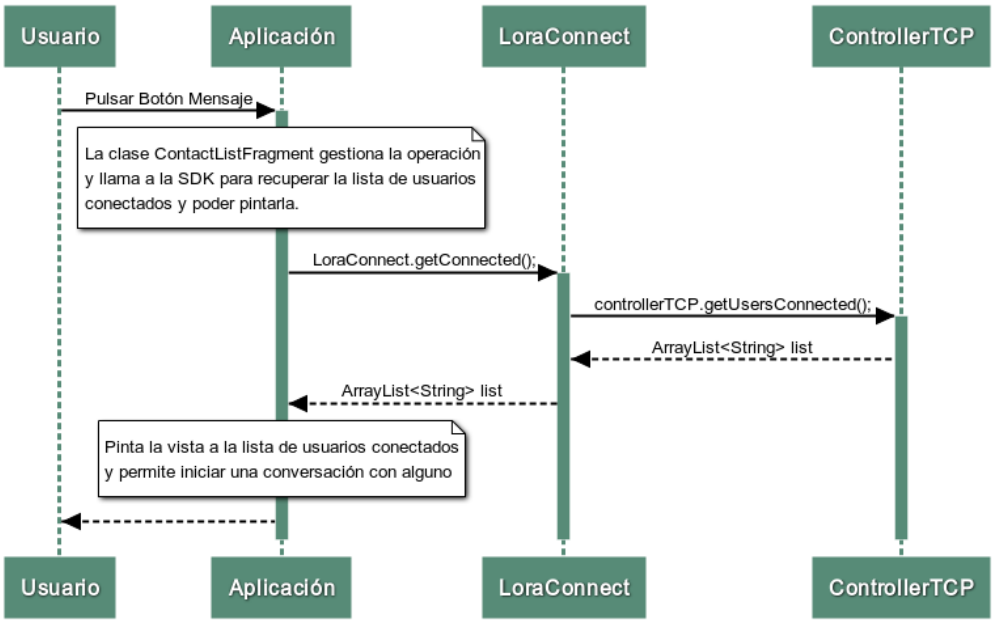


Figura 8.6: Secuencia: Listar **usuarios conectados**.

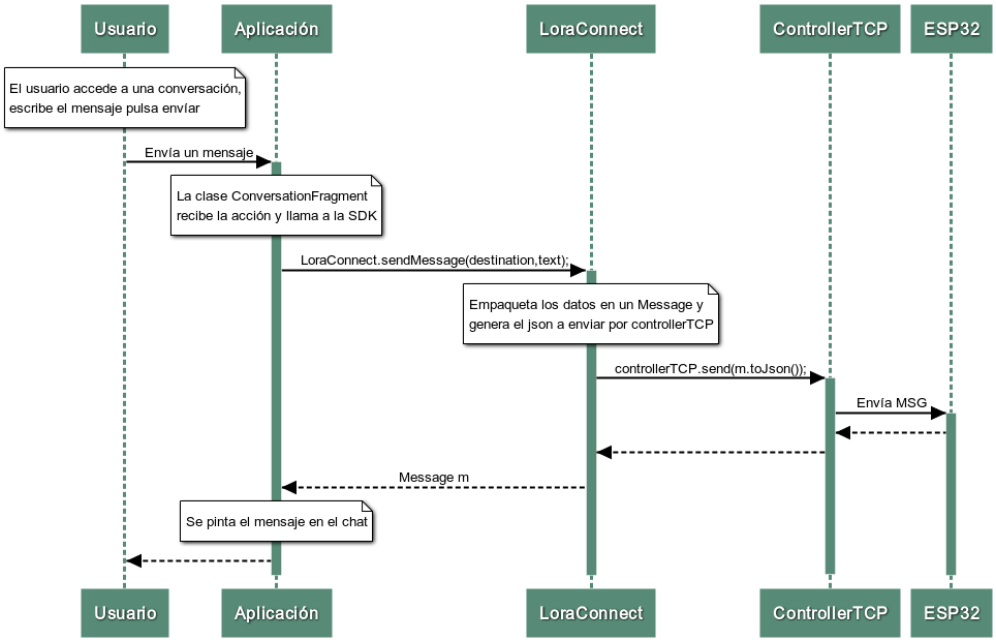


Figura 8.7: Secuencia: **Envío** de mensaje.

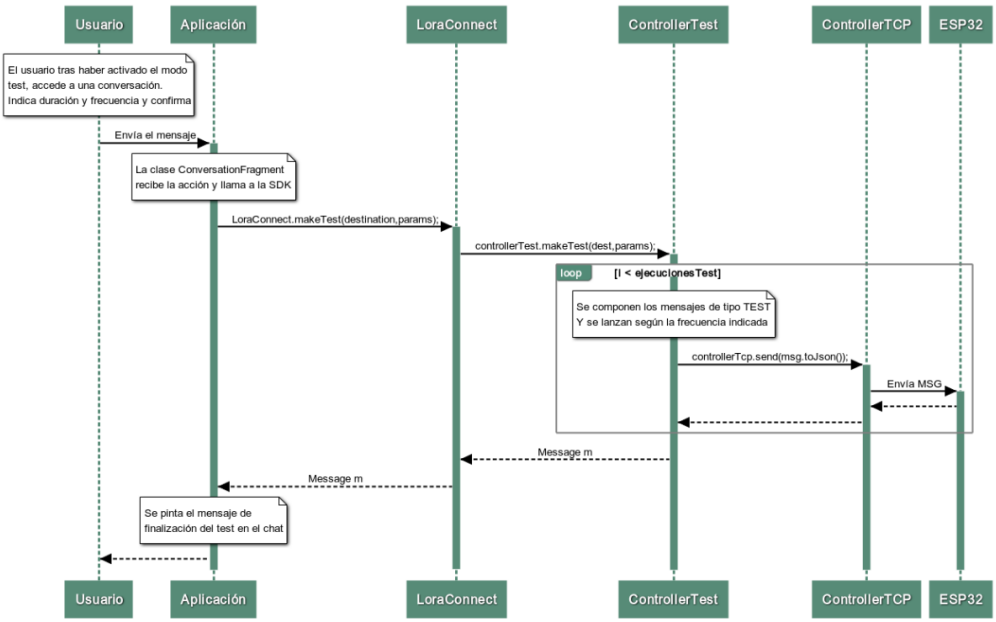


Figura 8.8: Secuencia: **Recepción** de mensaje.

