Conexión de una placa ESP32 a un telefonillo para realizar una comunicación inalámbrica y gestión a través de una aplicación móvil

---

Connection of an ESP32 board to a telephone to perform wireless communication and management through a mobile application

Trabajo Fin de Grado

Sergio Cuenca López: 9
Dorzhi Aylagas Tsyzhipov: 9

Hortensia Mecha
Juan Carlos Fabero

Convocatoria: Extraordinaria

Grado en Ingeniería de Computadores
Facultad de Informática
Universidad Complutense de Madrid

1 de Septiembre de 2025

Agradecimientos a Hortensia y Juan Carlos
por toda la ayuda que nos han aportado.

Gracias a aquellos amigos que pasaron por el
aro antes que nosotros y nos aconsejaron.

Gracias de corazón a todos aquellos que nos
apoyaron desde el inicio y confiaron.

Y por último, gracias a la cafeína, sin ella
este proyecto no hubiese sido posible.

Resumen

Este Trabajo de Fin de Grado presenta el desarrollo de un sistema de telefonillo inteligente basado en la integración de hardware ESP32 y una aplicación móvil multiplataforma desarrollada con Flutter. El objetivo principal ha sido ofrecer una solución moderna, accesible y flexible que permita la comunicación audiovisual bidireccional y el control remoto de acceso desde cualquier dispositivo. A lo largo del proyecto se han abordado aspectos como la selección del entorno de desarrollo, la implementación de protocolos de comunicación eficientes, la integración de tecnologías como mDNS para el descubrimiento automático de dispositivos, y la mejora continua de la experiencia de usuario. El resultado es una plataforma funcional y adaptable, que responde a las necesidades actuales de conectividad y domótica en el hogar.

Palabras clave

ESP-32, mDNS, APP, TCP, UDP, ADC, FEC, NVS, HTTP, I2S

Summary

This Final Degree Project presents the development of a smart intercom system based on the integration of ESP32 hardware and a cross-platform mobile application developed with Flutter. The main objective has been to provide a modern, accessible, and flexible solution that enables bidirectional audiovisual communication and remote access control from any device. Throughout the project, aspects such as the selection of the development environment, the implementation of efficient communication protocols, the integration of technologies such as mDNS for automatic device discovery, and the continuous improvement of the user experience have been addressed. The result is a functional and adaptable platform that meets the current needs of home connectivity and smart home systems.

Keywords

ESP-32, mDNS, APP, TCP, UDP, ADC, FEC, NVS, HTTP, I2S

Índice de contenidos

Capítulo 1 - Introducción 1

1.1 Motivación 1

1.2 Objetivos 1

1.3 Plan de trabajo 2

Capítulo 2 - Introduction 3

2.1 Motivation 3

2.2 Objectives 3

2.3 Work Plan 4

Capítulo 3 - Estado de la cuestión 5

3.1 Evolución de los telefonillos 5

3.2 Mercado actual 5

3.3 Proyectos similares Open Source 5

3.4 Tecnologías de comunicación y descubrimiento 6

3.5 Desarrollo multiplataforma y experiencia de usuario 6

3.6 Retos actuales 6

3.7 Aportaciones al trabajo 7

Capítulo 4 - Protocolo de comunicación 9

4.1 Comandos de control 10

4.1.1 Comandos enviados desde el teléfono a la ESP32 10

4.1.2 Comandos enviados desde la ESP32 a el teléfono 11

4.1.3 Diagramas de comunicación 12

4.2 Formato de los mensajes de audio 12

4.3 Formato de los mensajes de vídeo 13

Capítulo 5 - Configuración WiFi 15

Capítulo 6 - Desarrollo del firmware para la ESP32 17

6.1 Entorno de desarrollo 17

6.2 Librerías Externas 19

6.2.1 FreeRTOS 19

6.2.2 ESP-IDF 19

6.2.3 ESP-ADF 20

6.2.4 ESP32-Cam 21
6.3 Scripts y pruebas realizadas durante el desarrollo 21
6.4 Arquitectura del sistema 24
6.4.1 Componentes Principales 24
6.5 Licencias del software utilizado 30
6.6 Repositorio 30
Capítulo 7 - Hardware utilizado 31
7.1 Características principales de el SOC ESP32-S3-WROOM-1 32
7.2 Audio Codec ES8311 32
7.3 Cámara OV2640 32
Capítulo 8 - Aplicación móvil 35
8.1 Entorno de Desarrollo 35
8.2 Arquitectura y flujo de la aplicación 36
8.3 Lógica de comunicación 38
8.4 Lógica del descubrimiento de vecinos por mDNS 43
8.5 Experiencia de usuario y sincronización Audio-vídeo 44
8.6 Pruebas 45
8.7 Repositorio y licencias 49
8.8 Interfaces 50
Capítulo 9 - Conclusiones y trabajo futuro 57
9.1 Conclusión 57
9.2 Trabajo futuro 57
Capítulo 10 - Conclusions and future work 61
10.1 Conclusion 61
10.2 Future work 61
Bibliografía 71

ÍNDICE DE FIGURAS

Figura 4-1. Diagrama de comunicación normal. 12
Figura 6-1 Funcionalidades idf.py monitor 18
Figura 6-2: Visualización de mensajes por UART mediante idf.py monitor 18
Figura 6-3: Captura de paquetes UDP mediante wireshark. 21
Figura 6-4: Script python de aprovisionamiento wifi. 22
Figura 6-5: Script python de test de audio y vídeo 23
Figura 6-6: Resultado script python test audio y vídeo. 23
Figura 6-7: Diagrama de aprovisionamiento 25
Figura 7-1: ESP32-S3-Korvo-2 V3.1.Fuente: Espressif (enlace) 31
Figura 7-2: ESP32-S3-Korvo-2 con altavoz y OV2640 conectados. 33
Figura 8-1: Diagrama de flujo de la aplicación móvil 37
Figura 8-2: Carpeta del drive con los distintos apks generados 46
Figura 8-3: Búsqueda de la ESP-32 50
Figura 8-4: Menú del aprovisionamiento 51
Figura 8-5: Redes escaneadas (móvil) 52
Figura 8-6: Rd seleccionada (móvil) 53
Figura 8-7: Menú principal 54
Figura 8-8: Interfaz de la llamada 55

Índice de tablas

Tabla 4-1: Formato de los mensajes de audio. 13
Tabla 4-2: Formato de los mensajes de vídeo. 14

Capítulo 1 - Introducción

1.1 Motivación

Hoy en día, la integración de sistemas inteligentes en el hogar es una tendencia en crecimiento, impulsada por la búsqueda de mayor comodidad, seguridad y eficiencia. Los telefonillos tradicionales presentan límites en cuanto a la accesibilidad remota y funcionalidades avanzadas. Por ello, surge la necesidad de desarrollar una solución moderna y flexible que permita la comunicación audiovisual bidireccional y el control de acceso desde cualquier dispositivo, aprovechando tecnologías como Flutter, ESP32 y redes WiFi.

1.2 Objetivos

El objetivo principal de este trabajo es diseñar e implementar un sistema de telefonillo inteligente basado en ESP32 y una aplicación móvil multiplataforma desarrollada con Flutter. Los objetivos específicos son:

- Permitir la comunicación audiovisual bidireccional entre el usuario y el dispositivo ESP32.
- Implementar un descubrimiento automático de dispositivos mediante mDNS.
- Facilitar la configuración WiFi y la gestión de la red desde la app.
- Permitir la apertura remota de la puerta desde la aplicación.
- Garantizar una experiencia de usuario moderna, intuitiva y multiplataforma.

1.3 Plan de trabajo

El desarrollo del proyecto se ha estructurado en las siguientes fases:

1. Revisión del estado del arte y análisis de requisitos.
2. Diseño de la arquitectura del sistema (hardware y software).
3. Implementación del firmware para ESP32 y de la aplicación Flutter.
4. Integración de funcionalidades clave: mDNS, comunicación TCP/UDP, audio y vídeo, configuración WiFi y control de acceso.
5. Pruebas, validación y mejoras constantes a los resultados.
6. Documentación y redacción de la memoria del proyecto.

Capítulo 2 - Introduction

2.1 Motivation

Nowadays, the integration of intelligent systems in the home is a growing trend, driven by the pursuit of greater comfort, security, and efficiency. Traditional intercoms present significant limitations in terms of remote accessibility and advanced functionalities. Therefore, there is a need to develop a modern and flexible solution that enables bidirectional audiovisual communication and access control from any device, leveraging technologies such as Flutter, ESP32, and WiFi networks.

2.2 Objectives

The main objective of this project is to design and implement a smart intercom system based on ESP32 and a cross-platform mobile application developed with Flutter. The specific objectives are:

- Enable bidirectional audiovisual communication between the user and the ESP32 device.
- Implement automatic device discovery using mDNS.
- Facilitate WiFi provisioning and network management from the app.
- Allow remote door unlocking through the application.
- Ensure a modern, intuitive, and cross-platform user experience.

2.3 Work Plan

The development of the project has been structured into the following phases:

1. Review of the state of the art and requirements analysis.
2. Design of the system architecture (hardware and software).
3. Implementation of the firmware for the ESP32 and the Flutter application.
4. Integration of key functionalities: mDNS, TCP/UDP communication, audio and vídeo, WiFi provisioning, and access control.
5. Testing, validation, and iterative improvements of the solution.
6. Documentation and writing of the project report.

Capítulo 3 - Estado de la cuestión

3.1 Evolución de los telefonillos

Los vídeoporteros tradicionales han sido durante décadas una solución estándar para el control de acceso en viviendas y edificios. Sin embargo, su funcionalidad suele estar limitada a la comunicación local y a la apertura de puertas mediante cableado físico, sin posibilidad de acceso remoto ni integración con otros sistemas inteligentes.

3.2 Mercado actual

En el mercado existen múltiples vídeoporteros inteligentes, como Ring, Fermax o BTicino, que ofrecen funcionalidades avanzadas como la visualización remota desde el móvil, notificaciones en tiempo real y grabación en la nube. No obstante, la mayoría de estas soluciones son sistemas cerrados, con poca flexibilidad para la personalización, y en muchos casos requieren suscripciones o infraestructuras propietarias.

3.3 Proyectos similares Open Source

La aparición de microcontroladores como el ESP32 ha permitido a la comunidad desarrollar soluciones personalizadas y de bajo coste. Existen numerosos proyectos open source que integran ESP32 con cámaras, sensores y aplicaciones móviles, facilitando la comunicación por WiFi y el control remoto. Sin embargo, muchos de estos proyectos carecen de una interfaz de usuario pulida o de mecanismos sencillos de configuración y descubrimiento de dispositivos.

3.4 Tecnologías de comunicación y descubrimiento

El uso de protocolos como TCP/UDP para la transmisión de audio y vídeo es común en sistemas de comunicación en tiempo real. Además, tecnologías como mDNS (Multicast DNS) permiten el descubrimiento automático de dispositivos en la red local, simplificando la experiencia de usuario al evitar configuraciones manuales de direcciones IP o puertos.

La configuración WiFi, es decir, la configuración inicial de la red inalámbrica en dispositivos IoT, sigue siendo un reto en términos de facilidad y seguridad para el usuario final.

3.5 Desarrollo multiplataforma y experiencia de usuario

Entornos como Flutter han revolucionado el desarrollo de aplicaciones móviles, permitiendo crear apps modernas y funcionales para Android, iOS y web desde una única base de código. Esto facilita la integración de nuevas funcionalidades y la adaptación a diferentes dispositivos, mejorando la accesibilidad y la experiencia de usuario.

3.6 Retos actuales

A pesar de los avances en domótica y el aumento de soluciones inteligentes para el hogar, todavía existen desafíos importantes en el desarrollo de telefonillos conectados. La integración entre dispositivos de distintos fabricantes resulta compleja debido a la falta de estándares abiertos y la fragmentación del mercado. Además, la seguridad y privacidad en la transmisión de datos requieren mecanismos robustos de cifrado y autenticación para proteger la información del usuario.

Por otro lado, la facilidad de uso sigue siendo un aspecto fundamental, ya que los procesos de configuración, aprovisionamiento y descubrimiento deben ser lo más intuitivos posibles para usuarios sin experiencia técnica. Finalmente, es esencial que las soluciones sean escalables, fácilmente actualizables y asequibles, permitiendo su adaptación a nuevas necesidades, sin suponer grandes costes ni complicaciones técnicas.

3.7 Aportaciones al trabajo

Este proyecto se podría decir que proporciona una solución abierta y flexible a cambios, proponiendo un sistema de telefonillo inteligente basado en ESP32 y Flutter, que integra descubrimiento automático, aprovisionamiento sencillo, comunicación audiovisual bidireccional y una interfaz moderna, todo ello sin depender de servicios propietarios ni suscripciones.

Capítulo 4 - Protocolo de comunicación

El protocolo de comunicación empleado está basado en RTP (Realtime Transport Protocol), aunque simplificado de forma significativa para facilitar la implementación. En nuestra adaptación se mantienen elementos esenciales para la transmisión en tiempo real de audio y vídeo, pero hemos omitido de forma deliberada funcionalidades avanzadas de RTP que no son necesarias para el contexto de este trabajo (corrección de errores hacia adelante, calidad de audio y vídeo adaptable según las condiciones de red, entre otras).

