UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escuela Técnica Superior de Ingeniería Informática
Universidad Politécnica de Valencia

ARIS Radio
Red de Comunicación para Sistemas Autónomos, IoT y Aplicaciones Espaciales

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Lin Lei Zheng

Tutor: Juan Luis Posadas Yagüe

Curso 2025-2026

El presente trabajo consta de: investigación, estandarización, diseño, y un poco de implementación de pruebas y documentación de un módulo de comunicación destinado a misiones espaciales, redes de dispositivo espacial y redes de IoT terrestre.

El proyecto tiene una extensión un poco densa, ya que el proyecto a realizar contiene varios campos y procesos, lo que requiere un enfoque multidisciplinario y un análisis detallado en cada etapa del desarrollo. Sin embargo el trabajo que se presenta en esta memoria se centra principalmente en las primeras pruebas iniciales, el diseño del protocolo de comunicación y el diseño del *hardware*, dejando para futuras versiones la implementación completa del sistema, y la definición detallada de la arquitectura de red.

Resum

En un món cada vegada més interconnectat i dependent de la tecnologia, coordinar sistemes autònoms en missions crítiques s'ha convertit en un repte fonamental. L'automatització de *rovers*, missions espacials, interconnexió de dispositius satèl·lits i la comunicació IoT en entorns sense cobertura han adquirit gran rellevància. Per garantir la seua eficàcia, és essencial comptar amb una infraestructura de comunicacions robusta, àgil i de llarg abast que superi les limitacions de cobertura i funcione en condicions extremes.

El present treball sorgeix del projecte previ **Oroneta**, un sistema d'automatització i gestió d'enjambres de drons per a tasques de rescat. Davant les limitacions en la transmissió de broadcast i la manca d'una arquitectura de xarxes adequada, s'identifica la necessitat de desenvolupar un estàndard de comunicació sense fils de llarg abast, que facilite una gestió de xarxa més dinàmica i un broadcast eficient.

El projecte en què es centra el treball, denominat **ARIS Radio** (*Adaptive Radio Interchange System*) es focalitza en la tecnologia **LoRa**, que utilitza modulació d'espectre estès basada en **CSS** (*Chirp Spread Spectrum*). LoRa maximitza la potència del senyal, oferint comunicacions de llarg abast, alta sensibilitat i baix consum energètic. Aquesta tecnologia és ideal per superar les limitacions d'altres solucions en entorns de cobertura limitada i és perfecta per a aplicacions IoT, xarxes de sensors i sistemes distribuïts en zones remotes.

L'objectiu d'aquest treball és dissenyar un protocol de comunicació i desenvolupar un modul hardware, que inclou tant el codi com el disseny electrònic, seguint els estàndards establits i les directrius de l'arquitectura de xarxa. Aquest dispositiu serà compatible amb altres *hardware* integrats dins de rovers, satèl·lits, drons, sistemes IoT (Arduino) o qualsevol dispositiu hardware, i oferirà compatibilitat amb dispositius personals com PC, mòbils o sistemes embeguts, a través d'un driver que simplifique la seua implementació i ús.

Paraules clau: Protocols de comunicació, satèl·lits, dispositius hardware, comunicació sense fils, modulació d'espectre, LoRa, drivers, arquitectura de xarxa, Arduino IoT, enjambre de drons, infraestructura, disseny electrònic

Resumen

En un mundo cada vez más interconectado y dependiente de la tecnología, coordinar sistemas autónomos en misiones críticas se ha convertido en un desafío fundamental. La automatización de **rovers**, misiones espaciales, interconexión de dispositivos satelitales y la comunicación IoT en entornos sin cobertura han adquirido gran relevancia. Para garantizar su eficacia, es esencial contar con una infraestructura de comunicaciones robusta, ágil y de largo alcance que supere las limitaciones de cobertura y funcione en condiciones extremas.

El presente trabajo surge del proyecto previo **Oroneta**, un sistema de automatización y gestión de enjambres de drones para tareas de rescate. Ante las limitaciones en la transmisión de broadcast y la falta de una arquitectura de redes adecuada, se identifica la necesidad de desarrollar un estándar de comunicación inalámbrica de largo alcance, que facilite una gestión de red más dinámica y un broadcast eficiente.

El proyecto en el que se centra el trabajo, denominado **ARIS Radio** (*Adaptive Radio Interchange System*) se enfoca en la tecnología **LoRa**, que utiliza modulación de espectro ensanchado basada en **CSS** (*Chirp Spread Spectrum*). LoRa maximiza la potencia de la señal, ofreciendo comunicaciones de largo alcance, alta sensibilidad y bajo consumo energético. Esta tecnología es ideal para superar las limitaciones de otras soluciones en entornos de cobertura limitada y es perfecta para aplicaciones IoT, redes de sensores y sistemas distribuidos en zonas remotas.

El objetivo de este trabajo es diseñar un protocolo de comunicación y desarrollar un módulo hardware, que incluye tanto el código como el diseño electrónico, siguiendo los estándares establecidos y las directrices de la arquitectura de red. Este dispositivo será compatible con otros *hardware* integrados dentro de rovers, satélites, drones, sistemas IoT (Arduino) o cualquier dispositivo hardware, y ofrecerá compatibilidad con dispositivos personales como PC, móviles o sistemas embebidos, a través de un driver que simplifique su implementación y uso.

Palabras clave: Protocolos de comunicación, satélites, dispositivos hardware, comunicación inalámbrica, modulación de espectro, LoRa, drivers, arquitectura de red, Arduino IoT, enjambre de drones, infraestructura, diseño electrónico

Abstract

In an increasingly interconnected and technology-dependent world, coordinating autonomous systems in critical missions has become a fundamental challenge. The automation of "rovers," space missions, interconnection of satellite devices, and IoT communication in coverage-limited environments have gained significant importance. To ensure their effectiveness, it is essential to have a robust, agile, and long-range communication infrastructure that overcomes coverage limitations and operates in extreme conditions.

This work stems from the previous **Oroneta project**, a drone swarm automation and management system for rescue tasks. Given the limitations in broadcast transmission and the lack of an appropriate network architecture, the need to develop a long-range wireless communication standard is identified, one that facilitates more dynamic network management and efficient broadcast.

The project this work focuses on, called **ARIS Radio** (Adaptive Radio Interchange System), focuses on **LoRa** technology, which uses **CSS** (Chirp Spread Spectrum) based spread spectrum modulation. LoRa maximizes signal power, offering long-range communication, high sensitivity, and low energy consumption. This technology is ideal for overcoming the limitations of other solutions in environments with limited coverage and is perfect for IoT applications, sensor networks, and distributed systems in remote areas.

The objective of this work is to design a communication protocol and develop a hardware module, which includes both the code and the electronic design, following the established standards and network architecture guidelines. This device will be compatible with other hardware integrated into rovers, satellites, drones, IoT systems (Arduino), or any hardware device, and will offer compatibility with personal devices such as PCs, mobile phones, or embedded systems through a driver that simplifies its implementation and use.

Key words: Communication protocols, satellites, hardware devices, wireless communication, spectrum modulation, LoRa, drivers, network architecture, Arduino IoT, drone swarm, infrastructure, electronic design

Índice general

Índice de figuras

Índice de tablas

Agradecimientos

Con este trabajo concluye una etapa significativa en mi vida, y por ello, deseo expresar mi más sincero agradecimiento a todas aquellas personas que han contribuido a la realización de este proyecto.

En primer lugar, quiero manifestar mi profunda gratitud a mi tutor, **Juan Luis Posadas Yagüe**, por su invaluable conocimiento, orientación y apoyo a lo largo de este proceso. Sus consejos y su implicación han sido fundamentales para el desarrollo y éxito de este trabajo. Asimismo, deseo agradecer a los miembros del equipo con quienes inicié este proyecto dentro de **ARIS Alliance**, en especial a Tomás Otero Matteri y Daniel Dimov Kisyov, por su compromiso y dedicación en los primeros pasos del proyecto, permitiéndonos ver materializados los resultados de nuestro esfuerzo conjunto.

De igual manera, extiendo mi reconocimiento a los integrantes del equipo con el que colaboré en un proyecto predecesor, **Oroneta**: Alexis Montalvo Callau, Elena Clofent Muñoz, Zhi Lin Li, María Zapata y Hai Tao Wu, quienes, con su trabajo y colaboración, sentaron las bases para lo que hoy es este proyecto.

No quiero dejar de mencionar a todos mis compañeros que me han acompañado a lo largo de estos cuatro años, quienes han sido una fuente constante de aprendizaje y motivación. También agradezco a la empresa **F1-Connecting** por el conocimiento adicional que me ha brindado, así como por las valiosas experiencias de trabajo en equipo. Finalmente, expreso mi reconocimiento a la Universitat Politècnica de València por proporcionarme un entorno académico enriquecedor y los recursos necesarios para mi desarrollo personal y profesional.

A todos ustedes, gracias. Sin su apoyo, este proyecto no habría sido posible.

Motivación

0.1 Motivación personal

Antes de comenzar el Grado en Ingeniería Informática, ya sentía un profundo interés por el mundo de la tecnología. Todo comenzó una tarde, a principios del segundo año de secundaria, cuando un amigo me mostró cómo modificar una línea de código usando lo que hoy conocemos como el inspector de elementos del navegador. Ese pequeño gesto fue suficiente para abrirme las puertas a un universo completamente nuevo. Desde ese momento, comencé a aprender por mi cuenta: buscaba tutoriales, libros y cursos que me ayudaran a entender más sobre el desarrollo web. Fue un camino autodidacta, lleno de entusiasmo pero también de desafíos.