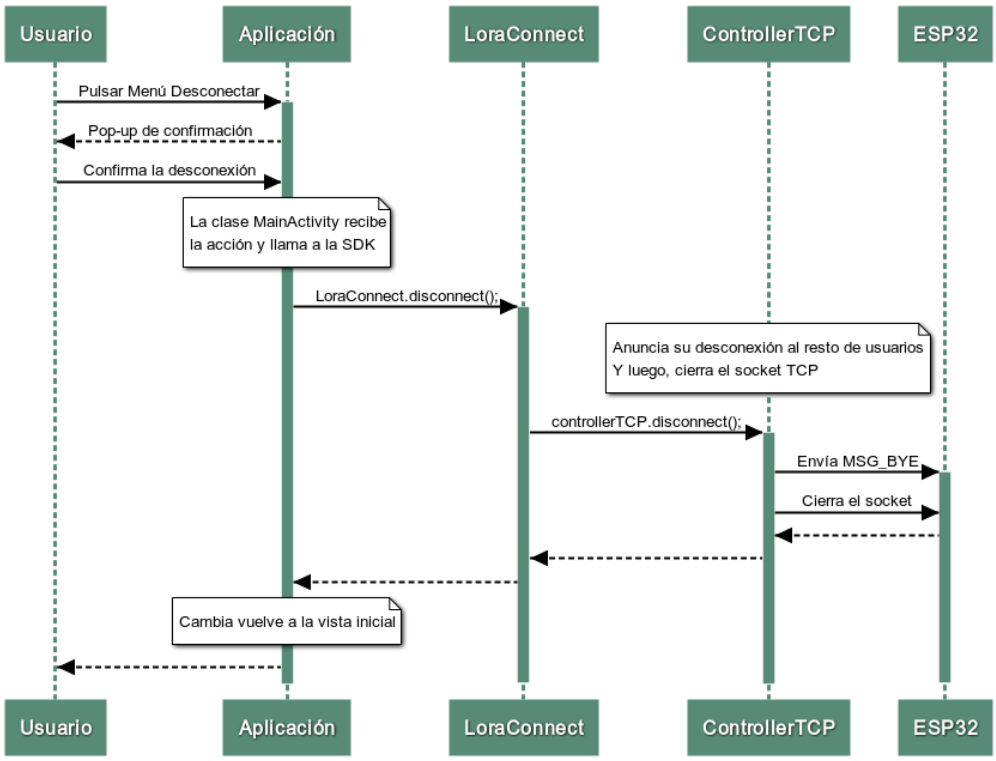


Figura 8.9: Secuencia: **Envío** de Test.

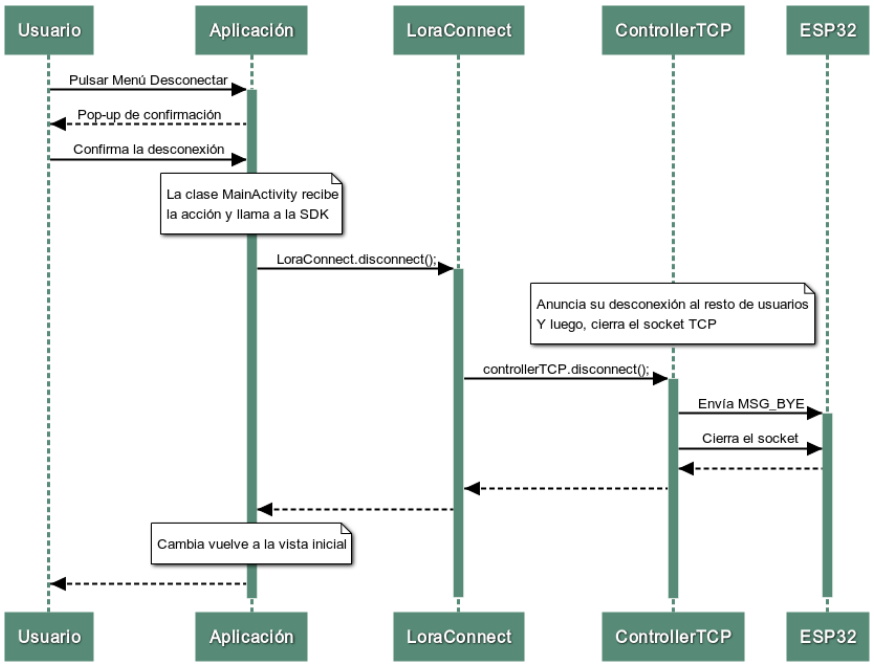


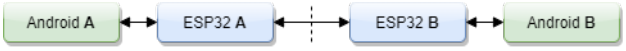
Figura 8.10: Secuencia: **Desconexión** de la ESP32.

Capítulo 9. Otras aplicaciones y usos

La realización de la aplicación *POC* tenía como objetivo realizar una implementación práctica del uso de la *SDK* y demostrar su funcionamiento. Pero esta *POC* solamente es un ejemplo de las múltiples aplicaciones que se le pueden dar. A continuación se comentan otros posibles ejemplos de aplicaciones que pueden integrar la *SDK* de cara a beneficiarse de poder utilizar la red *LoRa* desde un dispositivo *Android*.

9.1. Ejemplo 1: Aplicación de tracking deportivo

Este ejemplo sigue la misma estructura de despliegue que la *POC*:



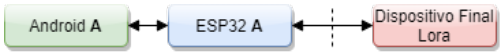
No necesariamente tiene que tratarse de un intercambio de mensajes de chat. Imaginemos una aplicación para tener localizados a los participantes de un evento deportivo en la montaña -*como puede ser una carrera de trekking o de MBT*- mediante sus posicionamientos *GPS*.

Al no tener conectividad por red dificulta su efectividad. Esta *SDK* y una *ESP32* solventa el problema aportando la capacidad de transmitir estas coordenadas a tiempo real por medio de la *LPWAN LoRa*.

No necesariamente tiene que ser una estructura de despliegue donde haya otro dispositivo *Android*, u otro usuario al otro lado de la conexión, como en los siguientes casos.

9.2. Ejemplo 2: Aplicación de notificación de intrusos

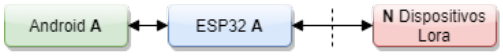
La estructura del despliegue sería:



Imaginemos en que tenemos un sensor o alarma que alertaría al detectar intrusos. En un entorno rural o una zona sin cobertura, enviando el mensaje de alerta por *LoRa* a la *ESP32* e implementando una aplicación con la *SDK*, recibiríamos la alerta al instante en nuestro dispositivo móvil.

9.3. Ejemplo 3 Aplicación de control directo de actuadores en un finca

Por ultimo, otra posibilidad es la estructura de despliegue:



Pensemos ahora en un usuario que tiene distintos *servos*, motores o actuadores con conectividad *LoRa* desplegados en una finca de gran extensión. Con una simple aplicación que implemente la *SDK* y una *ESP32* podría comunicarse directamente contra estos dispositivos mediante un móvil *Android* y enviar los comandos que quiera.

Capítulo 10. Trabajo futuro

En esta sección se comentan ideas que podrían aportar más valor aún al proyecto, pero no ha sido posible llevarlas a cabo por su complejidad o debido a un aumento del alcance y del tiempo de desarrollo que superaría las expectativas de un *TFM*.

- Desarrollar un **control de entrega de mensajes** entre dispositivos finales, de manera que ningún mensaje se perdiese y hubiese un acuse de recibo.
  Para ello sería necesario cambiar la lógica de la gestión de mensajes recibidos, el modelo de datos, añadir un nuevo tipo de mensaje *(ACK)* en la *SDK* y finalmente, gestionar la representación de ello en la aplicación *POC* de ejemplo mediante un doble *tick* bajo el mensaje.