Para garantizar una baja latencia en la transmisión de audio y vídeo, el transporte de estos se realiza sobre UDP, cuya principal ventaja es la velocidad. Cabe mencionar que esto también puede traer problemas, ya que en malas condiciones de red podríamos tener pérdida de paquetes. Para subsanar este problema, se puede hacer uso de corrección de errores hacia adelante (Forward Error Correction, abreviado como FEC). No todos los mensajes se pueden enviar por UDP, hay mensajes en los que necesitamos asegurar la recepción. Estos serán los mensajes de control enviados por TCP, con los que controlaremos el flujo de datos entre la esp32 y el teléfono.

En este capítulo detallaremos los comandos disponibles para el control, el formato de los mensajes, la recuperación de paquetes perdidos y una comparación con RTP.

4.1 Comandos de control

En esta sección se describirán los distintos comandos de control intercambiados entre la ESP32 y el teléfono, como ya se ha mencionado previamente, estos se envían sobre TCP, ya que es necesaria una entrega fiable, ordenada y confirmada. La principal función de estos comandos es coordinar la comunicación, habilitando y finalizando sesiones de audio/vídeo; mediante estos también podemos notificar eventos al cliente.

El formato de estos comandos es muy sencillo, se envía un número entero de 4B donde cada número indicará un comando distinto.

4.1.1 Comandos enviados desde el teléfono a la ESP32

En esta sección sólo encontraremos los comandos enviados desde el teléfono, las respuestas a estos estarán especificadas en la siguiente sección.

COMANDOS:

- **CMD_REQUEST_TALK (0) :** el usuario solicita iniciar una sesión de comunicación.
- **CMD_END_TALK (1) :** el usuario señala que la sesión de comunicación va a terminar.
- **CMD_OPEN_DOOR (7) :** el usuario solicita que se abra la puerta (lo simularemos enviando un mensaje por UART ya que no tenemos un telefonillo real).

4.1.2 Comandos enviados desde la ESP32 a el teléfono

Estos comandos son respuestas o notificaciones que informan al usuario del estado del sistema.

- **COMANDOS DE RESPUESTA**

    - **CMD_GRANT_TALK (2) :** informa a la aplicación de que la petición de conversación ha sido aceptada. (Respuesta a CMD_REQUEST_TALK)
    - **CMD_DENY_TALK (3) :** informa a la aplicación de que la petición de conversación ha sido denegada. Esto puede suceder si hay otro cliente que ya esté comunicándose con la ESP32. (Respuesta a CMD_REQUEST_TALK)
    - **CMD_TALK_ENDED (4) :** informa a la aplicación que la conversación ha finalizado correctamente (respuesta a CMD_END_TALK).
    - **CMD_TALK_DID_NOT_END (5) :** informa a la aplicación que la conversación no ha podido finalizar. Se responderá con este comando en el caso de que se intente terminar la conversación sin ser el cliente que envía el comando de CMD_END_TALK el que está teniendo la conversación o en caso de que no haya ninguna conversación en curso (respuesta a CMD_END_TALK).

- **COMANDO DE NOTIFICACIÓN**

    - **CMD_DOORBELL_RING (6) :** informa a la aplicación de que se ha presionado el botón del telefonillo (simulado por un botón de la ESP32).

4.1.3 Diagramas de comunicación

En este diagrama podemos ver un ejemplo de comunicación normal entre la ESP-32 y el teléfono.

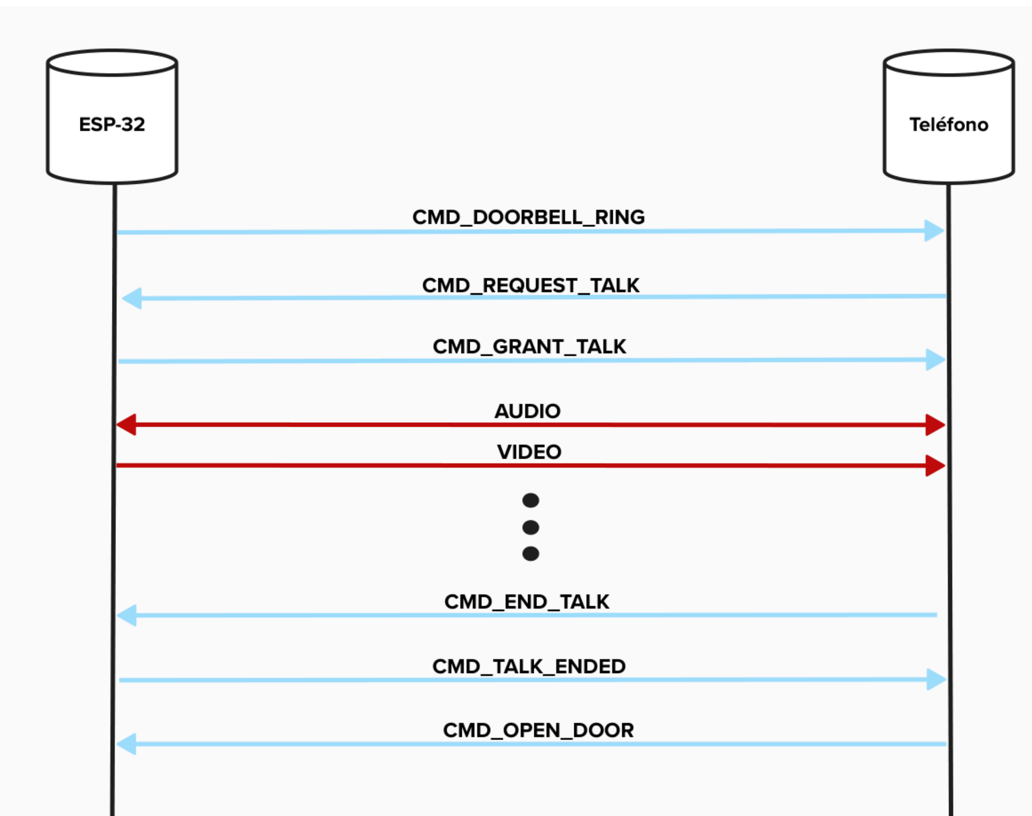
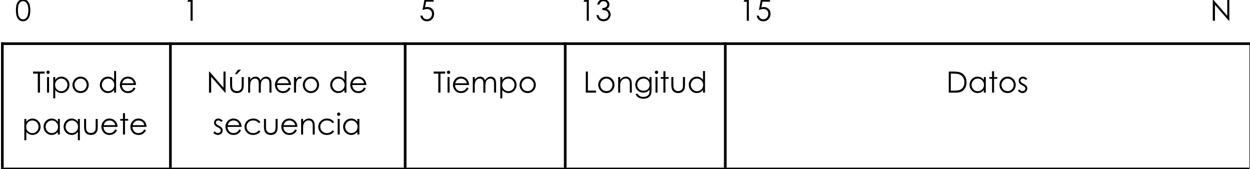


Figura 4-1. Diagrama de comunicación normal.

4.2 Formato de los mensajes de audio

A continuación en la tabla se describirán los distintos campos y las longitudes de estos, ordenados desde el byte menos significativo hasta el más significativo, de los paquetes de audio.



0	1	5	13	15 N
Tipo de paquete	Número de secuencia	Tiempo	Longitud	Datos

Tabla 4-1: Formato de los mensajes de audio.

- **TIPO DE PAQUETE:** específica qué contenido se encontrará en ese paquete, en el caso del audio, este campo tomará el valor 0x00.

- **NÚMERO DE SECUENCIA:** este número identifica el paquete enviado, el primer paquete enviado será el 0 e irá incrementando de uno en uno.

- **TIEMPO:** indica el número de milisegundos transcurridos desde *Epoch*.

- **LONGITUD:** indica el tamaño de los datos en bytes.

- **DATOS:** este campo contiene los datos del audio.

4.3 Formato de los mensajes de vídeo

Los paquetes de vídeo son muy parecidos, pero hay una pequeña diferencia, al no poder enviar una imagen entera y tener que enviar los fotogramas fragmentados debido al tamaño de MTU, el formato será ligeramente distinto. En la siguiente tabla se muestran los distintos campos, ordenados desde el byte menos significativo al más significativo.

0	1	5	13	15	17	19 N
Tipo de paquete	Número de fotograma	Tiempo	Longitud	Número de secuencia del paquete	Número total de paquetes del fotograma	Datos

Tabla 4-2: Formato de los mensajes de vídeo.

- **TIPO DE PAQUETE:** específica qué contenido se encontrará en ese paquete, en el caso del vídeo, este campo tomará el valor 0x01.

- **NÚMERO DE FOTOGRAMA:** este número identifica el fotograma enviado, el primer fotograma enviado será el 0 e irá incrementando de uno en uno.

- **TIEMPO:** indica el número de milisegundos transcurridos desde *Epoch*.

- **LONGITUD:** indica el tamaño de los datos en bytes.

- **NÚMERO DE SECUENCIA DEL PAQUETE:** indica el número de secuencia del paquete perteneciente al fotograma especificado en el número de fotograma.

- **DATOS:** este campo contiene los datos del vídeo.

Capítulo 5 - Configuración WiFi

El aprovisionamiento de WiFi permite configurar la red inalámbrica del ESP32 desde la app, facilitando la conexión del dispositivo a la red local del usuario sin necesidad de interfaces físicas.

- **¿Cómo funciona el aprovisionamiento?**

1. Modo AP del ESP32:
   Cuando el ESP32 no está configurado, crea su propia red WiFi (modo Access Point, por ejemplo, ESP32-XXXX).
2. Conexión del móvil:
   El usuario conecta su móvil a la red WiFi del ESP32 desde los ajustes del sistema.
3. Escaneo de redes:
   La app, conectada al AP del ESP32, realiza una petición HTTP a http://192.168.4.1/scan para obtener la lista de redes WiFi cercanas detectadas por el ESP32.
4. Selección y envío de credenciales:
   El usuario selecciona la red deseada e introduce la contraseña. La app envía estos datos mediante una petición POST a http://192.168.4.1/config.
5. Cambio de red:
   Tras recibir las credenciales, el ESP32 se reinicia o cambia su modo de red, conectándose a la WiFi seleccionada y dejando de estar accesible en 192.168.4.1.

6. Reconexión:

   El usuario debe volver a conectar su móvil a la red WiFi habitual. La app puede entonces buscar el dispositivo por mDNS en la red local.

- **Consideraciones técnicas**

El endpoint /scan devuelve un listado de SSID, RSSI y tipo de seguridad.

El endpoint /config recibe un JSON con el SSID y la contraseña.

Es normal que tras el POST a /config la conexión se aborte, ya que el ESP32 apaga el AP.

La app muestra un mensaje indicando que el aprovisionamiento se ha enviado y que el usuario debe volver a su red WiFi habitual cuando el aprovisionamiento ha sido exitoso. En caso contrario se mostrará un mensaje de error y se mantendrá el punto de acceso creado por la ESP32 abierto, permitiendo reintentar el aprovisionamiento.

Capítulo 6 - Desarrollo del firmware para la ESP32

En este capítulo se describe el firmware desarrollado para la ESP32 que habilitará la comunicación entre la placa y la aplicación móvil. El firmware sigue un enfoque modular y escalable, de tal manera que se puedan integrar de una manera sencilla nuevas funcionalidades.

Para su implementación se han empleado diversos entornos que serán descritos en las siguientes secciones de este capítulo, siendo el más importante de estos ESP-IDF, construido sobre el sistema operativo de tiempo real FreeRTOS.

6.1 Entorno de desarrollo

Para desarrollar el firmware el editor de texto usado ha sido VS Code, escogimos este editor porque permitía instalar ESP-IDF de una forma sencilla gracias a las extensiones.

Para compilar el proyecto, ver los mensajes enviados por UART, modificar la configuración de la placa y sobreescribir el firmware, se utiliza el script de Python proporcionado por el SDK de ESP-IDF, idf.py. Este script emplea Ninja como sistema de construcción, y Ninja, a su vez, requiere CMake para generar los archivos de construcción necesarios.

```
--- esp_idf_monitor (1.6.2) - ESP-IDF Monitor tool
--- based on miniterm from pySerial
---
--- Ctrl+]   Exit program
--- Ctrl+T   Menu escape key, followed by:
--- Menu keys:
---    Ctrl+T         Send the menu character itself to remote
---    Ctrl+]         Send the exit character itself to remote
---    Ctrl+R         Reset target board via RTS line
---    Ctrl+F         Build & flash project
---    Ctrl+A (or A)  Build & flash app only
---    Ctrl+Y         Toggle output display
---    Ctrl+L         Toggle saving output into file
---    Ctrl+I (or I)  Toggle printing timestamps
---    Ctrl+P         Reset target into bootloader via the DTR/RTS lines
---    Ctrl+X (or X)  Exit program
```

Figura 6-1 Funcionalidades idf.py monitor

Al no tener JTAG de forma nativa en la placa (se puede activar moviendo dos resistencias, pero no tenemos la habilidad suficiente y solo teníamos una placa), hemos tenido que hacer uso de trazas por UART para poder depurar.

```
I (1192) WIFI_PROV: Attempting to load WiFi credentials from NVS...
I (1202) WIFI_PROV: No WiFi SSID found in storage
I (1202) WIFI_PROV: No WiFi credentials found, starting provisioning mode...
I (1212) phy_init: phy_version 680,a6008b2,Jun  4 2024,16:41:10
I (1252) wifi:mode : sta (8c:bf:ea:0b:3e:40) + softAP (8c:bf:ea:0b:3e:41)
I (1252) wifi:enable tsf
I (1252) wifi:Total power save buffer number: 16
I (1252) wifi:Init max length of beacon: 752/752
I (1262) wifi:Init max length of beacon: 752/752
I (1262) esp_netif_lwip: DHCP server started on interface WIFI_AP_DEF with IP: 192.168.4.1
I (1262) WIFI_PROV: WiFi APSTA started: SSID=ESP32-Setup-3E41, Password=123456789aSdF!_$
I (1272) WIFI_PROV: WiFi AP started - Provisioning mode active
I (1292) WIFI_PROV: HTTP provisioning server started on port 80
I (1292) WIFI_PROV: Waiting for WiFi connection...
```

Figura 6-2: Visualización de mensajes por UART mediante idf.py monitor

6.2 Librerías Externas

6.2.1 FreeRTOS

FreeRTOS es un sistema operativo de tiempo real para sistemas embebidos. Este sistema operativo introduce el concepto de tareas concurrentes y también da diversos mecanismos de comunicación y sincronización para estas. En FreeRTOS, las unidades básicas de ejecución son las tareas, que estas son equivalentes a los hilos de otros sistemas operativos.