Con el tiempo, me especialicé en desarrollo web y, confiando en mis habilidades, decidí postularme a vacantes como desarrollador. Sin embargo, a pesar de tener los conocimientos y la experiencia, fui rechazado en varias ocasiones por no contar con un título formal. Esa fue una de las razones que me impulsó a entrar a la universidad. Ya dentro del Grado, me di cuenta de la verdadera complejidad del sector: los fundamentos teóricos, las estructuras profundas y la lógica detrás de cada sistema.

Durante la carrera, además de afianzar lo que ya conocía, descubrí un nuevo campo que despertó aún más mi interés: la programación de bajo nivel y el desarrollo cercano al hardware. Un área que, aunque poco popular, requiere paciencia, precisión y una comprensión profunda del funcionamiento interno de los sistemas. Y fue allí donde encontré una nueva pasión dentro del amplio mundo de la informática.

Además, gracias a la empresa F1-Connecting de haber confiado en mí y darme la oportunidad de trabajar con ellos, a pesar de que aún me encontraba en el primer curso de la carrera. En un momento en el que la mayoría de las empresas priorizan títulos y experiencia, F1-Connecting no dudó en acogerme sin discriminar por esos aspectos. Esta oportunidad fue mucho más que un simple trabajo; fue una experiencia que me permitió desarrollarme profesionalmente y aprender de manera práctica.

En F1-Connecting, adquirí valiosos conocimientos sobre cómo trabajar en equipo, una habilidad que considero esencial en cualquier ámbito profesional. Además, pude explorar el sector de los pagos, un campo que me era desconocido pero que resultó ser fascinante y complejo. La experiencia me permitió adentrarme en la programación de bajo nivel, un área que, aunque desafiante, me ofreció una perspectiva completamente diferente sobre la interacción entre *hardware* y *software*.

Este TFG y el proyecto que lo acompaña representan un paso clave en ese camino, ya que me brindan la oportunidad de combinar todo lo aprendido a lo largo de la carrera con mis intereses en sistemas de comunicación, dispositivos físicos y el desarrollo de soluciones prácticas. A través de este trabajo, puedo aplicar mis conocimientos teóricos

a un reto real y tangible, mientras continúo perfeccionando esas habilidades blandas que considero igualmente esenciales para mi futuro profesional.

No es solo un proyecto académico, sino una oportunidad para profundizar en el tipo de tecnología que realmente me apasiona y que quiero seguir explorando en el futuro. El desarrollo de sistemas de comunicación eficientes, la integración de *hardware* y *software*, y la creación de soluciones que puedan tener un impacto significativo en el mundo real son los aspectos que me motivan a seguir adelante en este ámbito. Y, sobre todo, este proyecto representa un paso hacia la consolidación de mis conocimientos y habilidades para futuros desafíos, tanto técnicos como de liderazgo, en el mundo de la tecnología.

0.2 Motivación profesional

Conforme fui creciendo y avanzando en mi formación, descubrí que no solo me apasionaba la tecnología en general, sino que sentía un interés cada vez más fuerte por el campo aeroespacial y el sector de defensa militar. Siempre me ha fascinado cómo estos ámbitos combinan innovación tecnológica, precisión extrema y una constante búsqueda de soluciones seguras y eficientes en entornos críticos. Por esta razón, gran parte de mis decisiones académicas, así como la orientación de mis proyectos, han tenido como objetivo acercarme lo máximo posible a este tipo de aplicaciones.

Este Trabajo de Fin de Grado representa, para mí, mucho más que una simple culminación académica. Me ha permitido profundizar en un tema que considero estratégico tanto desde una perspectiva técnica como profesional. A través de este proyecto, he podido conectar mis conocimientos en programación de bajo nivel, integración de sistemas físicos y protocolos de comunicación con un enfoque aplicado, que se alinea directamente con las exigencias del sector aeroespacial y de defensa.

Además, trabajar en un proyecto con estas características me ha permitido aplicar no solo lo aprendido a nivel técnico durante la carrera, sino también las habilidades de liderazgo, gestión de equipos y visión estratégica que he desarrollado a lo largo de mi experiencia práctica, tanto en el entorno académico como profesional.

CAPÍTULO 1
Objetivos ODS

Este trabajo se ha diseñado con la intención de contribuir a los Objetivos de Desarrollo Sostenible (ODS) establecidos por las Naciones Unidas, específicamente en el area de ciudades y comunidades sostenibles (ODS 11) y la industria, innovación e infraestructura (ODS 9). A través del desarrollo de un sistema de comunicación inalámbrica eficiente y adaptable, se busca fomentar la innovación tecnológica que pueda ser aplicada en entornos urbanos inteligentes, sistemas autónomos y redes IoT distribuidas, promoviendo así un uso más sostenible de los recursos tecnológicos y mejorando la calidad de vida en las comunidades.

Al mismo se quiere reducir la creación de nuevos protocolos de comunicación propietarios y cerrados en la que se generan muchos residuos electrónicos y se limita la interoperabilidad entre dispositivos, fomentando un enfoque más abierto y colaborativo en el desarrollo de tecnologías de comunicación.

CAPÍTULO 2
Introducción

En la actualidad, la interconexión de dispositivos y sistemas en diversos campos, como la exploración espacial, el monitoreo ambiental y las aplicaciones de rescate, ha incrementado la importancia de coordinar de manera efectiva sistemas autónomos. La automatización de *rovers*, misiones espaciales, interconexión de dispositivos satelitales y la comunicación en entornos sin cobertura de redes tradicionales han adquirido relevancia debido a la necesidad de garantizar una comunicación eficiente y robusta en condiciones extremas. Estos avances tecnológicos han planteado nuevos desafíos en el ámbito de las infraestructuras de comunicación, que deben ser lo suficientemente **ágiles**, **escalables** y de **largo alcance** para superar las limitaciones de cobertura y operar en zonas remotas.

Este trabajo se desarrolla a partir del proyecto previo **Oroneta**, una plataforma orientada a la automatización y coordinación de enjambres de drones para tareas de rescate en escenarios de difícil acceso. Durante el desarrollo de Oroneta, se identificaron limitaciones importantes en la capacidad de transmisión *broadcast* y en la arquitectura de red, lo que evidenció la necesidad de diseñar un sistema de comunicación más **eficiente**, **escalable** y compatible con distintos entornos operativos. De ahí surge la iniciativa de crear un nuevo módulo de comunicación inalámbrica, tanto a nivel de **protocolo** como de **hardware**, que permita una gestión de red más dinámica, una interoperabilidad más amplia y una integración sencilla con distintos tipos de dispositivos, desde plataformas móviles y satelitales hasta sensores IoT y sistemas embebidos.

Ante esta situación, se necesita ampliar las capacidades de los sistemas actuales, desarrollando tecnologías de comunicación que no solo sean **resistentes** y **eficientes**, sino también **adaptables** a diferentes topologías de red y tipos de misión. La dependencia de redes centralizadas o de infraestructura terrestre limitada no es viable en muchos de estos contextos, lo que impulsa la búsqueda de soluciones inalámbricas capaces de operar de manera autónoma, con **bajo consumo energético** y **alta confiabilidad**.

En un principio, se optó por desarrollar un sistema de radio físico propio, con características específicas orientadas a alcanzar un mayor rango de cobertura y robustez frente a condiciones adversas. Sin embargo, debido a la limitación de tiempo disponible para investigación y pruebas, y con el objetivo de validar de forma efectiva un primer prototipo funcional, se decidió adoptar temporalmente la tecnología **LoRa** como capa física. Esta elección permite aprovechar una solución ya establecida, basada en modulación de espectro ensanchado mediante CSS, que ofrece comunicaciones de **largo alcance**, **alta sensibilidad** y **bajo consumo energético**. En este contexto, LoRa se presenta como una alternativa sólida para entornos distribuidos y de difícil acceso.

No obstante, el proyecto contempla a largo plazo el desarrollo completo del sistema de radio propio, con el objetivo de superar las limitaciones actuales de **ancho de banda** impuestas por LoRa y escalar el rendimiento del sistema, dividiendo el enfoque

del proyecto en dos ámbitos principales: uno orientado a **aplicaciones espaciales** y otro centrado en el ecosistema IoT. El Trabajo de Fin de Grado, trabajo actual, se enfocará específicamente en la **capa de comunicación inalámbrica** del modelo de comunicaciones, asumiendo LoRa como tecnología base en la capa física, mientras que la capa de red será abordada de forma parcial pero no en profundidad.

2.1 Objetivos

El sistema desarrollado tiene como propósito principal obtener una comunicación fluida y estable entre distintos dispositivos físicos, tanto en entornos especializados como *rovers*, satélites, drones o sistemas IoT basados en plataformas como Arduino, entre muchos otros, como en dispositivos de uso general, tales como ordenadores personales, teléfonos móviles o sistemas embebidos.

El sistema busca garantizar principalmente las siguientes directrices:

- Descentralizado
- Largo alcance
- Red escalable
- Estable
- Seguro
- Red privada
- Rápido

Para lograr este objetivo se plantean los siguientes pasos específicos:

- **Diseño del protocolo de comunicación de red:** Desarrollo de un protocolo propio, eficiente y adaptable, orientado a la transmisión de datos entre dispositivos físicos en entornos con recursos limitados, garantizando integridad, sincronización y compatibilidad.
- **Diseño del hardware:** Creación de un módulo físico que integra el protocolo de comunicación definido, incluyendo el uso de buses estándar como SPI, así como la definición del formato de datos necesario para la comunicación efectiva entre dispositivos.
- **Diseño de un driver de comunicación:** Aunque por cuestiones de tiempo se ha abordado de forma limitada, este componente es clave para la integración del sistema con plataformas externas (PCs, móviles o sistemas embebidos). Su desarrollo permite una abstracción del hardware, facilitando su uso e implementación.
- **Diseño de la arquitectura de red:** Definición de la estructura general del sistema de comunicación, incluyendo la organización de los dispositivos, flujos de datos, jerarquías y protocolos de enlace, con el objetivo de asegurar escalabilidad, modularidad y eficiencia en entornos distribuidos o heterogéneos (de forma básica).