- Desarrollar una solución de **Cifrado punto a punto** del contenido.
  Esto implica la implementación en la *ESP32* de un sistema de cifrado asimétrico. De forma que se intercambiaría una clave o un certificado público junto con los mensajes de tipo *HELLO* y *HELLO ACK* entre los dispositivos implicados y desarrollar una gestión de claves para su recuperación a la hora de cifrar y descifrar.

  Técnicamente esto tiene cierta complejidad a la hora de realizar el intercambio de certificados entre los N usuarios que se encuentran conectados sin saturar la red.

  Cabe destacar que esto añadiría también mayor consumo energético y un aumento de tiempos a la hora del envío y recepción de los mensajes.

Capítulo 11. Ejecución

A lo largo de esta sección se explica y se muestra mediante imágenes, tanto de la *ESP32* como del dispositivo *Android*, cómo es una ejecución de la solución.
Para ello se intercalan imágenes de los distintos eventos que se producen en la *ESP32* y capturas del dispositivo Android.

El primer paso es iniciar la *ESP32* conectándola a una fuente de alimentación. Al iniciar se presenta como en la figura 11.1. Se muestra un código qr cuya función es mostrar de forma codificada el nombre y la contraseña del punto de acceso Wifi que genera la *ESP32* y a su derecha el nombre representado en formato de texto. En este caso el nombre es *AP*.

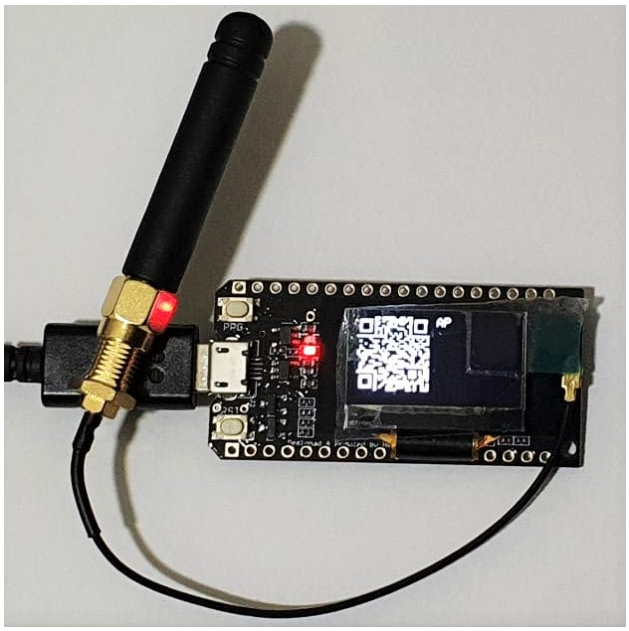


Figura 11.1: ESP32: Pantalla de inicio.

De esta forma se puede realizar un escaneo del código qr con la cámara del dispositivo Android y se establecerá la conexión wifi con dicho punto de acceso.

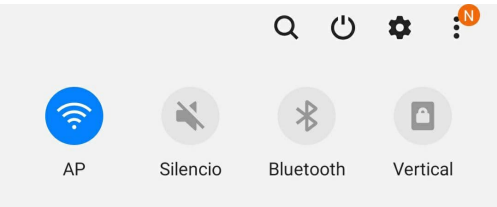


Figura 11.2: Android: Dispositivo conectado a la *ESP32*.

Una vez conectados al punto de acceso wifi de la *ESP32*, como indica la figura 11.2, iniciamos la aplicación. De esta forma veremos la pantalla de inicio de la figura 11.3.

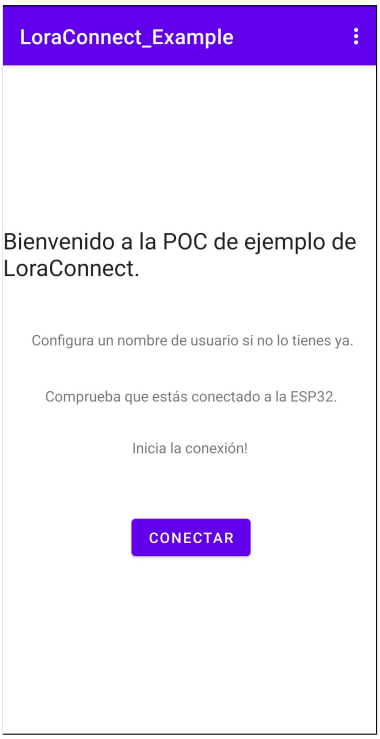


Figura 11.3: Android: Pantalla de inicio.

Desde esta pantalla podemos acceder al menú de configuración (figura 11.4) para cambiar el nombre de usuario o activar el modo test.

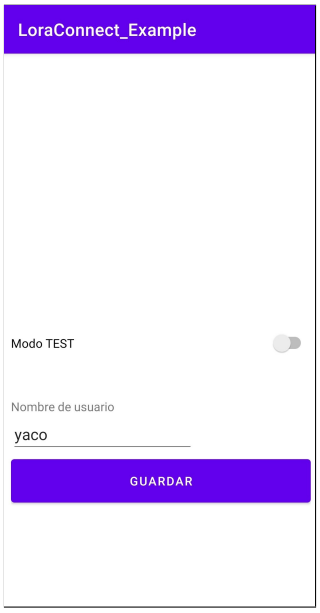


Figura 11.4: Android: Pantalla de configuración.

Tras guardar los cambios, volvemos a la pantalla de inicio (figura 11.3) y pulsamos el botón de **Conectar**. De esta forma iniciaremos la conexión TCP contra la *ESP32* y al hacerlo se enviará un mensaje automático de tipo *HELLO*, con el objetivo de notificar a los usuarios que nos hemos conectado a la red *LoRa*. Esto se puede observar en la figura 11.5.

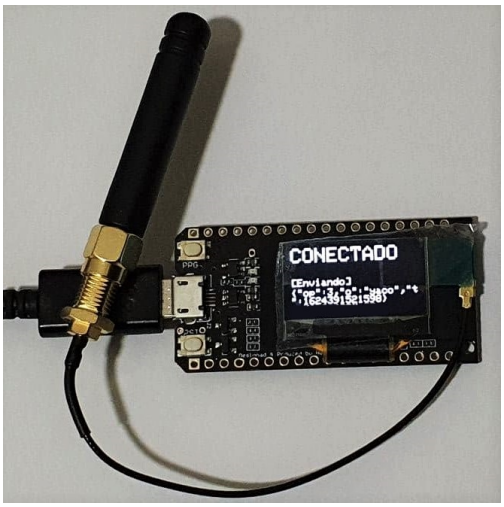


Figura 11.5: ESP32: Envío de mensaje de tipo *HELLO*.

Después de este envío obtenemos una respuesta con tipo de mensaje *HELLO_ACK*, como se observa en la figura 11.6 que nos indica que el usuario **paco** está conectado y le ha llegado nuestro mensaje de tipo *HELLO*.