Como hemos mencionado previamente, en el caso de la ESP32, FreeRTOS está integrado de forma nativa sobre el framework de desarrollo oficial de Espressif (ESP-IDF), permitiendo distribuir la aplicación en distintas tareas con distintas responsabilidades, encargándose el sistema operativo de la planificación y gestión de los recursos.

6.2.2 ESP-IDF

ESP-IDF constituye la base fundamental del firmware desarrollado para este proyecto, al proporcionar múltiples capas de abstracción que simplifican el proceso de desarrollo.

Entre las diversas funciones que aporta el firmware, las más destacables para este proyecto son las siguientes:

- **Driver WiFi:** proporciona conectividad inalámbrica, soportando varios modos de operación. Los modos de operación usados en este proyecto son STA y AP+STA. El modo STA permite conectarte a otros puntos de acceso, y el modo AP+STA hace que la ESP32 actúe como un punto de acceso, permitiendo también conectarse a otras redes, al igual que en el modo STA. Esto resultó de mucha ayuda para poder aprovisionar WiFi a la placa.

- **Stack TCP/IP:** proporciona la API de sockets POSIX, gestionando las conexiones UDP para streaming de audio y vídeo en tiempo real, y las conexiones TCP para los comandos de control.
- **Servicio mDNS:** facilita el descubrimiento automático del dispositivo en la red local, eliminando la necesidad de establecer una IP estática para poder localizar la placa.
- **Sistema NVS (Non-Volatile Storage):** permite que las credenciales WiFi obtenidas mediante el aprovisionamiento persistan tras un corte de alimentación. De esta forma el dispositivo puede reconectarse automáticamente a la red sin necesidad de repetir el proceso de configuración.
    - **Servidor HTTP:** el entorno proporciona una API con la que hemos implementado un servidor web ligero con el que podemos enviar los resultados del escaneo de redes, y con el que podemos recibir las credenciales necesarias para conectarnos a una red, siendo fundamental para el aprovisionamiento de WiFi.

      Aunque podríamos haber optado por el uso de HTTPS, hemos decidido dejar su implementación, junto con otras mejoras de seguridad como trabajo futuro.

6.2.3 ESP-ADF

ESP-ADF (ESP - Advanced Development Framework) constituye la capa especializada de audio que se integra sobre ESP-IDF, proporcionando las herramientas y componentes necesarios para poder implementar el sistema de comunicación de audio de este proyecto.

Las *pipelines* de audio son el núcleo de ESP-ADF. Permiten conectar distintos elementos de audio de forma modular y secuencial, permitiendo crear flujos de

procesamiento complejos. Los elementos de audio realizan funciones específicas y pueden ser de captura, procesamiento, transmisión o reproducción.

Esta arquitectura ha resultado fundamental para poder implementar de forma sencilla la captura, transmisión y recepción por UDP y la reproducción de audio.

6.2.4 ESP32-Cam

ESP32-Cam es un módulo de ESP-IDF que permite obtener fotogramas de la cámara, abstrayendo la complejidad del hardware. Este framework tiene soporte para varias cámaras, entre ellas, encontramos la OV2640, que es la cámara que hemos usado. Para que el módulo funcionase correctamente tuvimos que habilitar el uso de PSRAM en las opciones de idf.py menuconf. Aunque no era obligatorio, también cambiamos la velocidad de operación de la PSRAM a 80 Mhz que es el máximo permitido por la memoria del microprocesador.

6.3 Scripts y pruebas realizadas durante el desarrollo

Para probar el envío de audio y vídeo se hizo uso de varias herramientas, antes de tener scripts desarrollados y cuando el firmware era más básico (envío unidireccional de audio), usamos wireshark para comprobar que los paquetes se estaban enviando desde la placa al ordenador. (Ejemplo en la figura 6-3)

No.	Time	Source	Destination	Protocol	Length	Info
3	6.840847223	fe80::c5fd:a822:1b2…	ff02::fb	MDNS	92	Standard query 0x0000 A telrem.local, "QM" question
4	6.840870239	10.249.204.189	224.0.0.251	MDNS	72	Standard query 0x0000 A telrem.local, "QM" question
5	7.029765225	10.249.204.190	224.0.0.251	MDNS	82	Standard query response 0x0000 A, cache flush 10.249.204.190
14	7.454481320	10.249.204.190	10.249.204.189	UDP	1442	58556 → 12346 Len=1400
15	7.454482296	10.249.204.190	10.249.204.189	UDP	1442	58556 → 12346 Len=1400
16	7.454482448	10.249.204.190	10.249.204.189	UDP	1442	58556 → 12346 Len=1400
17	7.454524522	10.249.204.190	10.249.204.189	UDP	381	58555 → 12345 Len=339
18	7.454525352	10.249.204.190	10.249.204.189	UDP	381	58555 → 12345 Len=339
19	7.454525500	10.249.204.190	10.249.204.189	UDP	381	58555 → 12345 Len=339
20	7.454525657	10.249.204.190	10.249.204.189	UDP	381	58555 → 12345 Len=339
21	7.454525806	10.249.204.190	10.249.204.189	UDP	381	58555 → 12345 Len=339
22	7.454525957	10.249.204.190	10.249.204.189	UDP	381	58555 → 12345 Len=339
23	7.454526105	10.249.204.190	10.249.204.189	UDP	381	58555 → 12345 Len=339
24	7.454526256	10.249.204.190	10.249.204.189	UDP	381	58555 → 12345 Len=339
25	7.454726631	10.249.204.190	10.249.204.189	UDP	381	58555 → 12345 Len=339
26	7.455680130	10.249.204.190	10.249.204.189	UDP	381	58555 → 12345 Len=339
27	7.455680843	10.249.204.190	10.249.204.189	UDP	381	58555 → 12345 Len=339
28	7.455680999	10.249.204.190	10.249.204.189	UDP	381	58555 → 12345 Len=339

Figura 6-3: Captura de paquetes UDP mediante wireshark.

Más adelante desarrollamos scripts en python para probar las distintas funcionalidades. El primero fue el script para probar que el aprovisionamiento de wifi funcionaba correctamente. El funcionamiento es muy sencillo, primero envía un GET http a http://192.168.4.1/scan para que la ESP32 realice un scan de las redes wifi que tiene a su alcance, al recibirlo, muestra el resultado y permite elegir una de las redes mostradas, después pide que se introduzca la contraseña, y envía un POST request http a http://192.168.4.1/config con las credenciales, por último la ESP32 responde si ha conseguido conectarse o no. (Ejecución en la figura 6-4)

```
sergio@sergio:~/tfg_project/python_server$ python simple_provisioning.py
============================================================
Simple ESP32 WiFi Provisioning
============================================================

Would you like to scan for available WiFi networks? (y/n, default: y):
Scanning for available WiFi networks...
   Found 20 WiFi networks:
   ======================================================================
   SSID                      Signal   Channel  Security
   ----------------------------------------------------------------------
    1. MOVISTAR_6350_Yaiza     ----- Ch3  WPA2
    2. POCO X6 Pro 5G          ----- Ch13 WPA2
    3. Planta-alta-2.4         ----  Ch2  WPA2
    4. xiaomi-repeater-v2_miap31e4 ---   Ch1  Open
    5. Solax_SWPV6E2U3W        --    Ch1  Open
    6. SUN2000-102160158033    --    Ch4  WPA2
    7. MOVISTAR-WIFI6-03B0     --    Ch6  WPA2
    8. Chovos2                 --    Ch1  WPA2
    9. DigiDufyHalo            --    Ch11 WPA2
   10. Chovos2                 --    Ch1  WPA2
   11. vodafoneD82A            --    Ch1  WPA/WPA2
   12. zhangxueyou             --    Ch1  WPA2
   13. DIT DIT                 --    Ch3  WPA2
   14. DIGIFIBRA-6c4Z          --    Ch8  WPA/WPA2
   15. SEC_LinkShare_d4a0e2    --    Ch9  WPA2
   16. MOVISTAR_73A0-EXT       -     Ch1  WPA2
   17. MIWIFI_js3h             -     Ch6  WPA2
   18. vodafoneBA1513          -     Ch13 WPA2
   19. MOVISTAR_73A0           -     Ch1  WPA2
   20. bRfiCp2ktWfZIJ2GTCuJXf4S8hcgTo2J -     Ch1  Unknown
   ======================================================================

Select network by number, or type SSID manually (or 'q' to quit): 2
   Selected: 'POCO X6 Pro 5G' (WPA2)
Enter password for 'POCO X6 Pro 5G': 12341234

Starting provisioning for ESP32 at 192.168.4.1
WiFi SSID: POCO X6 Pro 5G
WiFi Password: ********


 Sending WiFi credentials...
   SUCCESS! ESP32 connected to WiFi network
   Provisioning completed successfully

============================================================
PROVISIONING SUCCESSFUL!
   ESP32 is now connected to the WiFi network.
============================================================
```

Figura 6-4: Script python de aprovisionamiento wifi.

Por último, desarrollamos un script de python que recibe audio y vídeo de la ESP-32. Muestra el vídeo y reenvía el audio recibido a la ESP-32, probando de esta forma el envío de vídeo, y el envío y recepción de audio, ya que si se reproduce el audio en el altavoz de la placa, el formato del audio es correcto. Al finalizar la ejecución, el script muestra métricas relevantes sobre los paquetes recibidos. (Figura 6-6)

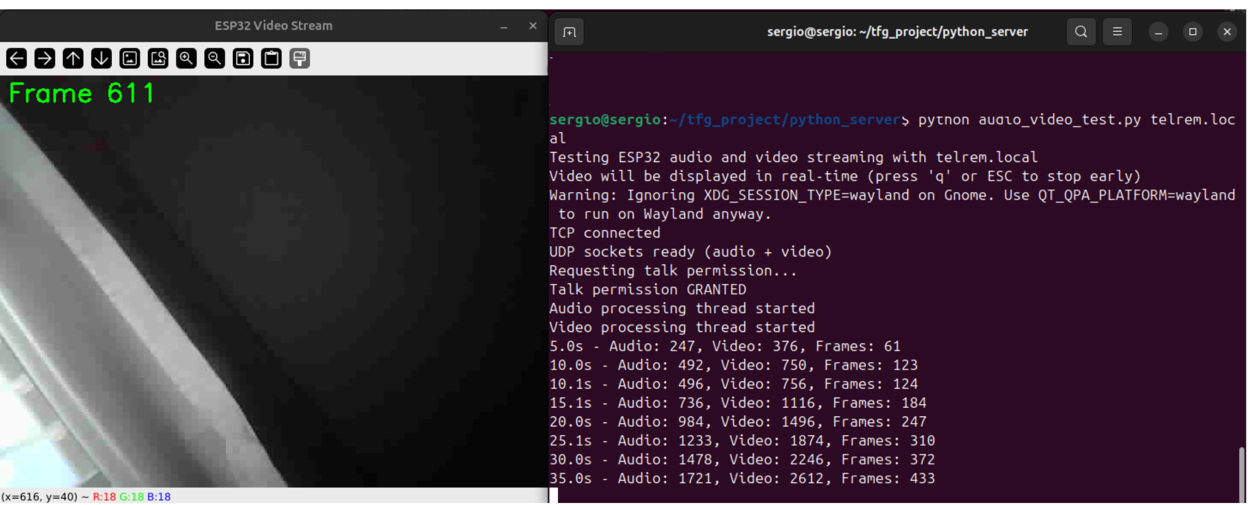


Figura 6-5: Script python de test de audio y vídeo

```
Test Results:
  Audio packets: 418
  Video packets: 625
  Completed video frames: 104
  Unique frames seen: 105
  Frame completion rate: 104/105 (99.0%)
  Audio rate: 49.2 packets/sec
  Video rate: 73.6 packets/sec
  Video frame rate: 12.2 frames/sec
  Expected audio rate: 49.4 packets/sec
```

Figura 6-6: Resultado script python test audio y vídeo.

6.4 Arquitectura del sistema

En este capítulo se describe la arquitectura del sistema desarrollado para el proyecto de comunicación de audio y vídeo con ESP32. La arquitectura está diseñada para ser modular, eficiente y escalable, permitiendo la integración de múltiples funcionalidades como transmisión de audio y vídeo, gestión de periféricos y conectividad de red.

6.4.1 Componentes Principales

- **Módulo de conectividad de red**
    - **Aprovisionamiento de wifi:** permite configurar el wifi de la ESP32, implementa un servidor de HTTP ligero con dos endpoints, /config recibe mediante un mensaje POST las credenciales a las que se intentará conectar (Diagrama en figura 6-7), y /scan, donde mediante un mensaje GET enviará un scan de las redes wifi accesibles por la ESP32.