A nivel técnico, los objetivos específicos del proyecto se detallan a continuación:

1. Diseñar un módulo de comunicación *hardware* con capacidad para integrarse en diferentes plataformas físicas (como sistemas embebidos, placas de desarrollo o *hardware* personalizado).
2. Desarrollar un protocolo de comunicación eficiente, seguro y flexible, adaptable a diferentes entornos de red y capaz de soportar múltiples dispositivos.

3. Implementar un driver multiplataforma que facilite la interoperabilidad del sistema con PCs, móviles u otros dispositivos externos.

4. Aplicar principios de programación de bajo nivel, con especial atención al manejo directo del hardware, la gestión de memoria y la optimización de recursos.

5. Garantizar la fiabilidad del sistema mediante pruebas funcionales, validación de datos, control de errores y mecanismos de recuperación ante fallos.

6. Documentar detalladamente tanto el diseño como el funcionamiento del sistema, de manera que pueda ser utilizado, ampliado o adaptado por terceros en otros proyectos.

2.2 Impacto esperado

Uno de los principales impactos esperados es que esta arquitectura pueda ser adoptada o considerada por fabricantes y desarrolladores de sistemas embebidos, dispositivos IoT, plataformas robóticas o incluso sistemas aeroespaciales o de defensa como una alternativa frente a las soluciones existentes en el mercado. Su enfoque abierto, su orientación con múltiples plataformas y su énfasis en la seguridad y la simplicidad de integración la convierten en un candidato viable para proyectos donde se requieren soluciones de comunicación personalizadas, seguras y optimizadas.

Además, al tratarse de un proyecto de código fuente abierto, se facilita que otros desarrolladores, investigadores o empresas puedan analizar, adaptar y extender el sistema según sus propias necesidades. Esto fomenta la creación de variantes específicas del protocolo y del *hardware*, promueve la colaboración en comunidad y permite una evolución tecnológica continua, basada en la mejora colectiva y el uso compartido del conocimiento.

Este trabajo, por tanto, no solo busca ofrecer una solución funcional inmediata, sino también contribuir al ecosistema tecnológico y académico como una base sólida para futuras investigaciones, desarrollos comerciales y aplicaciones reales en sectores de alta exigencia tecnológica, donde la fiabilidad y la integración entre *hardware* y *software* son fundamentales.

2.3 Trabajo

El propósito de este apartado es, por tanto, contextualizar el trabajo individual en el marco general del proyecto, delimitando claramente las contribuciones realizadas y permitiendo al lector identificar de forma precisa el alcance, los objetivos y las decisiones técnicas que corresponden directamente al trabajo desarrollado en este TFG.

Para facilitar la lectura de relación trabajo - proyecto se va a mostrar un esquema del modelo ARISR por capas de conocimiento a continuación:

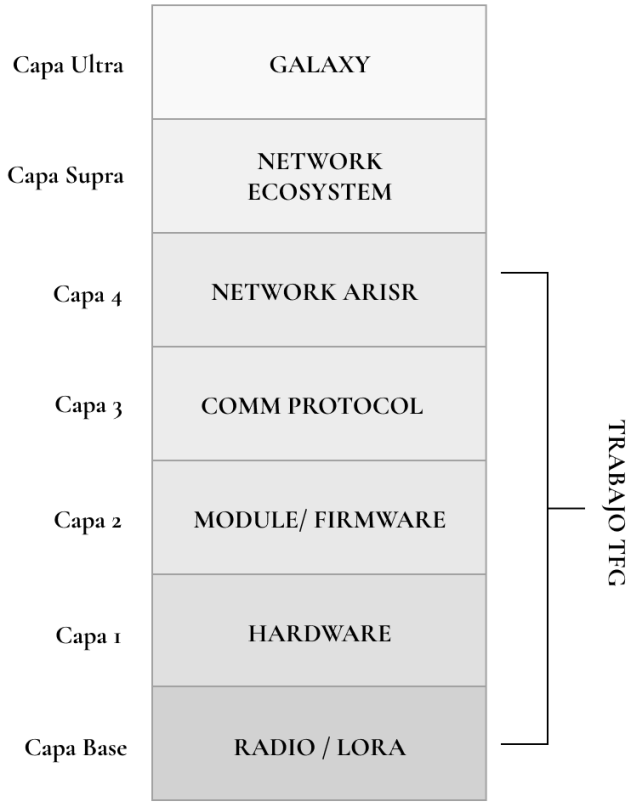

Figura 2.1: Esquema del modelo ARISR por capas de conocimiento

Y para presentar los distintos puntos que se han desarrollado para el proyecto del ámbito que abarca, es decir Capa Base, Capa 1, 2, 3 y 4 se adjunta de forma simplificada un diagrama de componentes.

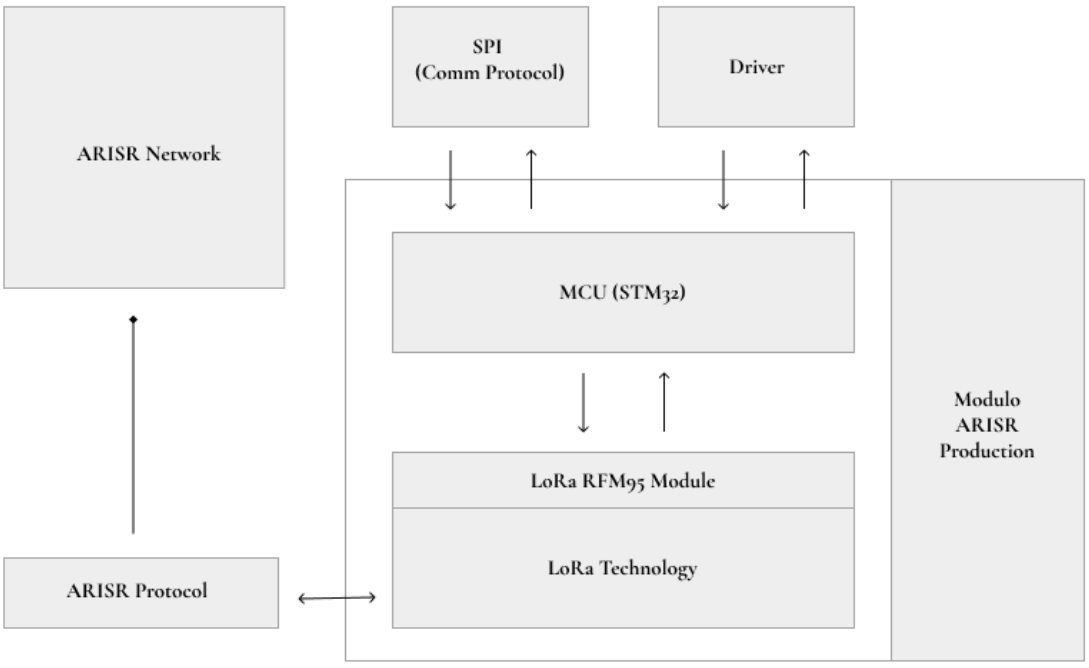

Figura 2.2: Diagrama de componentes

Sin embargo, el presente trabajo no abarca todos los elementos detallados en el diagrama, por lo que se especificará sólo lo que se incluirá en esta versión simplificada. Algunos componentes están representados con baja opacidad, lo que indica que no forman parte de este trabajo, aunque se mencionan debido a que fueron desarrollados previamente, como el LoRa, o por otro miembro del equipo.

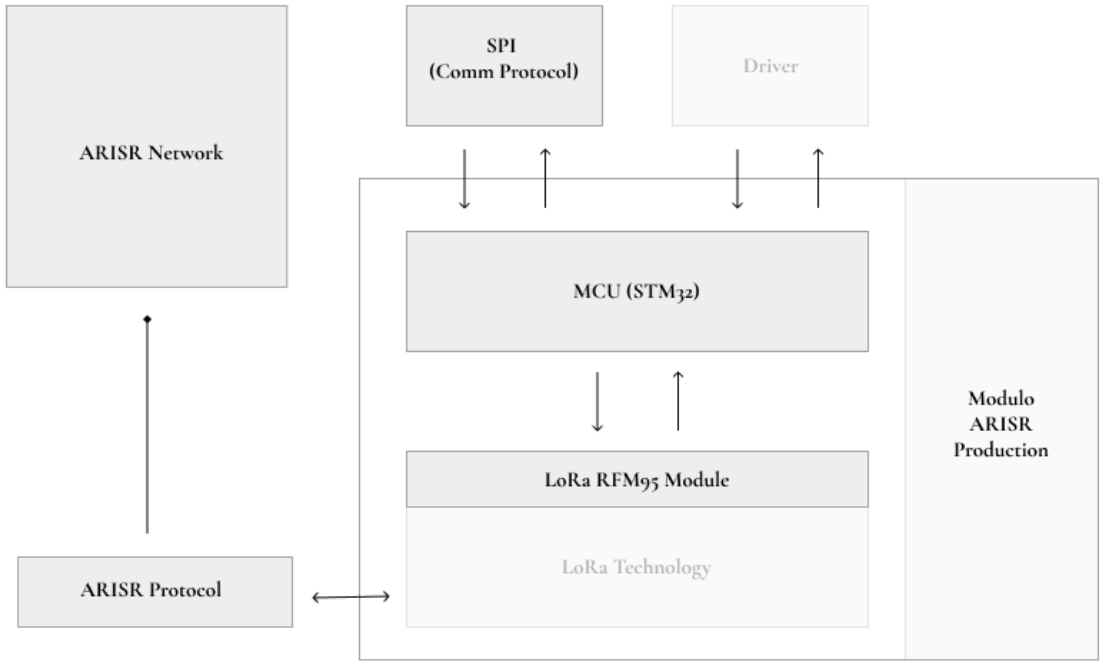

Figura 2.3: Diagrama de componentes del ámbito del TFG

2.4 Metodología

Con el fin de alcanzar los objetivos planteados en este trabajo, se ha seguido una metodología estructurada basada en el desarrollo iterativo y la validación práctica de cada uno de los componentes. A continuación, se describen las fases principales que han guiado el proceso de desarrollo:

1. **Investigación técnica preliminar**

   Se realizó una revisión teórica sobre protocolos de comunicación, interfaces *hardware* como SPI, drivers de bajo nivel y arquitecturas de red en sistemas embebidos. Esta fase fue fundamental para establecer las bases técnicas del proyecto y tomar decisiones informadas sobre el diseño.