Figura 11.6: Recepción de mensaje de tipo HELLO_ACK.

Al conectarnos no tendremos conversaciones aún, por lo que la lista aparecerá en blanco, como en la figura 11.7.

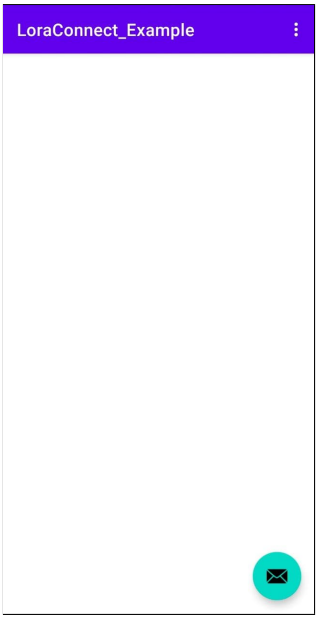


Figura 11.7: Android: Pantalla con la lista -*vacía*- de conversaciones.

Si pulsamos en el botón verde del sobre, consultaremos la lista de usuarios conectados actualmente, apareciendo la lista mostrada en la figura 11.8.

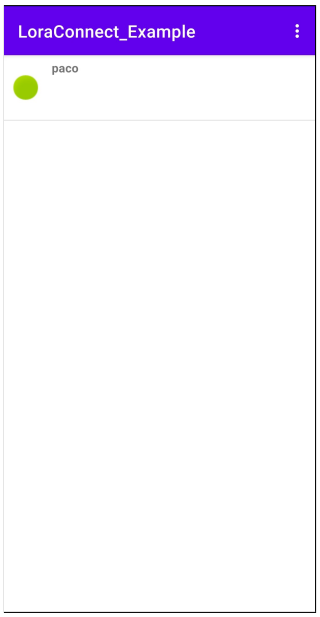


Figura 11.8: Android: Pantalla con la lista de usuarios conectados.

Pulsando sobre el nombre de **paco** podemos iniciar una conversación, y en ella podremos enviar y recibir mensajes. El envío y la recepción de mensajes al pasar por la *ESP32* tienen el aspecto que se ve en las figuras 11.9 y 11.10 respectivamente.

Figura 11.9: ESP32: Envío de mensaje de tipo Mensaje.

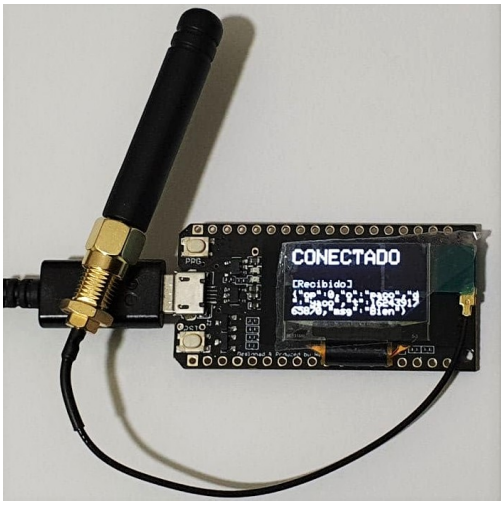


Figura 11.10: ESP32: Recepción de mensaje de tipo Mensaje.

Tras haber realizado el intercambio de unos pocos mensajes la conversación presenta el aspecto de la figura 11.11.

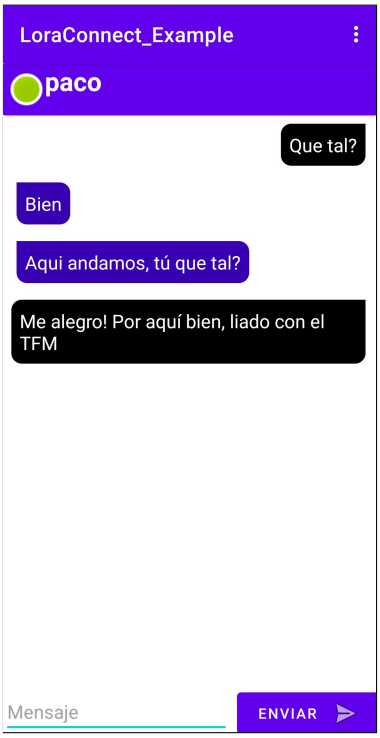


Figura 11.11: Android: Pantalla de conversación con paco.

Cabe destacar que según van llegando mensajes, se genera una notificación *push* que aparece en la lista de notificaciones como en la figura 11.12.

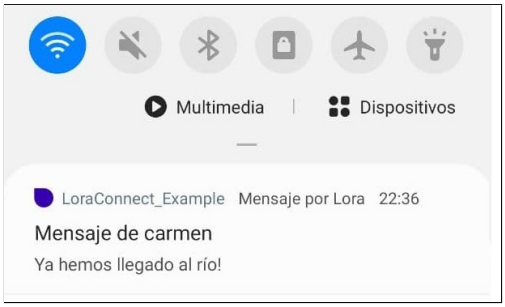


Figura 11.12: Android: Notificación push.

Una vez se han entablado conversaciones con diferentes usuarios el aspecto que tendría la pantalla con la lista de conversaciones habría dejado de estar vacía y luciría como en la figura 11.13 y no vacía como antes.

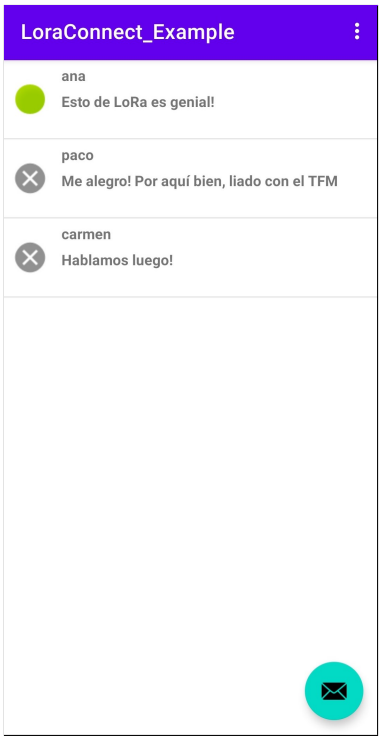


Figura 11.13: Android: Pantalla con la lista de conversaciones.

Como funcionalidad aparte, se va a mostrar qué aspecto tendría una ejecución de un test para el calculo de las latencias.

En la figura 11.14 se puede observar la pantalla del usuario emisor que lanza el test, y en su campo de mensaje se indica que el formato de mensaje para iniciar un test es 'tiempo;frecuencia'.