      Este módulo aporta dos funciones, una para empezar el aprovisionamiento de wifi, y otra para poder borrar las credenciales guardadas en flash y reiniciar la aplicación.

    - **Servicio mDNS:** permite que la ESP32 sea descubierta en la red local sin necesidad de conocer su dirección IP. Esto se logra mediante la resolución de nombres de dominio en redes locales, utilizando el protocolo mDNS. En este proyecto, el servicio mDNS está configurado para que el dispositivo sea accesible mediante el nombre telrem.local.

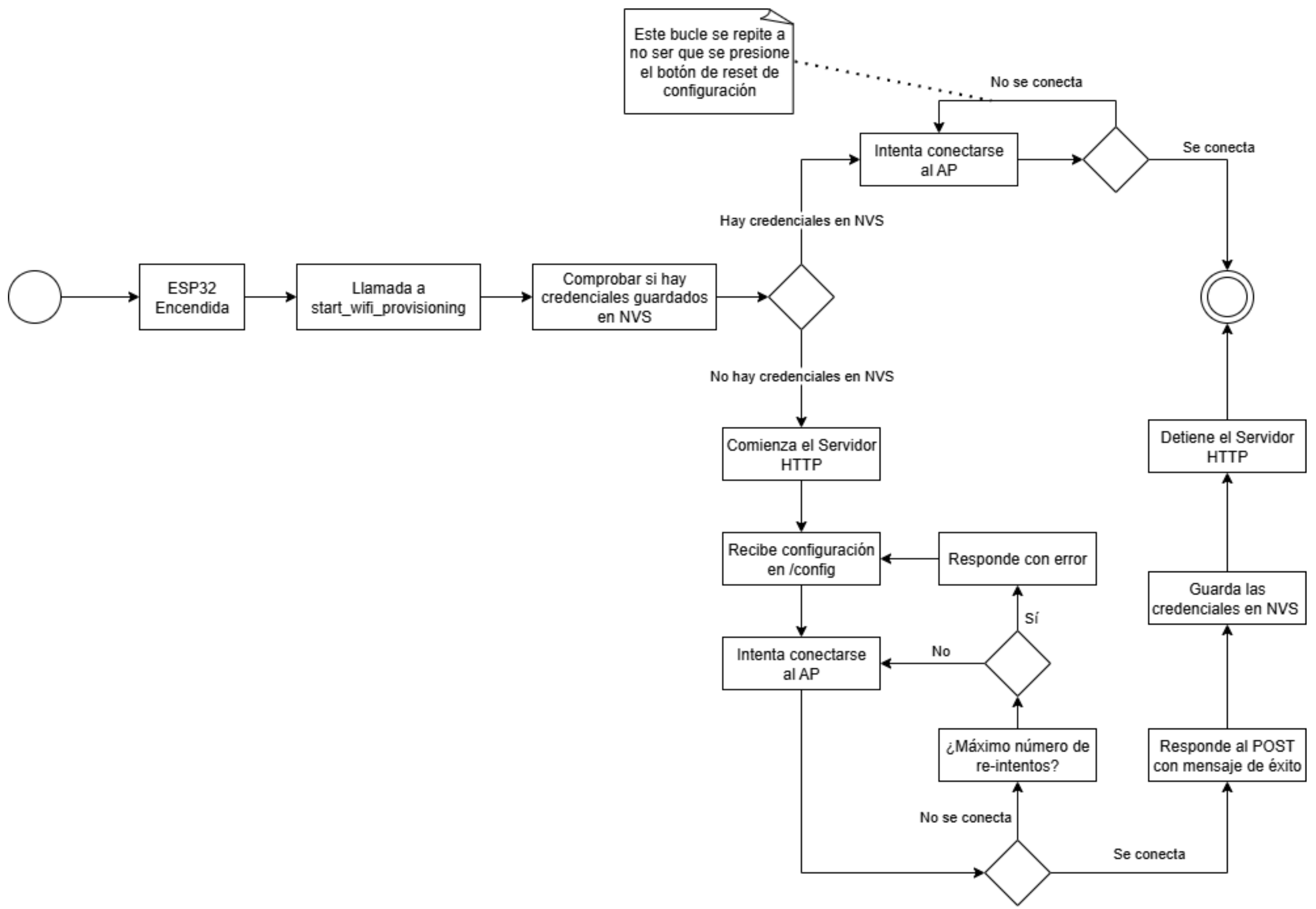


Figura 6-7: Diagrama de aprovisionamiento

- **Módulo de gestión de pipelines de audio:**
  - Este módulo permite gestionar las pipelines de audio, aportando las funciones necesarias para la inicialización, donde se configuran los distintos elementos de audio, y para la detención de estas.

    Hay dos pipelines de audio, una dedicada al envío y la otra a la recepción.
    - La pipeline de envío está compuesta por dos elementos, un lector i2s, que se encarga de leer el audio del micrófono conectado al audio codec, y un escritor udp, que recibe el audio del lector i2s,

le añade la cabecera descrita en el capítulo 5, y la envía a la ip configurada.

- La pipeline de recepción está compuesta por dos elementos, un lector udp, que recibe audio con el formato descrito en el capítulo 5, y un escritor i2s, que envía al audio codec el audio recibido por el lector udp.

- Configuraciones relevantes de los elementos de audio:
  - Elementos I2S:
    - Frecuencia de muestreo de 8 KHz.
    - 16 bits por muestra.
    - Formato mono.
    - Muestreo del canal izquierdo.
    - Tamaño de buffer del elemento lectura: 324 B (para tener una latencia baja, equivale a 20,2 ms de audio aproximadamente).
    - Tamaño de buffer del elemento de escritura: 1404 B (para poder reproducir el máximo tamaño de paquete aceptado por el lector de UDP. No se usó un tamaño de 1400 B porque los elementos de audio I2S requieren que el tamaño del buffer sea múltiplo de 12).
    - Amplificación de 30 db en la captura y reproducción de audio.
  - Elementos UDP:
    - Dirección ip a la que enviar o de la que recibir datos.

- Tamaño de buffer del elemento de lectura: 1400 B. (Necesario para poder recibir paquetes de audio con tamaño variable).
- Tamaño del buffer de elemento de escritura: 324 B (Igual que el elemento I2S de lectura).
- Tamaño de la pila de las tareas de los elementos: 4 KB (Necesario para poder usar sockets UDP y el manejo de la comunicación).

- **Módulo de gestión de vídeo:**
  - Este módulo se encarga de la captura, procesamiento y envío de los fotogramas de vídeo, garantizando que el tamaño de los paquetes no exceda la MTU, evitando así su fragmentación o rechazo. Aunque en una red local podríamos delegar esta tarea en la capa de red (IP), realizar la fragmentación en la capa de aplicación nos permitiría añadir nuevas funcionalidades, como la corrección de errores hacia adelante, de forma más sencilla y eficiente. En lugar de calcular los paquetes redundantes a partir de varios fotogramas, se generarían directamente a partir de los fragmentos de un único fotograma, lo que reduce de forma significativa la latencia necesaria para su reconstrucción.

    Para la captura era imprescindible tener una cámara compatible con compresión hardware para el formato JPEG, de otra forma, tendríamos que haber hecho la compresión mediante software, que es mucho más lenta y limitaría bastante el número de fotogramas por segundo.

- **Módulo de gestión de periféricos:**
  - Este módulo se encarga de la inicialización del codec de audio y de la gestión de dos botones con diferentes funciones. El primero permite reiniciar el aprovisionamiento de wifi, (usa la función para borrar las credenciales y reiniciar del módulo de aprovisionamiento). El segundo simula el botón de llamada del portero, que envía a todos los clientes conectados por TCP un mensaje indicando que el botón se ha presionado.

    Para evitar reinicios no deseados, el botón de reset de configuración solo funcionará si se ha presionado por más de 3 segundos. Asimismo, para evitar el envío de varias notificaciones de llamada seguidas, se ignorarán pulsaciones durante un periodo de 10 segundos desde la primera activación, de modo que únicamente se envíe notificación en ese periodo.
- **Módulo de control**
  - Este módulo coordina las operaciones del sistema interaccionando con los módulos de gestión de audio y vídeo para la inicialización e interrupción de estos. También gestiona las conexiones TCP que permiten a los clientes controlar las funciones descritas en el capítulo 4, admitiendo hasta 5 conexiones simultáneas. Cada conexión se maneja en una tarea independiente, lo que facilita su gestión concurrente. Para acceder de forma segura a los recursos compartidos (audio, vídeo, información de clientes, etc.) se emplean los mecanismos de sincronización de FreeRTOS, siendo los semáforos el recurso más utilizado.

- **Elemento de audio UDP Stream**
    - ESP-ADF no contaba con un componente de audio capaz de enviar datos por UDP, y obviamente tampoco soportaba el formato utilizado por nuestro protocolo. Por esta razón, implementamos un elemento personalizado para cubrir estas funcionalidades. Para integrarlo en el framework, fue necesario modificar el CMake de la carpeta de componentes, añadiendo nuestro nuevo elemento para que fuera reconocido durante la compilación.
    - El comportamiento de un elemento se define mediante una estructura de *callbacks* que se registran en el momento de su creación (open, close, destroy, process, write, read). El orden típico de llamadas cuando un pipeline se ejecuta es el siguiente:

        - Open: se invoca al inicializar el elemento. En el caso del elemento UDP implementado, esta función crea un socket y lo configura para el envío o la recepción de datos de audio.
        - Process: función principal del elemento de audio, se llama periódicamente a esta función desde la pipeline. Dentro de process se invocan de manera repetitiva:
            - Read: si un elemento de la cadena no tiene un *callback* específico registrado, obtiene los datos del elemento anterior. En el caso del lector UDP, el *callback* implementado se encarga de recibir el audio enviado por el cliente, interpretando el formato descrito para el audio en el capítulo 4.

            - Write: envía los datos al siguiente elemento de la cadena cuando no tiene un *callback* específico registrado. En el caso específico del escritor UDP, el callback implementado se encarga de empaquetar el audio conforme al formato descrito en el capítulo 4 y se transmite al cliente.
        - Close: se ejecuta cuando el pipeline se detiene, cierra el socket creado.
        - Destroy: libera recursos de forma definitiva cuando el elemento se elimina.

6.5 *Licencias del software utilizado*

- ESP-IDF: Apache License 2.0
- ESP-ADF: ESPRESSIF MIT License
- FreeRTOS: MIT Open Source License
- espressif/esp32-camera: Apache License 2.0
- esp-protocols/components/mdns: Apache License 2.0

6.6 *Repositorio*

- https://github.com/QnkPrg/telRem

Capítulo 7 - Hardware utilizado

Para el desarrollo de este proyecto se evaluaron diferentes plataformas de hardware. En un principio, debido a su precio, consideramos dos dispositivos, una combinación de la ESP32 Audio Kit de Ai-Thinker, y la ESP32-CAM. Sin embargo, nos encontramos con varios problemas de compatibilidad entre el framework de ESP-ADF y la ESP32 Audio Kit.

Por estos motivos decidimos cambiar a la ESP32-S3-KORVO-2, con la que podríamos desarrollar una solución con una sola placa, además de ofrecer esta una documentación mucho más completa, facilitando el desarrollo.

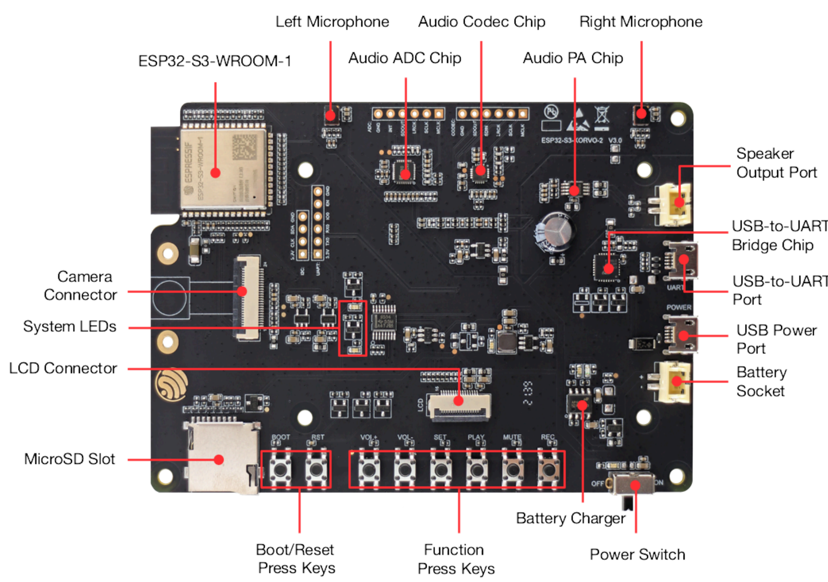


Figura 7-1: ESP32-S3-Korvo-2 V3.1.