2. **Definición de la arquitectura del sistema**

   Se diseñó una arquitectura modular que define cómo interactúan los distintos elementos del sistema: hardware, protocolo de comunicación, driver y dispositivos externos. Esta arquitectura fue pensada para ser escalable y adaptable a distintos escenarios de uso.

3. **Diseño del protocolo de comunicación**

   Se implementó un protocolo propio orientado a garantizar la estabilidad y la eficiencia en la transmisión de datos. El protocolo contempla la estructura de paque-

tes, el manejo de errores, la sincronización y la compatibilidad con múltiples dispositivos, así como mecanismos de broadcast eficiente y gestión de red.

4. **Diseño y desarrollo del hardware**

   Se llevó a cabo el diseño electrónico del módulo de comunicación, incluyendo la definición de pines, la conexión mediante SPI, el formato de datos y la compatibilidad con plataformas como Arduino y otros sistemas embebidos. Esta fase también abarcó la elaboración de esquemáticos y la realización de pruebas de validación en un entorno controlado.

2.5 Estructura de la memoria

La memoria de este Trabajo Fin de Grado se organiza en siete capítulos adicionales al presente. A continuación, se describe brevemente el contenido de cada uno de ellos:

- **Capítulo 2. Marco teórico y estado del arte.** Introduce conceptos y tecnologías relevantes como LoRaWAN, ExpressLRS, LTE-M, Zigbee, Bluetooth Mesh, DTN (Delay/Disruption Tolerant Networking), CCSDS y SCPS, junto con sus propuestas de arquitectura de red. Su objetivo es proporcionar el contexto necesario para comprender los aspectos técnicos del trabajo y justificar las decisiones de diseño adoptadas.

- **Capítulo 3. Alternativas a WiFi para IoT distribuido.** Analiza las limitaciones del uso de WiFi y del estándar ISO/IEC 8266 en el ámbito abordado, a pesar de su amplia adopción. Se comparan estas tecnologías con alternativas más adecuadas para entornos exigentes, destacando las ventajas de la solución propuesta.

- **Capítulo 4. LoRa y hardware.** Describe el hardware empleado en el sistema, centrado en microcontroladores STM32. Se detalla su integración mediante SPI, la selección de la frecuencia de 868 MHz y el flujo de datos gestionado por el MCU. Asimismo, se justifican las decisiones de diseño de la placa y la elección de componentes.

- **Capítulo 5. Primeros pasos.** Presenta las acciones preliminares al desarrollo, incluyendo herramientas de apoyo para el prototipado y la validación inicial del firmware. Se establecen así las bases para un desarrollo progresivo y fiable.

- **Capítulo 6. Comunicación serial.** Describe la comunicación serie entre los distintos componentes del sistema, el formato de intercambio de datos y los registros SPI utilizados. También se explica el funcionamiento de la comunicación asíncrona y las normas de uso del sistema.

- **Capítulo 7. Librerías y open source.** Detalla las librerías externas empleadas y presenta las desarrolladas para ARISR, como `lib-protoarisr-c`. Se describe la estructura del firmware, el entorno de desarrollo y el modelo de licenciamiento. Además se profundiza en las librerías propias del sistema, describiendo su arquitectura, funcionalidades e integración con el hardware y el protocolo. Se incluyen ejemplos de uso.

- **Capítulo 8. Protocolo de comunicación.** Describe el protocolo diseñado, incluyendo la estructura de los mensajes, los campos que los componen y la máquina de estados asociada. También se abordan los mecanismos de control de errores y fragmentación.

- **Capítulo 9. Seguridad de la comunicación.** Expone los mecanismos de integridad y cifrado empleados, como CRC-16-CCITT y AES-128, así como los métodos de autenticación utilizados.
- **Capítulo 10. Arquitectura de red.** Define la arquitectura de red del sistema, incluyendo los roles de los nodos, las reglas de comunicación y la topología descentralizada adoptada.
- **Capítulo 11. Implementación y resultados.** Describe la integración del sistema completo y presenta los resultados experimentales obtenidos en distintos escenarios de prueba.
- **Capítulo 12. Conclusiones.** Resume los resultados alcanzados, evalúa el cumplimiento de los objetivos y plantea posibles líneas de trabajo futuro.

2.6 Abreviaciones y glosario

A continuación, se presenta una lista de abreviaturas, conceptos y tecnologías utilizadas a lo largo de este trabajo, junto con sus respectivas definiciones. Este glosario tiene como objetivo facilitar la comprensión de los términos técnicos empleados, especialmente en el ámbito de las comunicaciones inalámbricas, sistemas embebidos y arquitecturas distribuidas.

SPI *Serial Peripheral Interface*. Protocolo de comunicación síncrono ampliamente utilizado en sistemas embebidos para la transferencia de datos entre un microcontrolador y uno o varios periféricos. Se basa en un modelo maestro-esclavo y utiliza líneas dedicadas para reloj, datos de entrada y salida.

MCU *Microcontroller Unit*. Dispositivo integrado que combina en un único circuito una unidad de procesamiento (CPU), memoria y periféricos de entrada/salida, utilizado para el control de sistemas electrónicos.

LoRa *Long Range*. Tecnología de comunicación inalámbrica de largo alcance basada en modulación de espectro ensanchado mediante *Chirp Spread Spectrum*. Se caracteriza por su alta sensibilidad y bajo consumo energético, lo que la hace adecuada para entornos distribuidos y de difícil acceso.

LoRaWAN *Long Range Wide Area Network*. Protocolo de red que define la arquitectura y el funcionamiento de redes basadas en LoRa, orientado a aplicaciones IoT de bajo consumo y largo alcance.

IoT *Internet of Things*. Paradigma tecnológico que describe la interconexión de dispositivos físicos a través de redes de comunicación, permitiendo la recopilación, intercambio y procesamiento de datos de forma automatizada.

DTN *Delay/Disruption Tolerant Networking*. Arquitectura de red diseñada para operar en entornos con alta latencia, desconexiones frecuentes o enlaces intermitentes, como redes espaciales o escenarios de emergencia.

CCSDS *Consultative Committee for Space Data Systems*. Organización internacional que define estándares para la comunicación y gestión de datos en misiones espaciales.

SCPS *Space Communications Protocol Specifications*. Conjunto de protocolos optimizados para mejorar el rendimiento de las comunicaciones en redes con alta latencia y limitaciones de ancho de banda, especialmente en entornos espaciales.

ExpressLRS *Express Long Range System*. Sistema de comunicación de radiofrecuencia de código abierto, diseñado para ofrecer largo alcance, baja latencia y alta eficiencia, especialmente en aplicaciones de control remoto de drones.

LTE-M *Long Term Evolution for Machines*. Tecnología celular derivada de LTE, optimizada para dispositivos IoT que requieren bajo consumo energético, amplia cobertura y buena penetración en interiores.

Zigbee Protocolo de comunicación inalámbrica basado en el estándar IEEE 802.15.4, utilizado en redes de sensores, automatización y aplicaciones de bajo consumo.

Bluetooth Mesh Extensión del protocolo Bluetooth Low Energy que permite la creación de redes malladas entre múltiples dispositivos, facilitando la comunicación en aplicaciones IoT distribuidas.

STM32 Familia de microcontroladores basados en arquitectura ARM Cortex desarrollados por STMicroelectronics, ampliamente utilizados en sistemas embebidos por su versatilidad y eficiencia.

SX1276 Chip transceptor de radiofrecuencia desarrollado por Semtech, capaz de operar con modulación LoRa y FSK, utilizado para comunicaciones inalámbricas de largo alcance.

RF95 Módulo de radio basado en el chip SX1276, utilizado en el prototipado de sistemas LoRa, que integra los componentes necesarios para facilitar su uso en sistemas embebidos.

ARISR Sistema de comunicación desarrollado en este trabajo, orientado a proporcionar una arquitectura flexible, escalable y eficiente para la interconexión de dispositivos en entornos distribuidos.

CRC *Cyclic Redundancy Check*. Mecanismo de detección de errores utilizado para verificar la integridad de los datos transmitidos en una comunicación.

AES *Advanced Encryption Standard*. Algoritmo de cifrado simétrico ampliamente utilizado para garantizar la confidencialidad de la información en sistemas de comunicación.

Chunk Unidad básica de datos definida en el protocolo ARISR, que encapsula la información transmitida junto con los campos de control, direccionamiento y verificación.

Broadcast Mecanismo de transmisión en el que un mensaje es enviado a todos los nodos de la red sin necesidad de direccionamiento específico.

Nodos Dispositivos o entidades dentro de una red que pueden enviar, recibir o retransmitir información. En el contexto de este trabajo, se refiere a los dispositivos físicos que forman parte del sistema de comunicación ARISR.

LPWAN *Low Power Wide Area Network*. Categoría de tecnologías de comunicación inalámbrica diseñadas para ofrecer conectividad de largo alcance con bajo consumo energético, ideal para aplicaciones IoT.