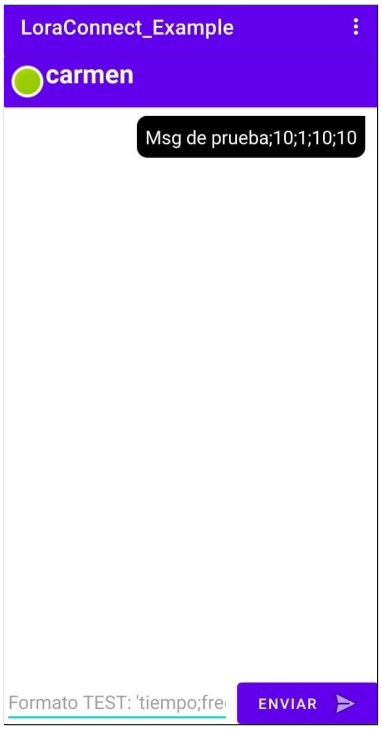


Figura 11.14: Android: Pantalla del emisor del test.

Tras iniciar el test, los mensajes irán llegando al destinatario, se irán pintando en la conversación (como en la figura 11.15), e internamente se irán calculando las latencias de los mensajes. Este calculo se realiza mediante la diferencia del timestamp actual y el timestamp que lleva el mensaje e indica su creación.

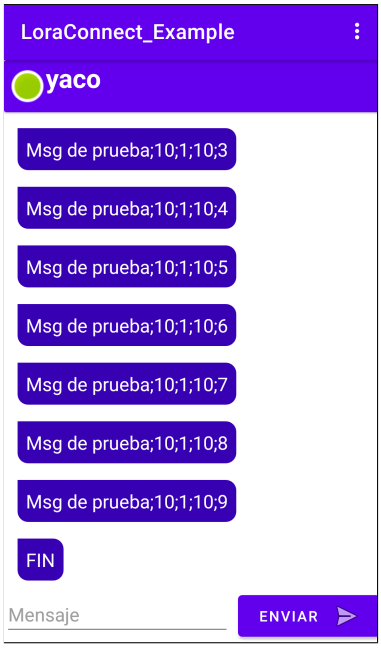


Figura 11.15: Android: Pantalla del receptor del test.

Y por último, se muestra como se realiza la desconexión.

Desde la vista principal, la lista de conversaciones, se despliega el menú como en la figura 11.16, se pulsa **Desconectar** y aparecerá el mensaje de confirmación que aparece en la figura 11.17.

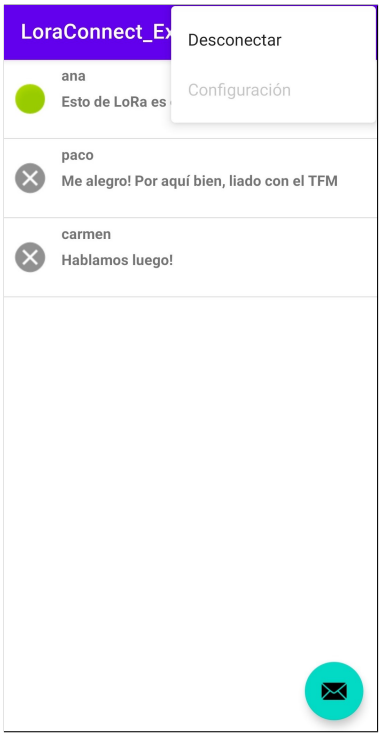


Figura 11.16: Android: Desconexión.

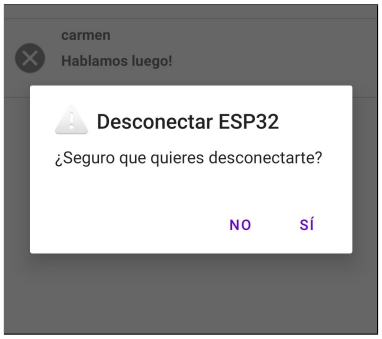


Figura 11.17: Android: Confirmación de desconexión.

En caso de confirmar, se enviaría un mensaje automático, al igual que se hizo al conectarnos, pero con tipo de mensaje *BYE*. Con esto se anuncia a todos los usuarios conectados que cortamos la conexión con la red *LoRa*.

Capítulo 12. Resultados

12.1. Método de obtención

Los resultados analizados se obtienen tras extraer los datos de los archivos de salida generados en al ejecución de los test. A continuación un ejemplo:

```
Test Latencias LoraConnect - Sun Jun 20 19:30:42 GMT+02:00 2021 -
Duracion:10 - Frecuencia: 1 - Ejecuciones: 10

139
160
79
101
122
144
167
86
130
151

MEDIA: 127
```

Interpretando y recolectando los datos de las distintas ejecuciones se han generado unas tablas con el fin de resumir la información obtenida, y posteriormente interpretarla.

12.2. Resultados obtenidos

Tras realizar ejecuciones en modo test se han obtenido los siguientes resultados:

Distancia 1 metro			
Duración	Frecuencia	Latencia media	% perdida
10 seg	1 seg	361	10
10 seg	2 seg	320	0
60 seg	1 seg	350	1,67
Distancia 15 metros			
Duración	Frecuencia	Latencia media	% perdida
10 seg	1 seg	354	0
10 seg	2 seg	345	10
60 seg	1 seg	347	5

Cuadro 12.1: Tabla resumen Ejecución en **interior** en zona urbana.

Distancia 1 metro			
Duración	Frecuencia	Latencia media	% perdida
10 seg	1 seg	127	0
10 seg	2 seg	120	0
60 seg	1 seg	132	1,67
Distancia 50 metros			
Duración	Frecuencia	Latencia media	% perdida
10 seg	1 seg	137	10
10 seg	2 seg	130	0
60 seg	1 seg	137	33,3
Distancia 200 metros			
Duración	Frecuencia	Latencia media	% perdida
10 seg	1 seg	132	20
10 seg	2 seg	140	30
60 seg	1 seg	137	20

Cuadro 12.2: Tabla resumen Ejecución en **exterior** en zona retirada.

12.3. Interpretación de los resultados

Como se observa en la tabla 12.1, en el entorno urbano se ha identificado que las latencias son el doble que en una zona rural o semi-urbana. Esto entra dentro de lo esperado, ya que existe un mayor número de interferencias.

Se han detectado ciertas perdidas de paquetes. Esto sería mejorable mediante la implementación de un sistema de de reintentos y de confirmación de recepción tal y como se indica en el apartado de *Trabajo futuro* (10).

Como se muestra en la tabla 12.2, en el entorno semi-urbano, se cumplen los objetivos buscados aunque no se han obtenido resultados tan optimistas como los esperados. Según las especificaciones de *LoRa*, ya que no ha sido posible alcanzar una distancia de kilómetros entre los dispositivos.

Tras analizar cuál puede haber sido el motivo, la conclusión ha sido que puede estar achacado al hecho de que la antena que trae esta *ESP32 (TTGO Lora32 Oled v2)* es muy pequeña y de poco alcance.
Sustituyéndola por una antena más grande y de mayor alcance se podrían llegar a alcanzar rangos como los indicados, del orden de kilómetros.