Fuente: Espressif [(enlace)]

7.1 Características principales de el SOC ESP32-S3-WROOM-1

- Microprocesador Xtensa dual-core 32-bit LX7r, de hasta 240 MHz
- 512 KB SRAM
- 8 MB PSRAM octal con velocidad de hasta 80 Mhz
- 16 MB de memoria Quad SPI Flash.
- WiFi 802.11b/g/n con velocidad de hasta 150Mbps

7.2 Audio Codec ES8311

- ADC
    - 24-bit, frecuencias de muestreo desde 8 hasta 96 kHz
- DAC
    - 24-bit, frecuencias de muestreo desde 8 hasta 96 kHz

7.3 Cámara OV2640

- Compatible con compresión JPEG hardware.
- Máximos fotogramas por segundo dependiendo del formato:
    - UXGA/SXGA: 15 fps
    - SVGA: 30 fps
    - CIF: 60 fps

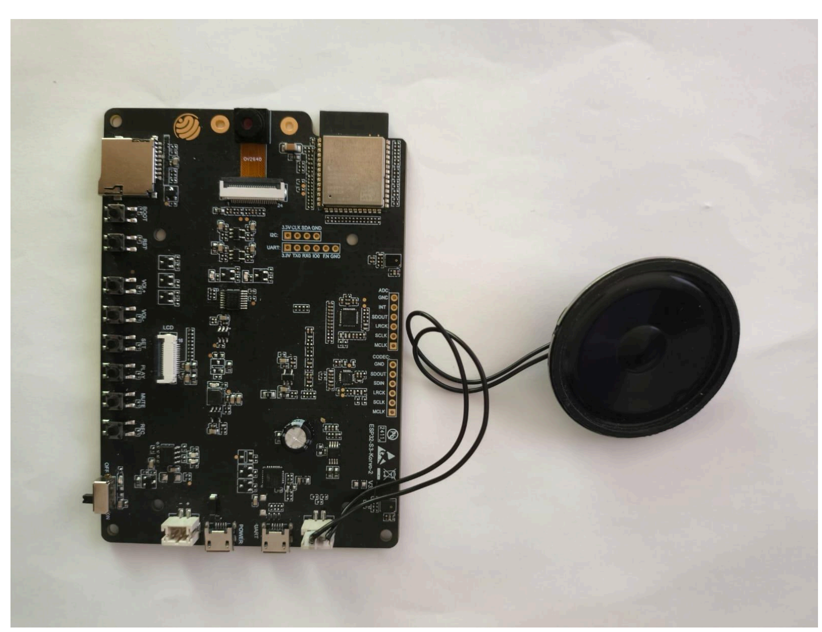


Figura 7-2: ESP32-S3-Korvo-2 con altavoz y OV2640 conectados.

Capítulo 8 - Aplicación móvil

La aplicación móvil desarrollada en Flutter actúa como interfaz principal para el usuario, permitiendo una interacción sencilla y eficiente con el sistema de telefonillo implementado con la ESP32. Su diseño modular y multiplataforma facilita la integración de nuevas funcionalidades y la adaptación a distintos dispositivos Android.

8.1 Entorno de Desarrollo

El desarrollo de la aplicación móvil se realizó utilizando Flutter SDK, un entorno multiplataforma que permite crear aplicaciones nativas para Android, iOS, Windows, Linux y web a partir de un único código fuente en Dart. Para la gestión del proyecto se empleó Visual Studio Code, aprovechando su integración con Flutter más la fácil organización que proporciona para la hora de navegar a través de archivos.

El uso de Flutter facilitó la creación de una interfaz moderna y atractiva, permitiendo poder probar distintos diseños y la funcionalidad de la aplicación de forma rápida.

Además, para la parte de comunicación con el hardware y la simulación de servicios en el *backend*, se desarrollaron y emplearon varios *scripts* en Python. Estos *scripts* permitieron:

- Simular el comportamiento del ESP32 durante el desarrollo y las pruebas.
- Implementar servidores TCP/UDP para pruebas de transmisión de audio y vídeo.

Flutter permite la directa instalación de APK(*Android Package Kit* es el formato que permite distribuir e instalar aplicaciones en Android) a través de un cable usb conectado de ordenador a móvil. Lo único fue que esa opción requería una serie de opciones de administrador en el móvil, cosa que generaba fallos, por lo que se optó por la generación de APK desde flutter a un archivo /app/outputs/flutter-apk que permite el SDK. De esa forma a través de Drive se pudieron probar las apks generadas.

8.2 Arquitectura y flujo de la aplicación

La app está estructurada en varias pantallas principales:

- **Descubrimiento de dispositivos**: utiliza mDNS para detectar automáticamente el ESP32 en la red local, simplificando la conexión inicial.
  - Módulo: device_selector.dart
- **Menú principal**: permite al usuario iniciar una llamada, abrir la puerta o acceder al aprovisionamiento WiFi.
  - Módulo: menu_screen.dart
- **Pantalla de llamada**: gestiona la transmisión bidireccional de audio y vídeo en tiempo real, mostrando el vídeo recibido y permitiendo hablar o escuchar.
  - Módulo: vídeo_audio.dart
    - Submódulos: audio_sender.dart , vídeo_receiver.dart
- **Aprovisionamiento WiFi**: facilita la configuración de la red WiFi del ESP32, tanto en modo normal como en modo AP (punto de acceso).
  - Módulo: wifi_provisioning_screen.dart

- **Diagrama de flujo**

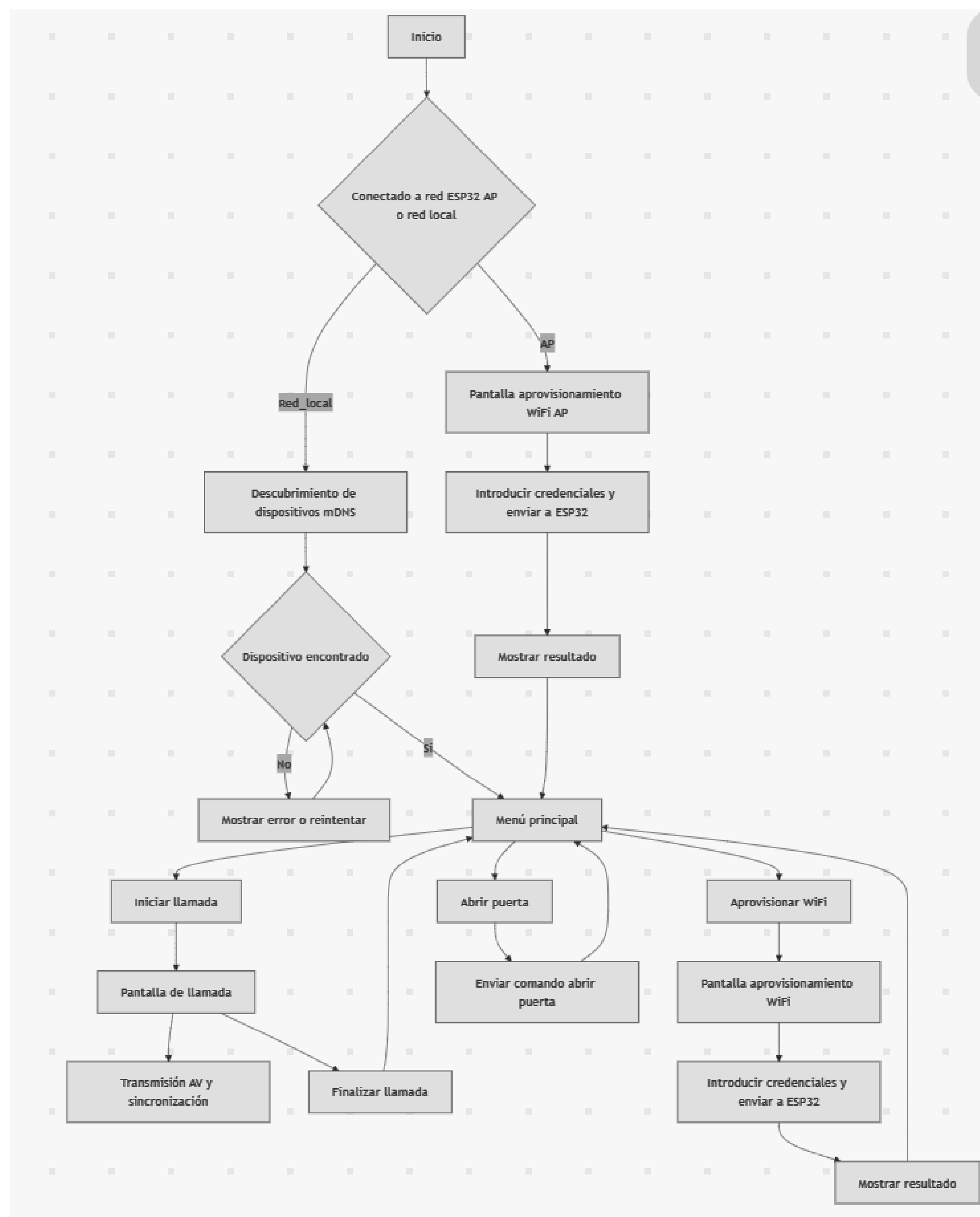


Figura 8-1: Diagrama de flujo de la aplicación móvil

La aplicación cuenta con otros dos módulos:

- **Sincronización AV**: define variables globales para la sincronización entre audio y vídeo, como el timestamp del último paquete de audio recibido, que es utilizado por el módulo de vídeo para mostrar los frames sincronizados.
    - Módulo: **sync_state.dart**
- **Entrada a la app**: aquí se define el punto de entrada a la app, donde se establece la pantalla de *buscar ESP32*.
    - Módulo: **main.dart**

Al principio se ideaba tener todo en una misma pantalla, pensado más para probar cada funcionalidad sin necesidad de cambiar entre pantallas, pero de cara al usuario se decidió usar distintas pantallas, mostrando menos información en cada pantalla de forma que sea más intuitivo.

También de cara al código lo hemos estructurado de una forma modular, agilizando la lectura y el cambio.

8.3 Lógica de comunicación

La aplicación se encarga de gestionar internamente todos los detalles técnicos del protocolo de comunicación, de modo que el usuario no tiene que preocuparse por ellos. El usuario solo interactúa con una interfaz sencilla (botones y mensajes claros), mientras que la app realiza automáticamente todas las operaciones necesarias para establecer la conexión, enviar y recibir datos, y mantener la sincronización entre audio y vídeo. Así, se ocultan las complejidades técnicas y se ofrece una experiencia de uso simple y directa.

La app gestiona dos canales principales:

- **Canal de control (TCP)**: se utiliza para enviar comandos críticos (iniciar/finalizar llamada, abrir puerta) y recibir respuestas o notificaciones del ESP32. La app implementa una lógica de reintentos y manejo de errores para asegurar la fiabilidad de estas operaciones. Por ejemplo, si no se recibe confirmación de un comando, se notifica al usuario y se permite reintentar la acción.
    - **Establecimiento de la conexión**:
        - La app intenta conectarse por TCP al ESP32 justo después de descubrir su IP mediante mDNS (cuando la placa ya está conectada a la red local).
        - La conexión se realiza al puerto 12345 del ESP32.
        - Si la conexión falla, se informa al usuario y se permite reintentar.
    - **Comandos enviados por TCP**:
        - Una vez establecida la conexión, la app puede enviar distintos comandos críticos, cada uno representado por un número entero:
            - 0: Solicitar inicio de llamada (CMD_REQUEST_TALK).
            - 1: Finalizar llamada (CMD_END_TALK).
            - 7: Abrir puerta (CMD_OPEN_DOOR).
    - Estos comandos se envían en el momento en que el usuario pulsa los botones correspondientes en la interfaz.
    - La app espera respuestas de la placa para confirmar la acción (por ejemplo, aceptación o denegación de la llamada).

- **Canal de datos (UDP)**: se encarga de la transmisión de audio y vídeo en tiempo real. La app implementa un buffer circular para los paquetes de audio y vídeo, permitiendo la reconstrucción de los datos incluso si llegan fuera de orden o con pequeñas pérdidas.
    - **Inicio de transmisión**:
        - Tras establecer la conexión TCP y recibir la confirmación para iniciar la llamada, la app comienza a capturar audio del micrófono usando la librería flutter_sound.
        - El audio se captura en bloques (chunks) definidos por el parámetro bufferSize (por ejemplo, 324 bytes), lo que permite un equilibrio entre latencia y eficiencia de red.
    - **Empaquetado y envío**:
        - Cada bloque de audio se empaqueta junto con una cabecera personalizada.
        - El paquete se envía por UDP al ESP32, usando la IP y puerto configurados.
        - El número de secuencia se incrementa con cada paquete, permitiendo al receptor detectar pérdidas o desorden.

- **Recepción de audio y gestión de buffer**:
  - La app recibe paquetes de audio por UDP y extrae los datos de audio de la cabecera.
  - Se utiliza un buffer FIFO tradicional, donde se añaden los paquetes recibidos.
  - En cada ciclo de reproducción, se procesan y reproducen todos los paquetes en orden de llegada, asegurando que no se pierda información y manteniendo la calidad del audio.
  - Esto puede suponer un aumento ligero de latencia debido a que se quiere reproducir cada paquete recibido.
- **Recepción del vídeo y reconstrucción**:
  - El vídeo se recibe por UDP en fragmentos, ya que un fotograma puede ocupar varios paquetes debido al tamaño de MTU.
  - La app almacena los fragmentos en un mapa temporal hasta tener todos los necesarios para reconstruir el fotograma.
  - Una vez reconstruido, se compara la marca de tiempo del frame con el del último audio recibido (lastAudioTimestamp), y solo se muestra el frame si está suficientemente sincronizado (dentro de un umbral de tiempo).
  - Si el frame llega muy desincronizado respecto al audio, se descarta para evitar mostrar vídeo fuera de tiempo.