UAVs Aeronaves no tripuladas utilizadas en aplicaciones de monitoreo y control.

CAPÍTULO 3
Marco teórico, estado del arte

Con el objetivo de ayudar a entender mejor el contexto y visión del presente trabajo, en este capítulo se describen los conceptos teóricos relacionados, así como las tecnologías y herramientas empleadas para desarrollar el presente trabajo. En primer lugar, se introduce un modelo de competencia denominado LoRaWAN junto con sus desventajas en relación en múltiples escenarios. En segundo lugar, se introducen los modelos de estudio que han partido el proyecto: ExpressLRS y LTE-M. Y modelos de referencia que han sido ayuda a la implementación de ARIS cómo: Zigbee, Bluetooth Mesh, DTN – Delay/Disruption Tolerant Networking, CCSDS y SCPS. Finalmente, se presentan las comparaciones de los modelos y escenarios de uso de cada uno y áreas que puede abarcar.

3.1 LoRaWAN

LoRaWAN es un protocolo de red diseñado para comunicaciones inalámbricas de largo alcance, baja potencia y baja tasa de datos. Esta tecnología forma parte del ecosistema LPWAN y se utiliza comúnmente en aplicaciones de IoT, tales como monitoreo ambiental, control de infraestructura urbana, agricultura inteligente, entre otros. La arquitectura de LoRaWAN está basada en una topología de estrella, donde los dispositivos finales (nodos) se comunican directamente con *gateways*, los cuales actúan como puente entre los dispositivos y el servidor de red central. La modulación física utilizada es LoRa, que permite transmisiones a larga distancia (hasta 15 km en condiciones óptimas) y gran penetración en interiores, aunque sacrificando velocidad de transmisión.

Entre sus principales ventajas destacan:

- Bajo consumo energético, ideal para dispositivos que funcionan con baterías durante largos períodos.
- Cobertura extensa, lo que permite conectar dispositivos en zonas rurales o de difícil acceso.
- Licenciamiento gratuito del espectro ISM, lo que facilita su implementación sin costos adicionales por uso de frecuencia.

No obstante, LoRaWAN también presenta limitaciones importantes, especialmente en escenarios que requieren mayor capacidad de datos o baja latencia. Entre sus principales desventajas se encuentran:

- Baja tasa de transferencia, que lo hace inadecuado para aplicaciones que requieren enviar grandes volúmenes de datos o actualizaciones frecuentes.

- Limitada escalabilidad en entornos densos, ya que la coexistencia de múltiples dispositivos puede generar colisiones y pérdida de paquetes.

- Seguridad básica, aunque ofrece mecanismos de cifrado AES de extremo a extremo, la gestión de claves y autenticación puede ser limitada en algunas implementaciones.

Además, al basarse en una arquitectura de red en estrella, LoRaWAN presenta un enfoque altamente centralizado, donde todos los dispositivos finales se comunican exclusivamente con un *gateway* o servidor central. Esta estructura implica que no existe comunicación directa entre nodos (no hay transmisión *peer-to-peer*) ni entre diferentes redes. Como consecuencia, LoRaWAN no admite nativamente el uso de broadcast, lo que limita su aplicabilidad en escenarios que requieren la distribución simultánea de mensajes a múltiples dispositivos, como actualizaciones masivas, alertas en tiempo real o acciones sincronizadas entre dispositivos.

Aunque el protocolo LoRaWAN ha introducido capacidades de multicast a partir de la versión 1.1, esta funcionalidad aún se encuentra sujeta a múltiples restricciones. Por ejemplo, para que los dispositivos puedan recibir mensajes multicast de forma eficiente, deben estar sincronizados en tiempo y escuchar en una ventana específica, lo que entra en conflicto con el objetivo principal de LoRaWAN de mantener el consumo energético al mínimo. Además, la implementación del multicast requiere una configuración previa y específica de los dispositivos, lo que limita su uso dinámico o espontáneo en aplicaciones móviles o cambiantes.

Por otro lado, la ausencia de capacidades de retransmisión entre nodos implica que LoRaWAN no es compatible con topologías en malla, a diferencia de otras tecnologías de red de bajo consumo como Zigbee, Thread o Bluetooth Mesh. Esta carencia restringe significativamente la expansión orgánica de la red, ya que la cobertura depende exclusivamente del gateway y no puede ser extendida por los propios nodos. Como resultado, en entornos urbanos densos, edificios con múltiples niveles o zonas con obstrucciones físicas, la conectividad puede verse severamente afectada si no se planifica una infraestructura de *gateways* adecuada.

Asimismo, la comunicación entre diferentes redes LoRaWAN (por ejemplo, entre redes de distintos operadores o infraestructuras) no está prevista por defecto en el estándar, lo que representa un obstáculo adicional para la interoperabilidad y escalabilidad del sistema. Aunque existen iniciativas y soluciones propietarias que buscan superar estas barreras, como LoRa roaming o redes federadas, su adopción aún es limitada y su implementación compleja.

En resumen, la naturaleza centralizada, la falta de broadcast real, la dependencia de *gateways* para toda la comunicación y la ausencia de transmisión entre redes o nodos, convierten a LoRaWAN en una solución potente para ciertos contextos muy específicos (como IoT rural o sensorización de bajo tráfico), pero la hacen inadecuada para sistemas distribuidos, dinámicos o colaborativos donde la comunicación entre dispositivos y la flexibilidad de red son elementos clave.

Dado este contexto, el uso de LoRaWAN se justifica principalmente en aplicaciones donde la eficiencia energética y la cobertura son más importantes que la velocidad o la capacidad de transmisión. Este análisis permite entender sus ventajas comparativas respecto a otras tecnologías como ExpressLRS y LTE-M, las cuales se explorarán en los apartados siguientes.

3.2 Proyectos de estudio

Para contextualizar y fundamentar el desarrollo del presente trabajo, se han tomado como referencia de estudio dos tecnologías relevantes que presentan enfoques distintos al de LoRaWAN: ExpressLRS y LTE-M. Estos modelos ofrecen alternativas viables en distintos escenarios y permiten establecer comparaciones significativas en términos de velocidad, latencia, escalabilidad y cobertura.

3.2.1. ExpressLRS

ExpressLRS es un protocolo de radiofrecuencia de código abierto, diseñado originalmente para el control de drones y UAVs. Utiliza modulación LoRa, pero con una filosofía completamente diferente a LoRaWAN, priorizando baja latencia y alta tasa de actualización, lo cual lo hace ideal para aplicaciones en tiempo real.

Entre sus características principales se destacan:

- Latencia extremadamente baja, esencial para sistemas de control.
- Alta velocidad de actualización (hasta 500 Hz en versiones recientes).
- Uso eficiente del espectro, soportando frecuencias de 900 MHz y 2.4 GHz.
- Flexibilidad y personalización gracias a su comunidad activa de desarrollo open source.

A diferencia de LoRaWAN, ExpressLRS no emplea una arquitectura de red con *gateways* ni servidores centrales. La comunicación es directa entre el transmisor y el receptor (enlace punto a punto), lo que permite un control preciso pero limita su escalabilidad para aplicaciones de IoT masivo. Por esta razón, ExpressLRS no está pensado como un sistema de comunicación IoT tradicional, pero ofrece un modelo alternativo y eficiente para enlaces robustos de alta velocidad en distancias medias.

3.2.2. LTE-M

LTE-M, también conocido como Cat-M1, es una tecnología diseñada específicamente para dispositivos IoT por el consorcio 3GPP dentro del estándar LTE (4G). Su objetivo es ofrecer un equilibrio entre cobertura, velocidad, consumo energético y costes operativos, permitiendo que dispositivos con recursos limitados se comuniquen eficientemente a través de redes móviles.

LTE-M utiliza la infraestructura celular existente, con una arquitectura centralizada gestionada por operadores móviles. Los dispositivos se conectan directamente al núcleo de la red mediante SIM o eSIM, sin necesidad de *gateways* intermedios, lo que facilita la escalabilidad, la movilidad y el roaming entre celdas. Además, permite comunicación *multicast* y *group messaging* mediante mecanismos como MBMS, útil para actualizaciones de *firmware*, envío de alertas y sincronización de múltiples dispositivos.

Entre sus ventajas destacan la cobertura extendida gracias a eDRX y PSM, velocidades de hasta 1 Mbps, baja latencia y alta seguridad basada en autenticación SIM y cifrado. Sin embargo, presenta mayor consumo energético que LoRaWAN o NB-IoT, dependencia de operadores móviles y un coste de *hardware* superior. Se utiliza comúnmente en monitoreo industrial, rastreo de activos, dispositivos médicos, sistemas de seguridad y aplicaciones de ciudades inteligentes.

3.3 Proyectos de referencia

Aunque ExpressLRS y LTE-M ofrecen enfoques interesantes y complementarios para redes IoT, no satisfacen por completo los principios clave del esquema ARIS: descentralización, autonomía, largo alcance, auto-escalabilidad y confiabilidad. Por esta razón, se analizaron y adoptaron otras tecnologías más alineadas con estos objetivos. Entre ellas destacan Zigbee, Bluetooth Mesh, DTN (Delay/Disruption Tolerant Networking), CCSDS y SCPS, cuyos elementos más relevantes y adaptables han servido como base para el diseño de la red y tecnología propia de ARIS.

3.3.1. Bluetooth Mesh

Bluetooth Mesh es una extensión del protocolo Bluetooth *Low Energy*, pensada para aplicaciones IoT que requieren comunicación entre múltiples nodos distribuidos en una red robusta.