Capítulo 13. Conclusiones

Una vez finalizado este proyecto, tras haber planificado, desarrollado , implementado, probado y documentado todo el trabajo, se concluye lo siguiente:

- Ha sido posible **realizar con éxito** el desarrollo de este trabajo, pudiendo centrarme en una tecnología *LPWAN* como es *LoRa*. Esto ha sido posible gracias a los conocimientos en el campo del *Internet de las Cosas* que me ha aportado la realización de este máster.
- Se han **cumplido los objetivos** planteados al comienzo del proyecto. Se ha conseguido construir una solución válida y sólida, que brinda a toda la comunidad de desarrolladores *IOT* y/o *Android* la posibilidad de integrar conectividad *LoRa* a sus desarrollos.
- Se ha conseguido que la aplicación *POC* sea un **ejemplo muy completo** de todas las funcionalidades que aporta la *SDK Android* desarrollada.
- La realización de pruebas mediante un **despliegue real** de la solución ha aportado una buena experiencia.
- Aún siendo un proyecto ambicioso, existe posibilidad de mejora mediante la realización de las tareas indicadas en el apartado de **trabajo futuro** (10).

Chapter 1. Why equip an Android device with LoRa?

1.1. LPWAN

What is it?

LPWAN (Low Power Wide Area Network) are medium and long-range wireless networks, which can cover several kilometers of coverage [1].

Due to the current boom in the development of solutions in the field of the Internet of Things, its use has been extended and intensified.

What are the advantages over GSM or 4G?

Traditional long-range wireless networks are *GSM (Global System for Mobile communications)* and *4G (Fourth generation of mobile phone technologies)* and their use is widespread, mainly in telephony devices.

However, when making an *IOT* deployment, using a *LPWAN* provides the following advantages:

- The most important is the low energy consumption they have. This is a very positive

point because in most *IOT* deployments the power supplies are very limited.

- They do not require large hardware resources, as they are intended for devices with limited memory capacity and computing power and low energy consumption.
- They allow to have wireless connectivity in areas where there is no *GSM/4G* coverage.
- They are networks that allow the implementation of our own infrastructure, so that we do not necessarily depend on a network provider. Even so, more and more telephone operators offer infrastructure for this type of coverage, such as the *LTE-M* and *NB-IoT* networks.

What are the disadvantages?

- They provide a very limited bandwidth, on the order of *KB*. Decreasing the bandwidth is how they achieve coverage over such great distances.
- The use of some of these networks is under license and requires a financial outlay to be able to use them, as is the case of the aforementioned *LTE-M* and *NB-IoT*.

LoRa

LoRa is a wide spectrum modulation technology. This allows it to tolerate noise and multiple signal paths, while keeping power consumption very low. To achieve this, bandwidth is sacrificed, which is very low compared to other wireless technologies [2].

To carry out this project, it was decided to use *LoRa* as a wireless connection technology.

This decision was made on the basis that it is a license-free *LPWAN*, its signal provides a long range, and compared to other wireless technologies it has a very low battery consumption.

As mentioned before, these characteristics are obtained by sacrificing bandwidth, but this is not a problem since it will not require a wide bandwidth.

In terms of economic cost, as it is a free license it does not imply an outlay, and when it comes to finding compatible hardware, devices with *LoRa* connectivity have a lower cost compared to others with *LPWAN* connectivity such as *NB-IOT* or *Sigfox*.

Below you can see a graph that compares the *Sigfox, Wifi, NB-IoT and LoRaWAN* networks [3].

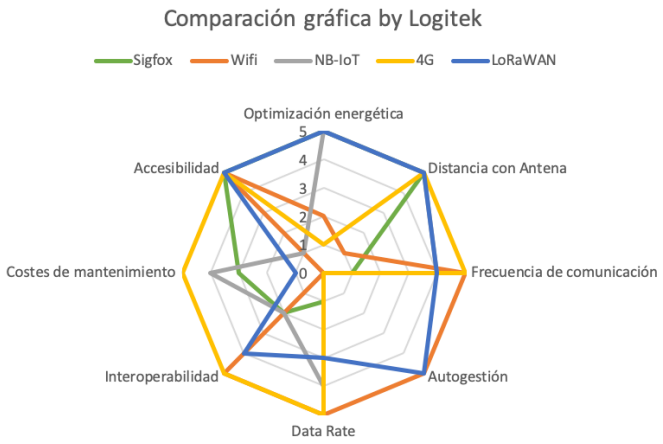


Figura 1.1: Comparison between different wireless technologies.

1.2. Motivation

The main motivation behind this project is to provide solutions when developing connectivity alternatives with *Android* devices in areas where traditional coverage is not available, especially in rural areas.

Currently there are areas in which, to get mobile coverage, it is necessary to access specific locations and sometimes difficult to reach. Other scenarios with connectivity problems include agricultural or outdoor areas where sports activities are carried out (outdoor races, excursions, competitions in mountainous areas, ...) Finally, it should be mentioned that many times it is not necessary to leave an urban nucleus to find ourselves isolated, due to the saturation of the network because of mass events such as concerts or festivals.

For these reasons, it was decided to propose this project to provide a solution to these cases by using a Low Power Wide Area Networks *(LPWAN)*. And, in addition, leaving the door open for other developers to take advantage of the work done and integrate the solution into their application.

1.3. Objectives

The main objective of this project is to create a solution that serves as a basis for other developers to incorporate it into their applications, allowing their applications to have connectivity in environments where there is no traditional coverage and provide answers to real-world problems such as disconnection in certain rural areas of our geography.

Additionally, from a technical point of view, this main objective is divided into the following specific objectives:

- Create a solution so that the **ESP32** board allows establishing a connection with the *Android* device and *LoRa* connections against other boards, as well as managing and transferring the messages exchanged from one network to another.
- Develop an *Android* library ***(SDK)*** that manages the connection with the *ESP32* and all the operations allowed in a simple, transparent and efficient way.
- Develop a final *Android* **application** as a **Proof of Concept** where a possible use of the previously mentioned *SDK* is exemplified in a real application.
- **Analyze the operation** of the solution in a real environment, and document the limitations of the established *LoRa* connections. In this way, maximum distances and latency degradation would be known.

1.4. Workplan

For the planning of the project, an agile methodology has been followed. Organizing in 4 large tasks called *Epics*. They are the development of this project: *ESP32*, *SDK*, *App POC* [1], and finally, the writing of this report.

These epics have been subdivided into narrow, small, and more development-focused tasks, which as soon as are identified and defined are recorded in the to-do list.