Librerías usadas:

- **Canal de control (TCP)**:
  - Librería utilizada: **dart:io**
  - Motivo: Permite la gestión directa de sockets TCP, necesaria para enviar comandos críticos (iniciar/finalizar llamada, abrir puerta, activar/desactivar FEC, etc.) y recibir respuestas o notificaciones del ESP32. El uso de dart:io proporciona control total sobre la conexión, la gestión de errores y la implementación de lógica de reintentos, asegurando la fiabilidad de las operaciones.

- **Canal de datos (UDP)**:
  - Librería utilizada: **dart:io**
  - Motivo: Permite la transmisión eficiente de audio y vídeo en tiempo real mediante sockets UDP. Se utiliza para enviar y recibir paquetes de datos con baja latencia. La app implementa buffers circulares para los paquetes de audio y vídeo, permitiendo la reconstrucción de los datos incluso si llegan fuera de orden o con pequeñas pérdidas.
- **Audio (captura y reproducción)**:
  - Librería utilizada: **flutter_sound**
  - Motivo: Facilita la grabación y reproducción de audio en tiempo real, compatible con múltiples plataformas. Permite trabajar con flujos de audio PCM, necesarios para la transmisión directa por UDP.
- **Permisos de micrófono**:

- Librería utilizada: **permission_handler**
- Motivo: Gestiona de forma sencilla los permisos necesarios para acceder al micrófono del dispositivo, mostrando diálogos nativos y comprobando el estado de los permisos.

8.4 Lógica del descubrimiento de vecinos por mDNS

Para que la aplicación detecte automáticamente el dispositivo ESP32 en la red local, se utiliza el protocolo mDNS (Multicast DNS). Este protocolo permite descubrir servicios y dispositivos en la red sin necesidad de conocer su dirección IP previamente.

- **¿Cómo funciona mDNS en la app?**
  - El ESP32 anuncia su presencia en la red local bajo el nombre telrem.local usando mDNS.
  - La app Flutter, mediante la librería multicast_dns, realiza una consulta mDNS buscando la dirección IPv4 asociada a telrem.local.
  - Si la consulta tiene éxito, la app obtiene la IP local del dispositivo y puede iniciar la conexión TCP/UDP para la comunicación AV o el control de la puerta.
- **Ventajas del uso de mDNS**
  - Zero-configuration: No es necesario que el usuario introduzca manualmente la IP del dispositivo.
  - Automatización: El proceso es transparente y automático para el usuario.

- Compatibilidad: mDNS es ampliamente soportado en redes domésticas y dispositivos IoT.

- **Limitaciones y consideraciones**
  - mDNS solo funciona cuando el móvil y el ESP32 están conectados a la misma red local (no en modo AP).
  - En modo AP, la app debe recurrir a la IP fija (192.168.4.1) para el aprovisionamiento, ya que mDNS no está disponible.
  - Si el dispositivo no se encuentra, la app ofrece opciones de reintentar la búsqueda o realizar el aprovisionamiento WiFi.

8.5 Experiencia de usuario y sincronización Audio-vídeo

La experiencia de usuario ha sido una prioridad en el diseño de la app. Se han implementado varias estrategias para garantizar una interacción intuitiva y una comunicación audiovisual fluida:

- **Sincronización AV**: para evitar desincronización entre audio y vídeo, la app utiliza marcas de tiempo incluidas en los paquetes. El audio recibido se almacena en un buffer y se reproduce con un pequeño retardo controlado por temporizador, permitiendo que el vídeo se sincronice con el audio real. El vídeo solo se muestra cuando su marca de tiempo coincide (o está muy próximo) al del audio en reproducción, evitando así el efecto de "labios desincronizados".

- **Gestión de buffers**: se ha ajustado el tamaño de los buffers y el intervalo de reproducción para minimizar la latencia sin sacrificar la estabilidad. Si la red empeora y se detectan saltos o pérdidas, la app puede aumentar temporalmente el retardo para evitar cortes perceptibles.
- **Mensajes e instrucciones claras**: en cada pantalla, la app informa al usuario del estado de la conexión, los pasos a seguir (por ejemplo, durante el aprovisionamiento WiFi en modo AP) y los posibles errores, facilitando la resolución de problemas sin conocimientos técnicos.

8.6 Pruebas

Se realizaron dos tipos de pruebas durante el desarrollo de la app:

- **Pruebas funcionales**: al querer implementar tantas funcionalidades se tuvieron que crear varios scripts en python simulando cada funcionalidad de la ESP32 por separado e ir probando distintos apks que se subieron al drive.

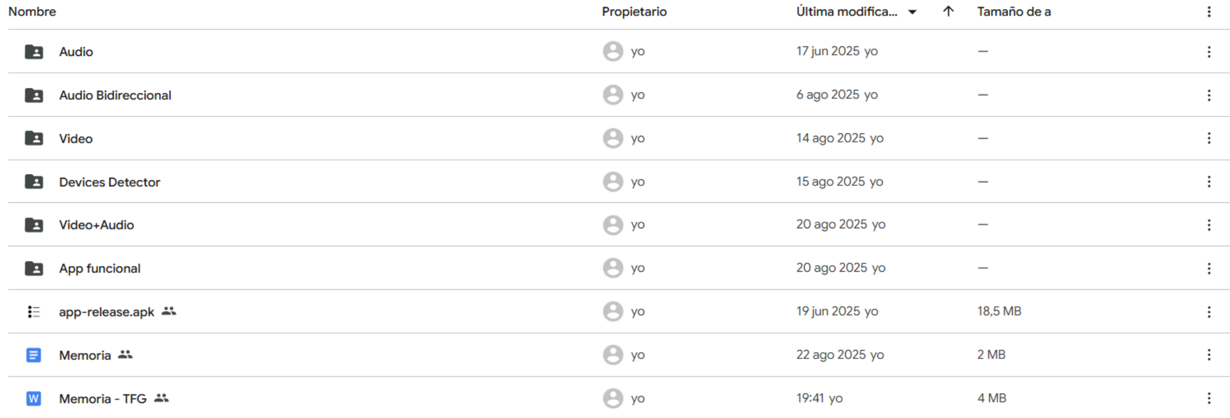


Figura 8-2: Carpeta del drive con los distintos apks generados

- **Audio**: primera funcionalidad implementada. Lo primero que se quiso realizar fue la detección del micrófono del móvil y también manejar los permisos necesarios para el audio en Android.
  - Se usó el script *recibir_audio.py*, con la biblioteca *audiopy*.

- **Audio bidireccional:** a partir del envío de audio se quiso también implementar la recepción. A su vez se implementó también el envío de comandos por TCP. Se tardó más de lo esperado debido a que el móvil que se usaba en ese momento para las pruebas no reproducía de forma correcta el audio.
  - Se usaron los scripts *audio_bidireccional.py*, también con *audio.py*, y *servidorTCP.py* que solo necesitó el uso de sockets.

- **Vídeo**: la implementación del vídeo fue relativamente rápida al solo tener que recibir y reconstruir los frames.

        - En el script *enviar_audio.py*, gracias a la biblioteca *cv2* se codificaron los frames a JPEG pudiendo así enviarlos a la app.

    - **Multicast DNS**: desde la última implementación se empezaron a hacer pruebas con la ESP32 y se optó por añadir una funcionalidad que no requiriera introducir cada vez de forma manual la IP de ambos móvil y placa.
        - Gracias a *mDNS.py* se pudo comprobar el comportamiento y gracias a *mdns_telrem_sim.py* se pudo simular el descubrimiento de la ESP32. Se usó la biblioteca *zeroconf*.

    - **vídeo+Audio**: al tener ambas funciones hechas por separado, solo se tuvieron que juntar y realizar la sincronización AV. Para la sincronización se pensaron varias ideas, de las cuales se optó por acabar realizando la comparación de timestamps, descartando aquellos que tengan un margen muy alto.
        - Al principio se usó *enviar_vídeoAudio.py* para realizar pruebas pero debido al problema con el móvil mencionado previamente se optó por hacer pruebas con la ESP32 directamente.

- **App funcional**: una vez realizado y probado lo anterior se empezaron a juntar todas las funcionalidades en un mismo apk, comenzado a pensar en distintos flujos posibles. A la vez se añadió la funcionalidad de wifi provisioning, que fue simple al estar ya implementado previamente en la ESP32.
    - Se realizaron muchas pruebas y depuraciones de los distintos flujos pensados. Debido a juntar las distintas funcionalidades, algunos empezaron a fallar.

- **Pruebas de interfaz**: gracias al comando *flutter run -d chrome* se pudo simular la interfaz sin necesidad de un móvil, agilizando así la depuración y permitiendo probar distintos formatos.
  - Al principio de la integración de la app no se tocó mucho la interfaz aparte de mostrar más botones, debido a que se quiso primero priorizar la correcta funcionalidad de la app antes que lo visual.

8.7 Repositorio y licencias

- **Repositorio:** https://github.com/gecoS/telefonillo_app

- **Flutter**
  - Licencia: BSD 3-Clause
  - https://github.com/flutter/flutter/blob/main/LICENSE
- **flutter_sound**
  - Licencia: MIT
  - https://pub.dev/packages/flutter_sound/license
- **just_audio**
  - Licencia: BSD 2-Clause
  - https://pub.dev/packages/just_audio/license
- **permission_handler**
  - Licencia: MIT
  - https://pub.dev/packages/permission_handler/license
- **multicast_dns**
  - Licencia: BSD 3-Clause
  - https://pub.dev/packages/multicast_dns/license
- **http**
  - Licencia: BSD 3-Clause
  - https://pub.dev/packages/http/license

8.8 Interfaces

Las capturas de pantalla fueron sacadas tanto de la versión web como del móvil para mostrar ambas formas que tuvimos de depurar.