Características clave

- **Optimizado para redes densas:** Funciona bien en espacios con múltiples dispositivos como edificios inteligentes.
- **Seguridad robusta:** Integra autenticación, cifrado y gestión de claves desde el diseño.
- **Topología mallada:** Similar a Zigbee, permite redes auto-escalables y tolerantes a fallos.
- **Mensajería basada en publicación/suscripción:** Facilita la comunicación grupal eficiente.

Relevancia para ARIS

Bluetooth Mesh representa un modelo funcional de descentralización y autoescalabilidad, aunque su alcance físico es más limitado en comparación con tecnologías como LoRa. Su estructura en malla y el mecanismo de comunicación basado en publicación/-suscripción permiten que nuevos nodos se integren de forma automática, sin depender de una infraestructura central. Este comportamiento ha servido como referencia para el diseño de ARIS, especialmente en lo relacionado con la expansión y adaptación dinámica de la red.

En el caso de ARIS, aunque no se adopta exactamente la misma lógica de roles que en Bluetooth Mesh, sí se parte de su filosofía de escalabilidad descentralizada. La red mallada de ARIS permite que cada nodo se conecte y colabore con otros de forma autónoma, sin necesidad de coordinación central. Esta inspiración ha sido clave para definir cómo nuevos nodos se integran, comparten información y aseguran su lugar dentro del sistema, manteniendo así la escalabilidad sin comprometer la estabilidad ni la seguridad de la red, los pasos de autoexpansión de bluetooth mesh son los siguientes:

1. Un nuevo nodo se une a la red mediante un proceso llamado provisioning, donde recibe una dirección, claves de red y parámetros de configuración.
2. No necesita conexión directa con todos, ya que los mensajes se transmiten a través de múltiples nodos retransmisores(*relay nodes*), lo que permite expandir el alcance

sin cambiar la infraestructura existente, en cambio en ARIS se ha adaptado a enviar su información a todo los nodos mediante *broadcast*.

3. A medida que se agregan nodos, estos actúan como nuevos puntos de transmisión o suscripción, ayudando a redistribuir el tráfico, extender la cobertura y balancear la carga, sin afectar la estabilidad general de la red

3.3.2. Zigbee

Zigbee es un protocolo de comunicación inalámbrica basado en el estándar IEEE 802.15.4, diseñado específicamente para aplicaciones de bajo consumo y redes PAN. Se utiliza comúnmente en soluciones domóticas y redes de sensores debido a su eficiencia energética y facilidad de implementación.

Características clave

- **Topología en malla:** Permite crear redes *autoorganizadas* y *autorreparables*, lo que incrementa la fiabilidad y la cobertura sin depender de una infraestructura central.
- **Bajo consumo energético:** Ideal para dispositivos alimentados por batería.
- **Escalabilidad:** Soporta cientos de dispositivos conectados en una misma red.
- **Corto alcance:** Aunque es una desventaja en términos de distancia, la topología en malla compensa esta limitación extendiendo la red de forma cooperativa.

Relevancia para ARIS

Su arquitectura descentralizada y su capacidad de formar redes malladas *resilientes* lo convierten en un referente importante para la implementación de ARIS, especialmente en entornos densos o distribuidos.

ARIS adopta una arquitectura en malla similar a Zigbee, permitiendo integrar nuevos nodos fácilmente mediante difusión. Los nodos activos comparten metadatos y participan en un proceso de inyección distribuida de claves. Una vez completado, el nuevo nodo puede comenzar a operar dentro de la red.

3.4 Arquitectura DTN

DTN es una arquitectura de red diseñada para entornos donde la conectividad es intermitente o con grandes retardos, como misiones espaciales, zonas rurales o redes móviles de difícil cobertura.

Características clave

- **Almacenamiento y reenvío:** Los nodos almacenan los datos hasta que pueden reenviarlos, asegurando la entrega en redes no persistentes.
- **Independiente de la infraestructura:** Puede funcionar sin una red continua o centralizada.
- **Alta tolerancia a fallos y desconexiones:** Diseñado para operar en condiciones adversas.

Relevancia para ARIS

DTN es un modelo ideal para redes autónomas y comunicaciones de largo alcance, especialmente útil en contextos donde la conectividad es intermitente o no garantizada. Su enfoque de almacenamiento y reenvío (*store and forward*) lo convierte en una solución clave para garantizar la resiliencia y confiabilidad en entornos adversos o distribuidos.

Inspirada en este modelo, la red ARIS ha sido diseñada para funcionar incluso sin enlaces punto a punto constantes. En este esquema, algunos nodos han sido configurados para actuar como relays pasivos o activos, dependiendo de su rol en la red. Estos nodos retransmiten los *chunks* (los paquetes dentro de la red ARIS) de datos, permitiendo que la información llegue a su destino aunque el receptor no esté dentro del rango directo del emisor. Este mecanismo asegura la continuidad y entrega de mensajes incluso en condiciones de conectividad limitada. A diferencia de DTN, ARIS no soporta el enfoque de almacenamiento y reenvío (*store and forward*) sino que desde ARIS Alliance se ha pensado y acordado que dicho trabajo debería delegarse por un sistema mucho más sencillo que junto con conocimientos adquiridos del estándar SCPS se plantea, se desarrolla la idea a continuación.

3.5 Estandar CCSDS

CCSDS es un conjunto de estándares desarrollados por agencias espaciales para la comunicación y el manejo de datos en misiones espaciales.

Características clave

- **Enfoque modular:** Se puede adaptar a distintos niveles de la pila de comunicación.
- **Estándares inter-operables:** Facilitan la cooperación entre diferentes sistemas y agencias.
- **Protocolos de alto rendimiento para entornos hostiles:** Fiabilidad en transmisiones a larga distancia con altas latencias.

Relevancia para ARIS

Su enfoque robusto frente a interrupciones y su orientación hacia la interoperabilidad hacen que sus modelos sean útiles para entornos descentralizados y distribuidos.

La idea de ARIS es poder establecer un estándar que permita la interoperabilidad entre múltiples sistemas y fabricantes de manera simultánea. Esta operación se conoce como *network binding*. En este proceso, se incorporan nodos de conexión que participan activamente en dos redes diferentes, funcionando como puentes entre ellas. Estos nodos actúan como intermediarios, facilitando el envío y la recepción de datos de una red a otra, asegurando que la información pueda fluir de manera eficiente y transparente entre sistemas diversos.

El concepto de *network binding* garantiza que diferentes redes, posiblemente basadas en tecnologías, protocolos y plataformas distintas, puedan integrarse de manera fluida. Este tipo de interconexión no solo favorece la interoperabilidad de sistemas heterogéneos, sino que también optimiza la transmisión de datos, reduciendo las barreras tecnológicas y potenciando una comunicación más flexible y escalable entre las partes involucradas.

De esta forma, ARIS promueve un entorno más colaborativo y eficaz en el que los diferentes actores del ecosistema tecnológico puedan interactuar y compartir información de manera confiable y sin fricciones.

3.6 Extensión SCPS

SCPS es un conjunto de extensiones del protocolo TCP/IP, optimizadas para aplicaciones espaciales y comunicaciones de larga distancia.

Características clave

- **Optimización del protocolo TCP/IP:** Adaptado para ambientes con latencias elevadas, pérdidas frecuentes y ancho de banda limitado.
- **Fiabilidad y eficiencia:** Mejora la eficiencia de transmisión de datos en entornos críticos.
- **Adaptabilidad:** Puede integrarse en sistemas existentes con bajo impacto.

Relevancia para ARIS

Su conjunto de directrices permiten una comunicación más confiable y eficiente en sistemas autónomos, dispersos y de largo alcance, lo que refuerza los principios técnicos de ARIS.

ARIS implementa un control de errores de transmisión relativamente sencillo. A diferencia de lo que se describe en los protocolos SCPS, no utiliza ventanas de congestión ni de recepción. Sin embargo, sí implementa un sistema de ACKs similar al de TCP/IP, pero adaptado al contexto de SCPS. En este modelo, el origen envía los *chunks* de forma secuencial, y el destinatario registra el índice de cada uno. Si algún chunk falta, el destinatario envía el ACK una vez que haya registrado el último chunk. Si el origen no recibe respuesta tras enviar un chunk y se alcanza el *timeout* (aproximadamente 100 ms), el chunk se vuelve a enviar. Una vez que el origen recibe el ACK, procede a retransmitir los *chunks* que faltaban.

CAPÍTULO 4
Alternativas a WiFi para IoT distribuido

En este capítulo se abordará el análisis de las tecnologías comúnmente utilizadas en el ámbito del IoT, específicamente el WiFi y el estándar ISO/IEC 8266, y por qué, a pesar de su amplia adopción, no resultan las más adecuadas para el contexto discutido en este trabajo. A través de una evaluación detallada, se examinarán las limitaciones técnicas que presentan estas soluciones frente a alternativas más especializadas, destacando cómo estas últimas pueden ofrecer ventajas significativas en entornos con mayores demandas de rendimiento, fiabilidad y eficiencia.

4.1 WiFi como tecnología predominante en IoT

El WiFi es una de la tecnología más utilizadas en el ámbito del IoT debido a su accesibilidad, facilidad de implementación y bajo costo. WiFi, basado en el estándar IEEE 802.11, es la opción preferida en muchos entornos urbanos para la interconexión de dispositivos como electrodomésticos inteligentes, cámaras de seguridad, sistemas de automatización del hogar y dispositivos de monitoreo de salud. Su capacidad para ofrecer conectividad de alta velocidad en distancias relativamente cortas lo hace adecuado para aplicaciones donde la velocidad de transmisión es esencial.

Por otro lado, el IEEE 802.11 regula cómo se deben comunicar los dispositivos en redes WLAN, proporcionando especificaciones para la conectividad de dispositivos a través de radiofrecuencia en un área de alcance limitado. Este estándar ha sido fundamental para la creación de redes WiFi en hogares y oficinas, y se adapta bien a aplicaciones donde se necesita una conexión rápida y estable en un rango corto.