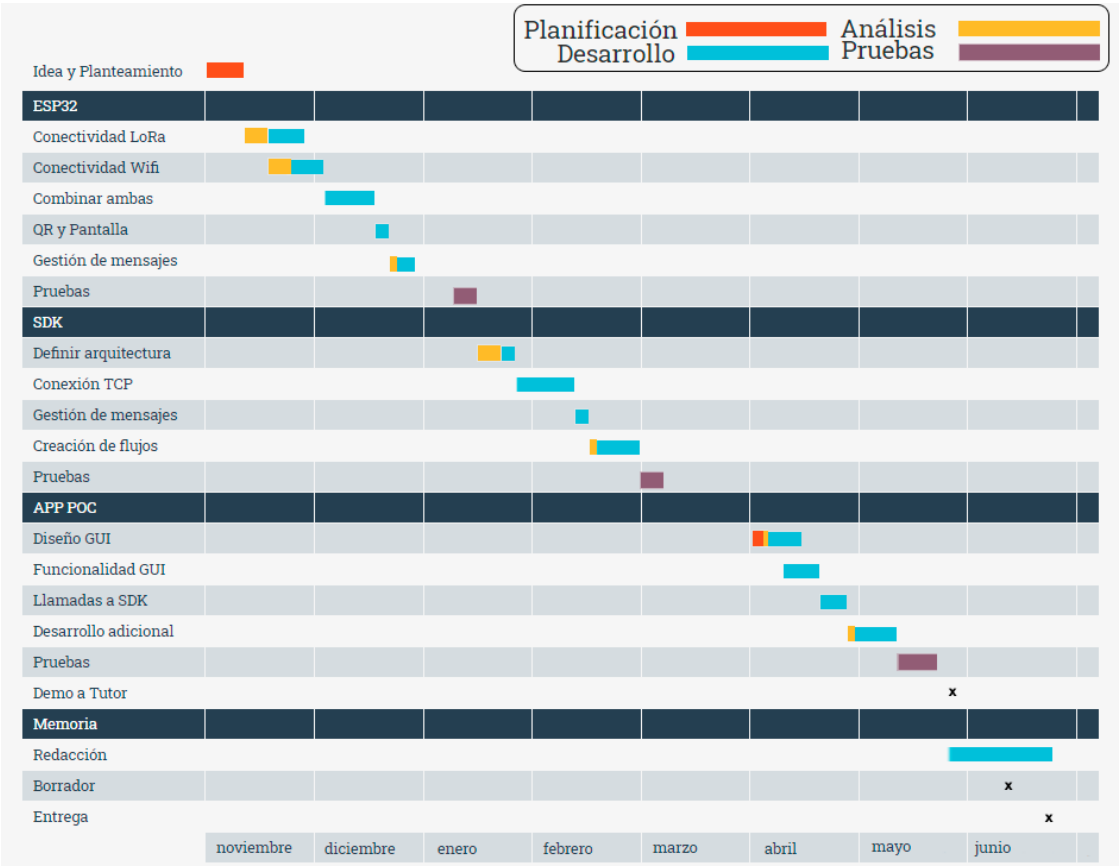


Figura 1.2: Temporary planning of the project.

It is temporarily organized by defining time periods called *Sprints* of approximately 15 days. In these periods it was marked to have completed a group of tasks that make up a

[1] Proof of concept, or Proof of concept

functionality, or failing that, if it has not been possible, identify the causes and re-analyze the tasks to be carried out if necessary.

Additionally, there are certain key dates called *goals* that mark, for example, the completion date of the development, the demonstration date of the work done to the tutor, the delivery of the draft to the tutor and, finally, the date of delivery of the definitive thesis.

To keep a record of the tasks and in what state they are, following the principles of the *Agile* methodology, we have had the support of a *Kanban* task board. Specifically, the *Github board* [2] related to the project repository has been used.

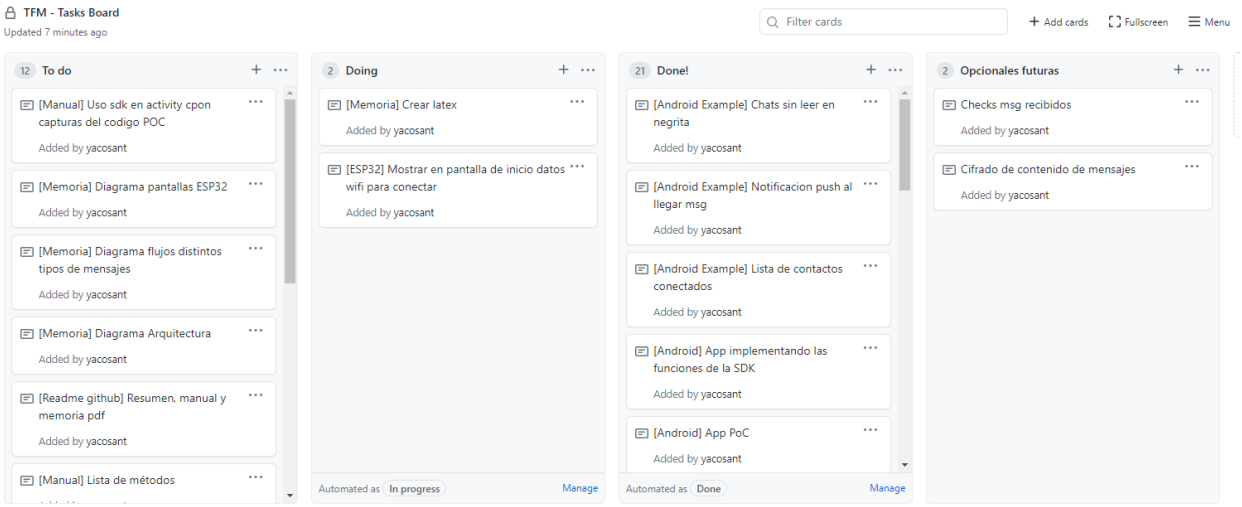


Figura 1.3: Github Kanban Board.

This board has been recording the tasks to be carried out *(To Do)*, those that were being done *(Doing)*, the finished ones *(Done)* and, finally, those that remain postponed for future work.

[2] https://github.com/yacosant/TFM/projects/1

1.4.1. Memory structure

This thesis is made up of thirteen chapters in Spanish, two chapters in English and two annexes:

- In the **first** chapter it is explained why it was decided to carry out this project, what are the objectives and how is the work plan that has been followed.
- In the **second** chapter it is developed which are the existing solutions that try to give a solution to the problem similar to ours.
- In the **third** chapter is indicated which technologies and libraries have been used and which ones have been discarded to implement the solution.
- In the **fourth** chapter the architecture that has been designed and implemented together with the main data structure of the project is presented.
- In the **fifth** chapter, the use cases that define the project's functionality are presented.
- In the **sixth**, **seventh** and **eighth** chapter, it is explained from a technical point of view how each piece of the solution is structured, and it is presented how it works using *UML*[3] diagrams.
- In the **ninth** chapter three other applications that can be carried out by implementing the solution are exposed.
- In the **tenth** chapter, the possible improvements that could be developed in the future are discussed.
- In the **eleventh** chapter the flow of a real execution is explained accompanied by photographs and screenshots.

[3]Unified Modeling Language

- In the **twelfth** chapter, the results obtained from testing the solution are collected and interpreted.
- In the **thirteenth** chapter the final conclusions of this work are presented.
- The chapters **first** and **thirteenth** have their version in English where the project is introduced and the conclusions are presented respectively.
- In **annex 1** abbreviations and terms that may need definition are listed.
- In **annex 2** is indicated where the source code of the project can be found.