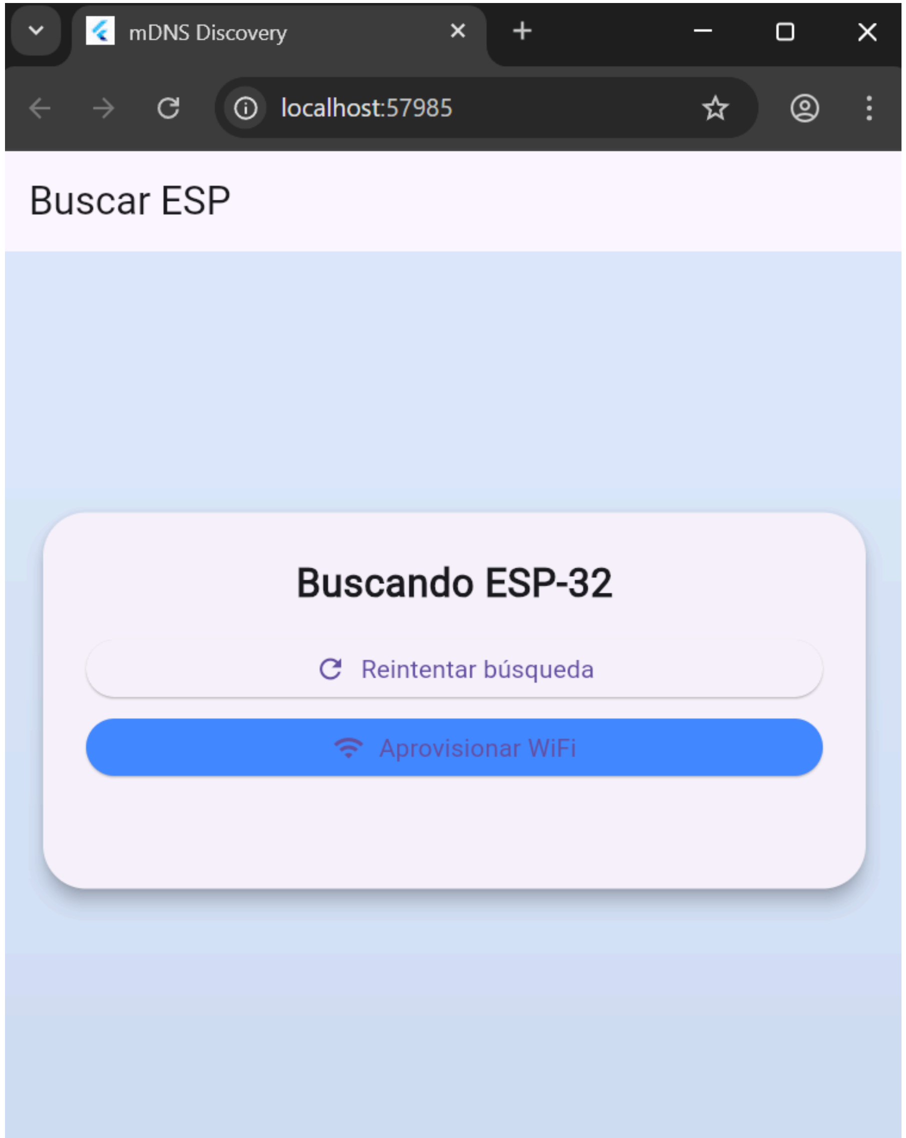


Figura 8-3: Búsqueda de la ESP-32

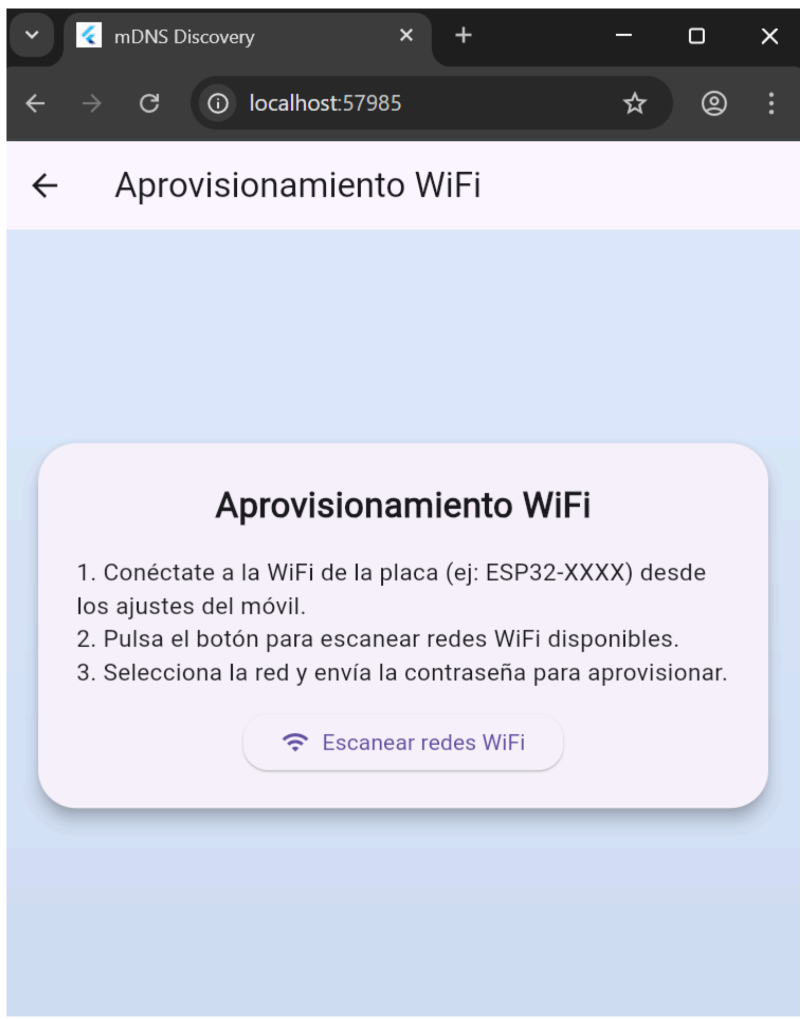


Figura 8-4: Menú del aprovisionamiento

Figura 8-5: Redes escaneadas (móvil)

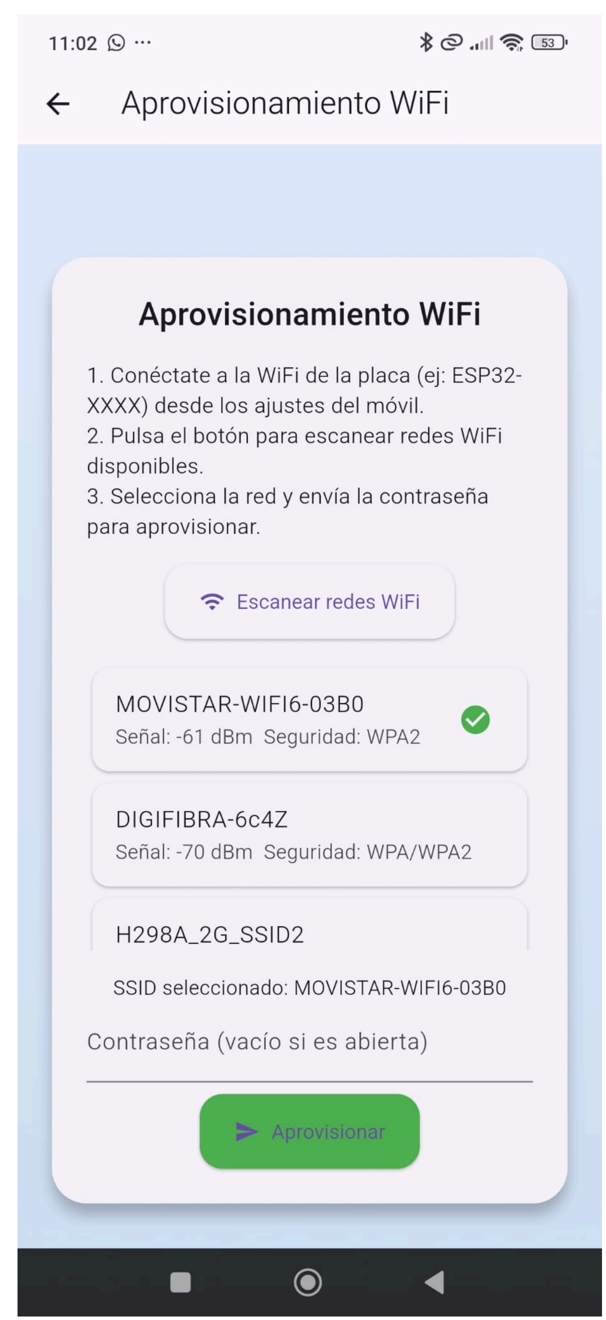


Figura 8-6: Red seleccionada (móvil)

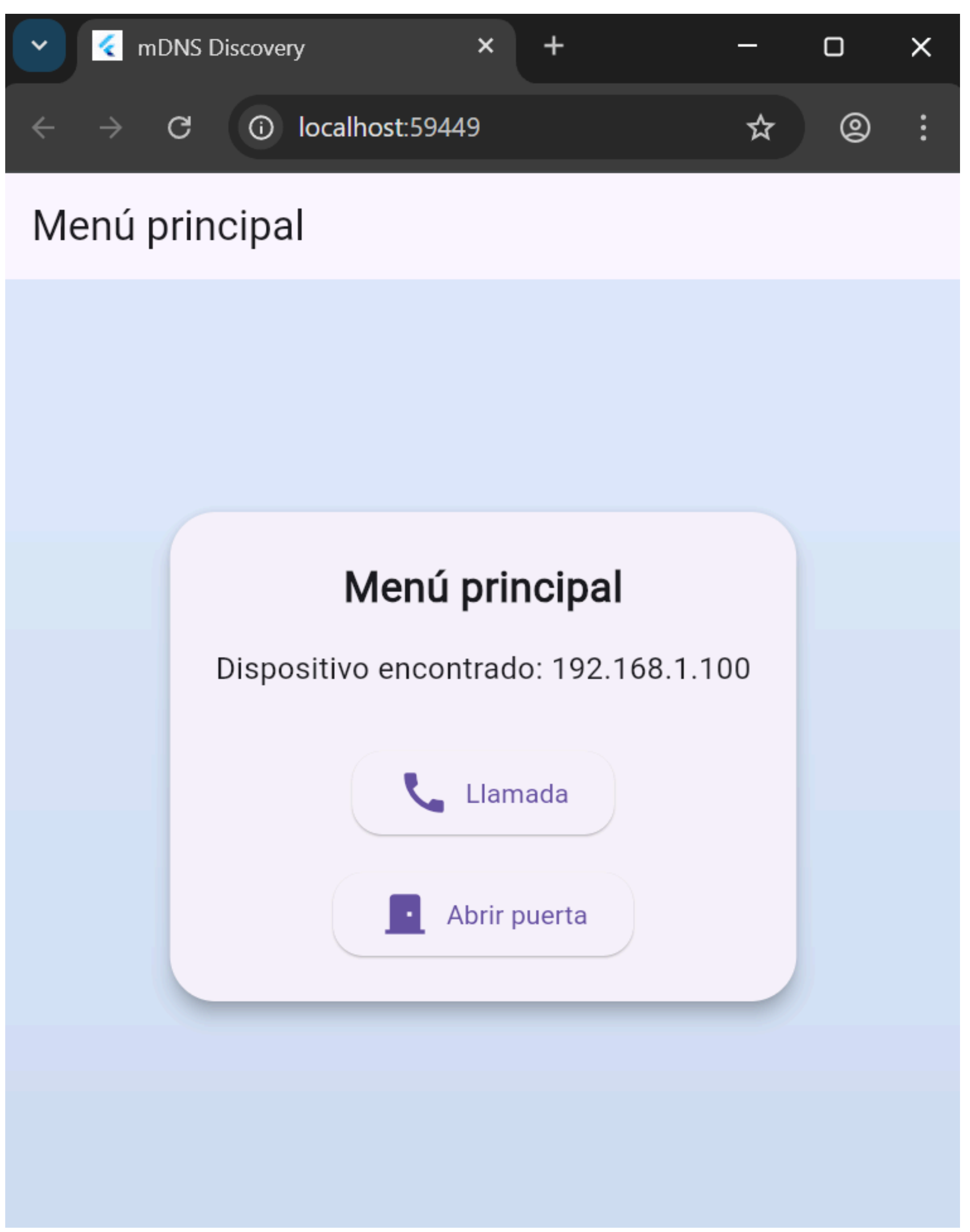


Figura 8-7: Menú principal

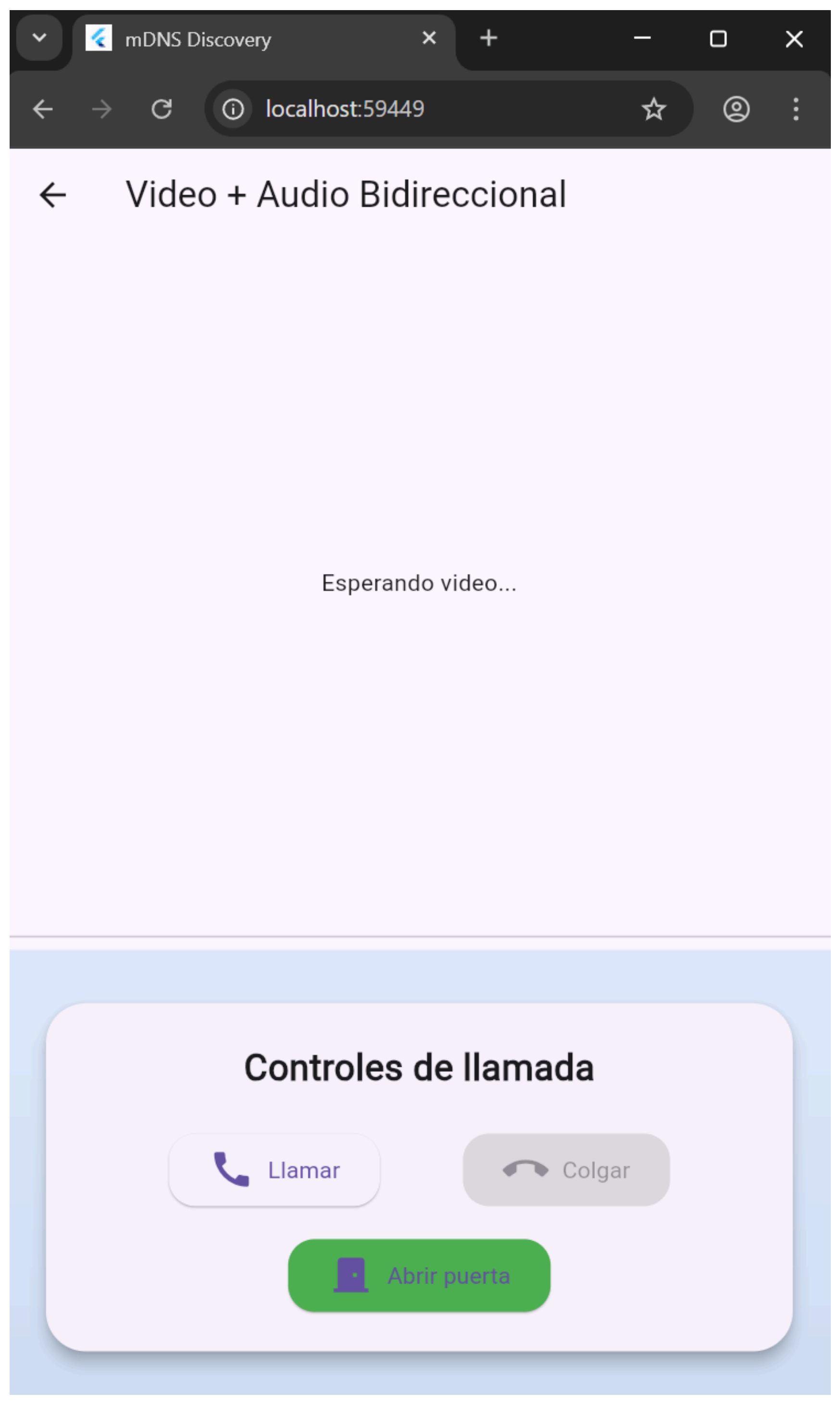


Figura 8-8: Interfaz de la llamada

Capítulo 9 - Conclusiones y trabajo futuro

9.1 Conclusión

A lo largo de este Trabajo de Fin de Grado se ha logrado diseñar e implementar un sistema de telefonillo inteligente que integra hardware ESP32 y una aplicación móvil multiplataforma desarrollada con Flutter. El proyecto ha permitido abordar retos técnicos como la comunicación audiovisual bidireccional, el descubrimiento automático de dispositivos mediante mDNS, el aprovisionamiento WiFi y el control remoto de acceso, todo ello con una interfaz moderna y accesible.

El trabajo realizado destaca por su flexibilidad y carácter abierto, facilitando la personalización y la integración en entornos domóticos actuales. Además, el trabajo realizado nos ha supuesto una oportunidad para aplicar conocimientos adquiridos durante la carrera y coordinarnos como equipo, enfrentando con éxito los desafíos del desarrollo de sistemas embebidos y aplicaciones móviles.