A pesar de su amplia adopción, WiFi presentan varias limitaciones cuando se aplican a entornos más exigentes, típicos del IoT. En primer lugar, WiFi tiene limitaciones de alcance, especialmente en entornos complejos como áreas rurales o interiores de edificios con múltiples obstáculos. La latencia y la congestión de la red también pueden ser un problema en entornos con muchos dispositivos conectados, lo que afecta el rendimiento de la red. Además, el alto consumo energético de los dispositivos WiFi no es ideal para aplicaciones IoT que requieren un bajo consumo de energía y una larga duración de batería.

Por otro lado, aunque el estándar IEEE 802.11 ha evolucionado para abordar algunas de estas limitaciones, como en las versiones más recientes que mejoran la eficiencia del espectro y el rendimiento en redes densas (por ejemplo, WiFi 6), aún enfrenta desafíos

cuando se trata de entornos de cobertura amplia, alta fiabilidad y bajo consumo energético, características que son cruciales para muchas aplicaciones IoT modernas.

En este capítulo, se analizarán estas limitaciones y se compararán WiFi y IEEE 802.11 con la tecnología ARIS. Estas soluciones emergentes están diseñadas para superar los desafíos de cobertura, latencia y eficiencia energética, ofreciendo soluciones más robustas y adaptables a aplicaciones de alto rendimiento, entornos distribuidos y misiones críticas.

4.2 WiFi y sus desventajas

Aunque WiFi sigue siendo una opción popular debido a su accesibilidad, hay varias razones por las que no siempre es adecuado para aplicaciones de IoT en entornos difíciles o críticos.

4.2.1. Limitaciones de alcance, latencia y consumo energético

El WiFi presenta limitaciones significativas en términos de alcance y latencia. A medida que se aumenta la distancia entre el dispositivo y el router, la señal se degrada rápidamente, lo que afecta tanto la calidad de la conexión como el rendimiento de los dispositivos conectados. Además, WiFi no está diseñado para funcionar de manera eficiente en entornos con alta densidad de dispositivos, lo que puede generar problemas de congestión. La latencia en redes WiFi también puede ser un factor limitante para aplicaciones que requieren comunicaciones en tiempo real. En cuanto al consumo energético, WiFi no es la opción más eficiente, especialmente cuando se utilizan dispositivos alimentados por baterías que deben operar durante períodos prolongados.

4.2.2. Restricciones en la capacidad de transmisión y rendimiento en entornos críticos

El estándar IEEE 802.11, que regula las tecnologías de redes WiFi, ha sido diseñado principalmente para entornos con una baja a media densidad de dispositivos, como oficinas, hogares y pequeños comercios, donde la conectividad de corta a media distancia es suficiente. Sin embargo, no está optimizado para alta densidad de dispositivos, lo cual es una limitación significativa cuando se aplica a aplicaciones del IoT en entornos más críticos o industriales.

Una de las principales restricciones del IEEE 802.11 radica en las frecuencias de operación. El estándar utiliza principalmente las bandas de 2.4 GHz y 5 GHz, las cuales, aunque populares, son susceptibles a interferencias en áreas con alta concentración de dispositivos. En entornos de alta densidad de dispositivos, como en aplicaciones IoT masivas o en áreas urbanas congestionadas, estas interferencias pueden afectar gravemente la capacidad de transmisión y el rendimiento de la red. La congestión en el espectro de radiofrecuencia limita la capacidad de los dispositivos para comunicarse de manera eficiente, lo que aumenta la latencia y reduce la fiabilidad de las transmisiones.

Además, IEEE 802.11 no está diseñado para escalar de manera eficiente en redes con muchos dispositivos conectados simultáneamente. En un escenario donde la alta densidad de dispositivos es clave, como en las ciudades inteligentes o entornos industriales complejos, WiFi puede experimentar problemas de congestión. A medida que más dispositivos se conectan a la misma red, el ancho de banda disponible se distribuye entre ellos, lo que puede generar una caída significativa en la calidad de la conexión, afectan-

do las aplicaciones que requieren alta velocidad de transmisión o conexiones estables en tiempo real.

El estándar también tiene limitaciones en cuanto a cobertura. Aunque la tecnología ha mejorado con WiFi 6, que ofrece un mejor rendimiento en redes densas, sigue siendo una opción menos eficiente en términos de alcance y cobertura en áreas extensas comparado con otras tecnologías como LoRa. Esto es especialmente importante en aplicaciones IoT que necesitan cobertura en zonas rurales o entornos distribuidos donde WiFi no puede garantizar una conexión fiable.

Por último pero no menos importante, el consumo energético de los dispositivos Wi-Fi es considerablemente alto en comparación con otras tecnologías diseñadas específicamente para IoT. Esto hace que WiFi no sea la opción más adecuada para dispositivos que requieren una larga duración de batería, como sensores remotos o dispositivos de monitoreo en áreas donde la alimentación eléctrica no es fácilmente accesible.

4.3 Beneficios de ARIS respecto a IoT

El sistema ARIS presenta una serie de ventajas significativas en comparación con tecnologías tradicionales como WiFi e IEEE 802.11, especialmente cuando se aplica a escenarios de IoT en entornos exigentes o críticos. ARIS está diseñado para abordar las limitaciones de otras tecnologías, como el alcance limitado, la congestión de red y el alto consumo energético, características comunes en redes WiFi.

Los beneficios clave de ARIS en el contexto de IoT son los siguientes:

- **LoRa: Ventajas en largo alcance, bajo consumo energético y cobertura en entornos remotos**

  Uno de los pilares fundamentales de ARIS es su uso de LoRa, una tecnología de modulación de espectro que permite comunicaciones de largo alcance con un bajo consumo energético. LoRa se destaca por su capacidad para cubrir distancias de hasta 15 km en entornos rurales y hasta varios kilómetros en entornos urbanos, lo que lo convierte en una opción ideal para aplicaciones IoT en zonas remotas, como el monitoreo ambiental, el rastreo de activos en áreas extensas, o en misiones espaciales y aeroespaciales.

  Además de su alcance, LoRa es extremadamente eficiente en términos de consumo energético, lo que permite que los dispositivos IoT funcionen durante largos períodos con baterías de baja capacidad, lo que no es posible con otras tecnologías. Esto es particularmente beneficioso en aplicaciones IoT donde los dispositivos deben ser autónomos y funcionar sin necesidad de recargar o cambiar las baterías con frecuencia.

- **Referencia a SCPS (Space Communications Protocol Specifications): Beneficios en la fiabilidad y eficiencia en entornos espaciales y distribuidos**

  ARIS incorpora tecnologías referenciadas al protocolo SCPS, que optimiza las comunicaciones en entornos con alta latencia, pérdidas de paquetes y ancho de banda limitado. SCPS, que se utiliza principalmente en aplicaciones espaciales y en redes distribuidas, mejora la fiabilidad de la transmisión de datos en entornos donde las condiciones de la red no son ideales, como en satélites, rovers o drones.

  SCPS es fundamental para garantizar la integridad de los datos incluso cuando los paquetes de información se pierden debido a la interferencia o los errores de transmisión. Además, al incorporar mecanismos de retransmisión controlada y reconocimientos (ACKs) en ARIS, el sistema asegura que los datos sean correctamente

entregados y que la comunicación siga siendo fiable y eficiente, incluso en condiciones adversas. Esto es esencial en entornos de misión crítica, donde la fiabilidad de las comunicaciones es vital.

- **Escalabilidad y adaptabilidad en redes distribuidas**

  Una de las principales ventajas de ARIS es su capacidad para escalar y adaptarse a diferentes topologías de red. A diferencia de WiFi, que depende de un punto de acceso centralizado, ARIS utiliza una arquitectura descentralizada, lo que permite que los dispositivos se conecten directamente entre sí, formando redes de autoorganización. Esto facilita la expansión de la red a medida que se agregan más dispositivos sin necesidad de una infraestructura centralizada costosa y difícil de mantener.

  Además, la adaptabilidad de ARIS permite que se ajuste dinámicamente a cambios en la red, como la adición de nuevos dispositivos o la modificación de la estructura de la red, lo que asegura que el sistema sea flexible y sostenible a largo plazo.

- **Integración con dispositivos IoT y sistemas heterogéneos (futuro)**

  Otro beneficio clave de ARIS es su capacidad para integrarse con una amplia variedad de dispositivos IoT y sistemas heterogéneos, como sensores, actuadores, rovers, satélites, drones y sistemas embebidos. Al ser compatible con plataformas como Arduino, Raspberry Pi y otros sistemas de bajo costo, ARIS permite a los desarrolladores implementar soluciones IoT de manera eficiente y escalable. Esta interoperabilidad facilita la integración de diferentes tecnologías y protocolos dentro de una red distribuida, promoviendo una comunicación fluida entre dispositivos de diferentes fabricantes.

CAPÍTULO 5
LoRa y Hardware

En este capítulo se realiza una descripción detallada de todo el *hardware* utilizado a lo largo de la investigación y el desarrollo del proyecto. Además, se explica en profundidad la función del módulo LoRa y su integración dentro del esquema general del proyecto, destacando su importancia y el nivel de relevancia que se le otorga dentro del diseño.

A continuación, se presenta la sección del esquema del proyecto que será el foco principal de este capítulo. Este capítulo aborda dos áreas clave y una conexión específica, que se detallarán a lo largo del texto. Es crucial comprender el papel de cada componente de *hardware* en el prototipo, así como sus características técnicas, ya que esta etapa es cuando surgen más desafíos y errores debido a la incursión en territorios desconocidos. Esta fase permite identificar fallos en el diseño y hacer ajustes antes de avanzar hacia el producto final, favoreciendo la evolución y el aprendizaje continuo.