Chapter 13. Conclusions

Once this project is finished, after having planned, developed, implemented, tested and documented all the work, the following is concluded:

- It has been possible to **successfully carry out** the development of this work, being able to focus on an *LPWAN* technology such as *LoRa*. This has been possible thanks to the knowledge in the field of the *Internet of Things* that the completion of this master has given to me.
- The **objectives** set at the beginning of the project **have been met**. A valid and solid solution has been built, which offers the entire community of IOT and/or Android developers the possibility of integrating LoRa connectivity to their developments.
- The *POC* application has been made to be a **very complete example** of all the functionalities provided by the developed Android SDK.
- Testing using a **real deployment** of the solution has provided a good experience.
- Even though it is an ambitious project, there is room for improvement by carrying out the tasks indicated in the **future work** section (10).

Apéndice A. Glosario

Lista de palabras y siglas, por orden de aparición, que puedan necesitar explicación.

- **IOT:** Internet of Things, traducido al castellano, Internet de las Cosas.
- **LPWAN:** Low Power Wide Area Networks, traducido al castellano, Redes de bajo consumo y de área extensa.
- **GSM:** Global System for Mobile communications, traducido, Sistema global para comunicaciones móviles.
- **LTE-M:** Es un tipo de estándar de tecnología de radio de red desarrollado por *3GPP* para permitir una amplia gama de dispositivos y servicios celulares.
- **NB-IoT:** Es un estándar de tecnología de radio desarrollado por *3GPP* para habilitar servicios de alta gama para dispositivos móviles
- **3GPP:** Es una colaboración de grupos de asociaciones de telecomunicaciones, conocidos como miembros organizativos.
- **KB:** Kilobyte es una unidad de almacenamiento de información.
- **UML:** Unified Modeling Language, en castellano Lenguaje unificado de modelado.
- **Unix Epoch:** Medida de tiempo estándar. Se define como la cantidad de milisegundos transcurridos desde la medianoche UTC del 1 de enero de 1970.

- **PCB:** Printed Circuit Board, traducido al castellano, placa de circuito impreso.
- **Ad hoc:** Es una locución latina que significa literalmente «para esto». Generalmente se refiere a una solución elaborada específicamente para un problema o fin preciso.
- **i2C:** Inter-Integrated Circuit, traducido, Circuito inter-integrado. Es un bus serie de datos.
- **GCC:** GNU Compiler Collection, es un conjunto de compiladores creados por el proyecto GNU.
- **IDE:** Integrated Development Environment, en castellano, entorno de desarrollo integrado o entorno de desarrollo interactivo.
- **SOC:** System on a Chip. Hardware que integra todos o gran parte de los módulos en un único circuito integrado.
- **Código QR:** Evolución del código de barras. Almacena información en una matriz de puntos.
- **BLE:** Bluetooth Low Energy. Es una tecnología de red de área personal inalámbrica.
- **WPA2:** Wi-Fi Protected Access 2, es un sistema para proteger las redes inalámbricas.
- **SPI:** Serial Peripheral Interface. Es un estándar de comunicaciones usado para la transferencia de información.
- **Transceptor:** Dispositivo que cuenta con un transmisor y un receptor que comparten parte de la circuitería.
- **LoraConnect:** Nombre que he puesto a esta solución.
- **Broadcast:** Mensaje de difusión que se envía a todos los destinos posibles.

- **POC:** Proof of Concept, traducido al castellano, Prueba de Concepto, utilizada para ejemplificar una posible implementación.
- **Kernel (de linux):** Núcleo del sistema operativo.
- **SO:** Sistema Operativo.
- **TFM:** Trabajo de Fin de Master.
- **GUI:** En inglés Graphical User Interface, traducido, interfaz gráfica de usuario.

Apéndice B. Código fuente

El código del presente Trabajo Fin de Máster: Integración de Conectividad LoRa en Dispositivos Android, se encuentra en mi repositorio público de GitHub: https://github.com/yacosant/TFM/ con licencia Apache License 2.0

Bibliografía

[1] matooma.com, *Redes LPWAN: ¿cuál es su grado de desarrollo?* https://www.matooma.com/es/informarse/noticias-iot-m2m/redes-lpwan.

[2] R. Hernandez, *The Things Network: ¿Qué es la tecnología LoRa y por qué es importante para IoT?* https://www.thethingsnetwork.org/community/santa-rosa/post/que-es-la-tecnologia-lora-y-por-que-es-importante-para-iot.

[3] www.m2mlogitek.com, *LPWAN: qué son y para qué se utilizan*, https://www.m2mlogitek.com/lpwan-que-son-y-para-que-se-utilizan/.

[4] meshtastic, *Meshtastic: Open Source hiking, pilot, skiing and secure GPS mesh communicator*, https://meshtastic.org/.

[5] pulsartronic, *Hackster.io: DIY Smartphone LoRa connection*, https://www.hackster.io/pulsartronic/diy-smartphone-lora-connection-bde258.

[6] Taiwo, *Hackaday.io: Skrypt*, https://hackaday.io/project/164242-skrypt.

[7] J. Höchst, L. Baumgärtner, F. Kuntke, A. Penning, A. Sterz y B. Freisleben, "LoRa-based Device-to-Device Smartphone Communication for Crisis Scenarios," en *17th International Conference on Information Systems for Crisis Response and Management (ISCRAM 2020) [accepted for publication]*, Blacksburg, Virginia, USA, mayo de 2020.

[8] AlfaIOT, *Blog sobre Internet of Things en donde se explica la tecnología LoRa*, https://alfaiot.com/tecnologias-iot/lora/.

[9] Semtech, *Semtech: Semtech SX1276*, https://www.semtech.com/products/wireless-rf/lora-core/sx1276.

[10] Myandroidsolutions, *Myandroidsolutions: Android tcp connection*, http://www.myandroidsolutions.com/2013/03/31/android-tcp-connection-enhanced/#.YNTdMKgzZhE.

[11] esp32.net, *Imagen $ESP32_{F}unction_{B}lock_{D}iagram.svg$*, http://esp32.net/images/_resources/ESP32_Function_Block_Diagram.svg.

[12] lora-panama.com, *Imagen Logo LoRa*, http://lora-panama.com/wp-content/uploads/2021/02/lora.png.

[13] wiki.dragino.com, *Imagen Logo TCP*, https://wiki.dragino.com/images/0/08/TCP.png.

[14] logodownload.org, *Imagen Logo Arduino*, https://logodownload.org/wp-content/uploads/2019/03/arduino-logo.png.

[15] www.oracle.com, *Imagen Logo Java*, https://www.oracle.com/a/ocom/img/rc30v1-java-se.png.

[16] wikimedia.org, *Imagen Logo Android*, https://upload.wikimedia.org/wikipedia/commons/thumb/6/64/Android_logo_2019_%28stacked%29.svg/1200px-Android_logo_2019_%28stacked%29.svg.png.

[17] logos-world.net, *Imagen Logo GitHub*, https://logos-world.net/wp-content/uploads/2020/11/GitHub-Logo.png.

[18] m2mlogitek.com, *Imagen Logo GitHub*, https://www.m2mlogitek.com/wp-content/uploads/comparativa1.png.

Yaco Alejandro Santiago Pérez - 2020/2021