En conclusión, el resultado obtenido responde a los objetivos planteados y sienta las bases para futuras mejoras y ampliaciones, contribuyendo al avance de soluciones inteligentes y accesibles para el hogar.

9.2 Trabajo futuro

A pesar de los resultados alcanzados, el presente trabajo deja abiertas diversas posibilidades de mejora y ampliación. En esta sección se presentan las principales líneas de trabajo futuro que podrían contribuir a optimizar el sistema y explorar nuevas funcionalidades.

Una posible línea de trabajo futuro sería la incorporación del protocolo RTP (Real-time Transport Protocol) para la transmisión de audio y vídeo. Esto requeriría implementar el soporte para este protocolo en el firmware, teniendo en cuenta el formato, los mecanismos de control mediante RTCP y la corrección de errores hacia adelante entre otras cosas. En un entorno de red local, la comunicación podría realizarse directamente entre la placa y el dispositivo móvil con el protocolo que hemos usado hasta ahora. No obstante, para escenarios de transmisión a través de Internet o con múltiples receptores, RTP sería necesario ya que es capaz de abordar las distintas complejidades introducidas al no estar el dispositivo y los clientes en la misma red, haciendo también necesaria la introducción de servidores intermedios que faciliten la gestión de NAT y la escalabilidad del proyecto.

También encontramos otra línea de trabajo bastante clara, la seguridad. Hasta ahora, al trabajar en una red local, la seguridad no ha sido el principal foco del desarrollo de este proyecto, haciendo que se pueda mejorar en diversos aspectos. Entre las posibles mejoras, podemos incluir la autenticación de los clientes al conectarse a la placa, el cifrado de las comunicaciones mediante protocolos como HTTPS o SRTP, y la protección de los datos sensibles almacenados en la placa como credenciales wifi u otros similares.

Otra línea de trabajo futuro sería la conexión del sistema a un telefonillo real, ya que actualmente estamos simulando uno. Para esto necesitaríamos diseñar una placa personalizada con las especificaciones necesarias para poder garantizar la comunicación con un telefonillo real. Entre las diversas especificaciones que serían necesarias, las principales serían la posibilidad de conectar al codec de audio al

telefonillo directamente, y el acceso a un ADC para poder detectar cuándo se está llamando al telefonillo, ya que un GPIO convencional no sería suficiente debido a que los voltajes del telefonillo superan los 3,3 V. Obviamente, estos cambios en la plataforma hardware también conllevarían cambios significativos en el firmware, siendo necesaria la implementación de los HAL de los componentes añadidos.

Capítulo 10 - Conclusions and future work

10.1 Conclusion

Throughout this Final Degree Project, it has been possible to design and implement a smart intercom system that integrates ESP32 hardware and a cross-platform mobile application developed with Flutter. The project has made it possible to address technical challenges such as bidirectional audiovisual communication, automatic device discovery via mDNS, WiFi provisioning, and remote access control, all with a modern and user-friendly interface.

The work carried out stands out for its flexibility and open nature, facilitating customization and integration into current smart home environments. Furthermore, the project has provided us with an opportunity to apply the knowledge acquired during our studies and to coordinate as a team, successfully facing the challenges of embedded systems development and mobile applications.

In conclusion, the results achieved meet the stated objectives and lay the groundwork for future improvements and extensions, contributing to the advancement of intelligent and accessible solutions for the home.

10.2 Future work

Despite the results achieved, this work leaves open several possibilities for improvement and expansion. In this subchapter, the main lines of future work that could contribute to optimizing the system and exploring new functionalities are presented.

One possible line of future work would be the incorporation of RTP (Real-time Transport Protocol) for audio and vídeo transmission. This would require implementing support for this protocol in the firmware, taking into account the format, control mechanisms via RTCP, forward error correction, among other things. In a local network environment, communication could be carried out directly between the board and the mobile device using the protocol we have used so far. However, for scenarios involving transmission over the Internet or with multiple receivers, RTP would be necessary, as it is capable of addressing the various complexities introduced when the device and clients are not on the same network, also making the introduction of intermediate servers necessary to facilitate NAT management and project scalability.

Another quite clear line of work is security. Until now, working on a local network, security has not been the main focus of this project's development, leaving room for improvements in various aspects. Possible improvements could include client authentication when connecting to the board, communication encryption through protocols such as HTTPS or SRTP, and the protection of sensitive data stored on the board, such as Wi-Fi credentials or other similar information.

Another line of future work would be connecting the system to a real intercom, as we are currently simulating one. For this, we would need to design a custom board with the necessary specifications to guarantee communication with a real intercom. Among the various required specifications, the main ones would be the ability to connect the intercom directly to the audio codec, and access to an ADC to detect when the intercom is being called, since a conventional GPIO would not suffice due to the intercom's voltages exceeding 3.3 V. Obviously, these changes to the hardware platform would also entail significant changes in the firmware, requiring the implementation of the HALs for the added components.

Contribuciones Personales

Dorzhi Aylagas Tsyzhipov

- **Compra**
    - Yo me he encargado de la compra de las ESP32-CAM más la compra de los puertos necesarios para conectar el altavoz proporcionado por nuestros tutores a la placa. La ESP32-CAM por internet y los puertos en Electrónica Embajadores.

- **Investigación**
    - Al principio, al igual que mi compañero Sergio, estuve investigando y haciéndome al entorno de ESP-IDF, al igual que pensando sobre posibles formas de implementar las funciones.
    - Después de la compra de la ESP32-korvo, debido a lo obsoletas que estaban las primeras placas ESP32 adquiridas, como solo se compró una, me dediqué a investigar sobre la app.
    - Esta investigación sobre la app implicó encontrar un entorno adecuado que permitiera las distintas funcionalidades pensadas, compatibilidad con ambos iOS y Android.

- **Bocetos**
    - He diseñado en canva futuras posibilidades de interfaces de la app, pensado puramente en lo visual.

- **Diseño y desarrollo**
    - He diseñado y desarrollado toda la parte de la aplicación móvil, implementando tanto las interfaces de usuario como la lógica detrás de la interacción con la placa ESP32.

- **Pruebas**
    - De las pruebas funcionales y de usabilidad de la app me encargué yo, corrigiendo errores y mejorando la experiencia de usuario de forma iterativa.

- **Memoria**
    - Me he encargado del estructurado de la memoria en función a nuestro plan de trabajo.
    - Toda la parte de la aplicación móvil ha sido redactada por mí. Aparte, la introducción, portada, estado de la cuestión, conclusión y la traducción al inglés también.

Sergio Cuenca López

- **Investigación**
    - Me encargué de investigar las distintas opciones a la hora de escoger la plataforma hardware sobre la que desarrollar el proyecto. (Excluyendo el altavoz, que fue proporcionado por nuestros tutores).
    - Asumí la responsabilidad de investigar sobre los distintos protocolos existentes que podrían cumplir las necesidades del proyecto.
    - Para el desarrollo del firmware, tuve que investigar sobre el funcionamiento de CMake, tanto para añadir componentes personalizados al framework de ESP-ADF como para poder compilar el firmware.
    - Durante el desarrollo del firmware busqué formas de disminuir la latencia entre la captura del audio y el envío de este, reduciendo el tamaño del buffer del elemento lector I2S.
    - Para el desarrollo del elemento de audio personalizado de UDP tuve que investigar el funcionamiento de otros elementos de audios desarrollados en el framework de ESP-ADF.
    - También investigué sobre cómo protocolos robustos como RTP tratan la pérdida de paquetes.

- **Diseño y desarrollo**
  - Me encargué de la definición del protocolo usado para el transporte de audio y vídeo, incluyendo el formato de los mensajes usados.

  - Propuse y definí el uso del servidor HTTP y el formato de los mensajes JSON empleados en el aprovisionamiento de WiFi.

  - Me responsabilicé del diseño y desarrollo completo del firmware, garantizando un código limpio, mantenible y modular, con funciones claramente delimitadas en cada módulo.

  - Desarrollé los scripts en Python descritos en el capítulo 6, los cuales resultaron útiles tanto para verificar la funcionalidad como para obtener distintas métricas de rendimiento. Dichas métricas permitieron tomar la decisión de eliminar los mensajes FEC, dado que la pérdida de paquetes en las condiciones de prueba era mínima, eliminando gran parte de la complejidad de la lógica de recepción de paquetes.

- **Pruebas**
  - Todas las pruebas descritas en el capítulo 6 fueron ideadas y ejecutadas por mí.

- **Memoria**
  - En cuanto a la memoria, me encargué de la redacción de los capítulos 4, 6, 7, y la parte de trabajo futuro de los capítulos 9 y 10.

Personal Contributions

Dorzhi Aylagas Tsyzhipov

- **Purchases**
  - I was responsible for purchasing the ESP32-CAM boards, as well as the connectors required to link the speaker provided by our tutors to the board. The ESP32-CAM was bought online, and the connectors at Electrónica Embajadores.

- **Research**
  - At the beginning, just like my teammate Sergio, I researched and familiarized myself with the ESP-IDF environment, while also considering possible ways to implement the required functions.
  - After purchasing the ESP32-Korvo, since the first ESP32 boards we acquired had become obsolete and only one new board was bought, I focused my research on the app.
  - This research on the app involved finding a suitable framework that would support the intended functionalities while ensuring compatibility with both iOS and Android.

- **Sketches**
  - I designed possible future app interfaces using Canva, focusing purely on the visual aspect.

- **Design and Development**
    - I designed and developed the entire mobile application, implementing both the user interfaces and the logic behind the interaction with the ESP32 board.

- **Testing**
    - I was in charge of the functional and usability testing of the app, fixing errors and iteratively improving the user experience.

- **Report**
    - I organized the structure of the thesis according to our work plan.
    - I wrote the entire section on the mobile application, as well as the introduction, cover page, state of the art, conclusions, and the English translation.

Sergio Cuenca López

- **Research**
  - I researched the different options when choosing the hardware platform on which to develop the project (excluding the speaker, which was provided by our tutors).
  - I was responsible for researching the different existing protocols that could meet the needs of the project.
  - For the firmware development, I researched how CMake works, both to add custom components to the ESP-ADF framework and to compile the firmware.
  - During firmware development, I looked for ways to reduce latency between audio capture and transmission by decreasing the buffer size of the I2S reader element.
  - For the development of the custom UDP audio element, I studied how other audio elements implemented in the ESP-ADF framework work.
  - I also researched how robust protocols such as RTP handle packet loss.

- **Design and Development**
  - I defined the protocol used for audio and video transport, including the message formats.
  - I proposed and defined the use of the HTTP server and the JSON message format used in WiFi provisioning.

    - I was responsible for the complete design and development of the firmware, ensuring clean, maintainable, and modular code with clearly defined functions in each module.
    - I developed the Python scripts described in Chapter 6, which were useful both to verify functionality and to obtain different performance metrics. These metrics led us to decide to remove FEC messages, since packet loss during testing was minimal, thus eliminating much of the complexity in the packet reception logic.

- **Testing**
    - All the tests described in Chapter 6 were conceived and carried out by me.

- **Report**
    - Regarding the report, I was responsible for writing Chapters 4, 6, and 7, as well as the future work section in Chapters 9 and 10.

Bibliografía

[1] Flutter. “Build apps for any screen.” https://flutter.dev/

[2] FreeRTOS. “FreeRTOS documentation.” https://www.freertos.org/Documentation/00-Overview

[3] Espressif Systems. “ESP-IDF Programming Guide.” https://docs.espressif.com/projects/esp-idf/en/latest/esp32/

[4] Google Developers. “Multicast DNS (mDNS) and Service Discovery.” https://code.google.com/archive/p/iotsys/wikis/mdnssd.wiki

[5] Flutter Sound. “A complete audio solution for Flutter.” https://pub.dev/packages/flutter_sound

[6] just_audio. “A feature-rich audio player for Flutter.” https://pub.dev/packages/just_audio

[8] Espressif Systems. “ESP-ADF API Reference” https://docs.espressif.com/projects/esp-adf/en/latest/api-reference/index.html

[8] Espresif Systems “ESP32-s3-wroom1 datasheet” https://www.espressif.com/sites/default/files/documentation/esp32-s3-wroom-1_wroom-1u_datasheet_en.pdf

[9] OmniVision “OV2640 data sheet” https://www.uctronics.com/download/cam_module/OV2640DS.pdf?srsltid=AfmBOoq7Z90kpybowHHrufdzgOIF-2QS3Bcb0nbLEH1VhxaBSn98ODIW

[10] Espressif Systems. “ESP32-camera examples.” https://github.com/espressif/esp32-camera

[11] Everest-Semi “ES8311 Datasheet” http://www.everest-semi.com/pdf/ES8311%20PB.pdf

[12] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson. “RTP: A Transport Protocol for Real-Time Applications” https://datatracker.ietf.org/doc/html/rfc3550

[13] Mo Zanaty , Varun Singh , Ali C. Begen , Giridhar Mandyam. “RTP Payload Format for Flexible Forward Error Correction (FEC)” https://datatracker.ietf.org/doc/html/rfc8627

[14] Espressif “esp32-s3-korvo-2-v3.0.png” https://docs.espressif.com/projects/esp-adf/en/latest/_images/esp32-s3-korvo-2-v3.0.png

[15] “Cross-platform Mobile Development.” https://www.jetbrains.com/help/kotlin-multiplatform-dev/cross-platform-mobile-development.html#cross-platform-mobile-development-definition-and-solutions