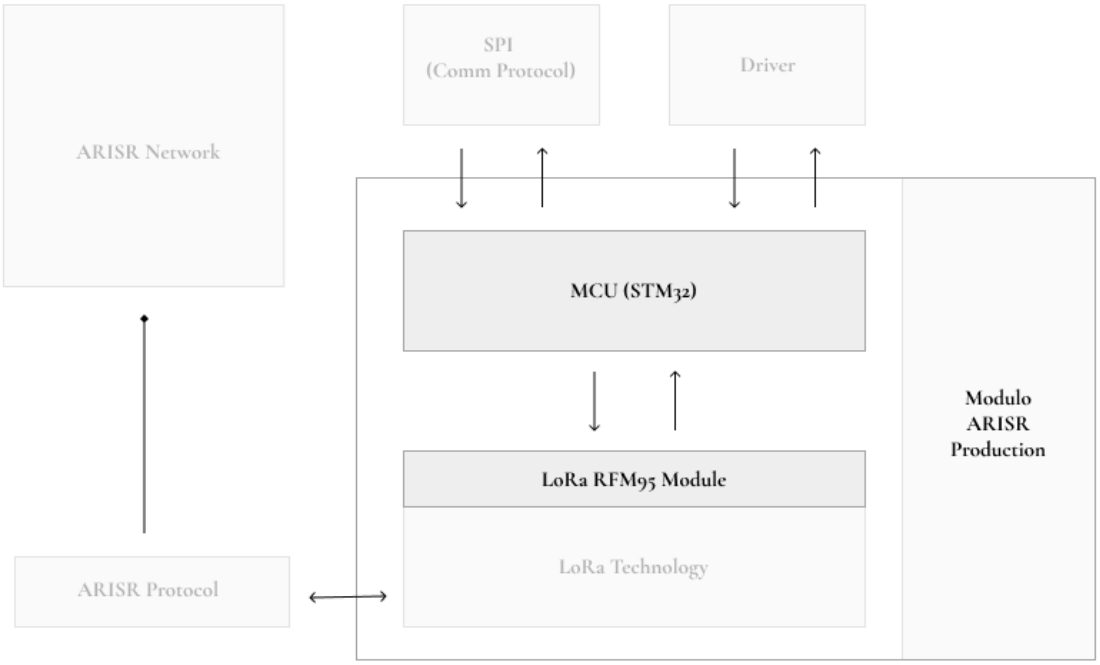

Figura 5.1: Diagrama de componentes de la capa hardware

Los dos módulos detallado en el diagrama corresponde a la Capa Base y la Capa 1 de ARISR

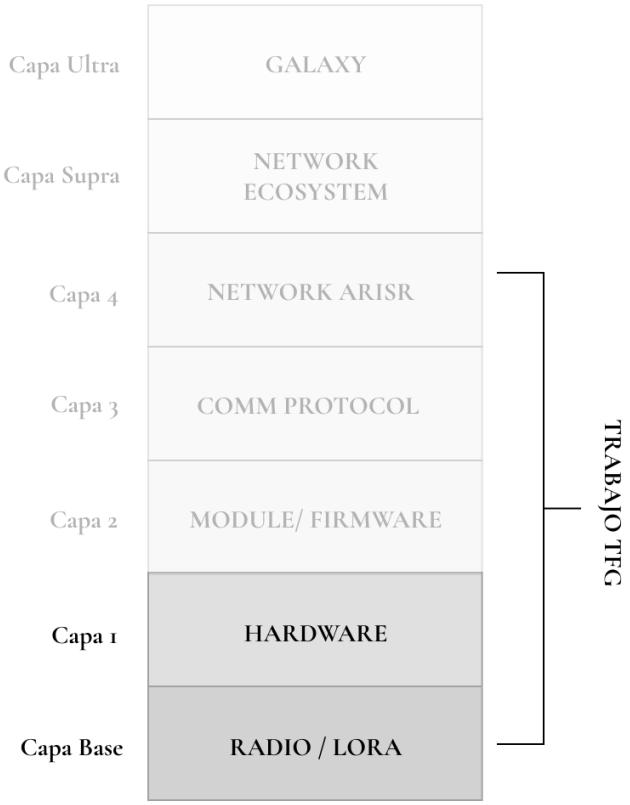

Figura 5.2: Diagrama de capas hardware

5.1 Hardware LoRa (SX1276) RFM95

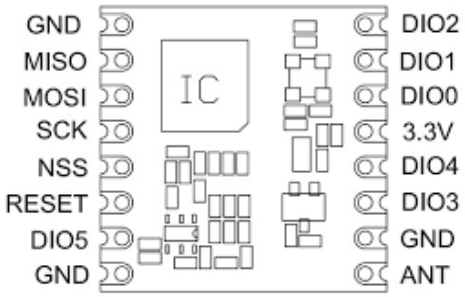

Figura 5.3: Módulo LoRa RFM95 utilizado en el prototipo.

El módulo LoRa utilizado en la maquetación del prototipo es el RFM95, que incorpora el chip SX1276 y opera en la frecuencia de 868 MHz. Este módulo se ha seleccionado debido a sus características técnicas y su capacidad para ofrecer una comunicación eficiente y de largo alcance. El RFM95 es ampliamente utilizado en aplicaciones de IoT debido a su bajo consumo energético y su capacidad para alcanzar largas distancias de transmisión, incluso en entornos con interferencias o obstáculos.

Para el prototipo, se ha optado por emplear una comunicación de frecuencia única a 868 MHz, aunque cabe destacar que el protocolo ARISR es compatible con dos bandas de frecuencia múltiple, siendo 868 MHz y 433 MHz la preferida para su uso en Europa. La elección de la frecuencia de 868 MHz está en línea con las normativas locales y las mejores prácticas para asegurar una transmisión estable y sin interferencias, aprovechando las ventajas del espectro de frecuencias permitido en esta región. Su uso debe estar regulado conforme a la zona en la que se opera. Por esta razón, ARIS Alliance contempla la posibilidad de implementar en el futuro un radio estándar para uso admitido internacionalmente, tanto en aplicaciones IoT como en misiones espaciales. Por ahora se mantiene LoRa como tecnología de la Capa Base como alternativa.

Para el diseño final del módulo ARISR, se prescindirá del uso del RFM95. En su lugar, se desarrollará una interfaz directa que permita la comunicación entre el MCU central y el chip SX1276.

Por preferencia de diseño, se empleará el protocolo SPI para la comunicación con todos los periféricos. En este esquema, el módulo LoRa se conectará al MCU a través de la interfaz SPI cero, mientras que los demás dispositivos utilizarán las siguientes interfaces SPI disponibles de forma sucesiva. En el capítulo siguiente se detallará en profundidad el funcionamiento de la comunicación serial y la configuración específica de cada periférico.

Se añade que ARISR, para poder diferenciar la clave de sincronización usada en LoRa, establece `0xA5` como valor de *SyncWord*. De esta forma se crea un filtrado automático de LoRa por parte de la capa de radio. Con esto podemos filtrar ondas con chirps usados por LoRaWAN o redes IoT privadas, comúnmente conocidos por usar: `0x34`, `0x12` y `0xF3`.

Aunque ARISR ya tenga un método de filtrado por *Payload*, al igual que LoRaWAN, es conveniente tener como convenio un filtrado extra para aislar los datos que se envían de distintas redes y tecnologías. Luego, internamente, el filtrado de ARISR discrimina por diferentes redes privadas. Esto se puede ver en la sección 10 con más detalle.

5.2 Hardware MCU (STM32F4) STM32F1

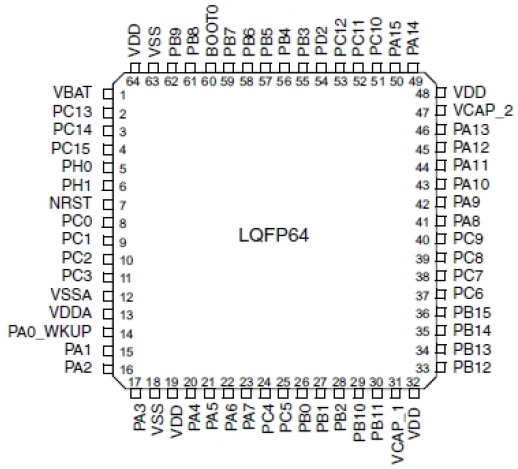

Figura 5.4: MCU STM32F4

En el desarrollo del sistema ARISR se han considerado dos familias de microcontroladores familia STM32, pertenecientes a STMicroelectronics: la serie STM32F1 para la etapa de prototipado y la serie STM32F4 para el diseño final de producción.

Durante la fase de pruebas y validación, se empleará el STM32F1, una línea ampliamente utilizada en el desarrollo de sistemas embebidos debido a su bajo consumo, buen rendimiento y facilidad de implementación. Su elección responde a la necesidad de contar con una plataforma accesible, robusta y bien documentada, ideal para iteraciones rápidas en el laboratorio.

Para la producción del módulo final, se utilizará un microcontrolador de la serie STM32F4, el cual ofrece un mayor rendimiento gracias a su núcleo ARM Cortex-M4, capacidades avanzadas de procesamiento digital de señales (DSP) y mayor disponibilidad de periféricos integrados. Esta transición garantiza un funcionamiento más eficiente y es-

calable del sistema ARISR, permitiendo soportar operaciones más complejas y exigentes en aplicaciones reales.

Ambas versiones comparten compatibilidad con el protocolo SPI y otros estándares de comunicación necesarios para interactuar con los periféricos del sistema, lo que facilita la migración del código y el diseño del *firmare* desde el prototipo hasta el producto final.

La elección de microcontroladores STM32 frente a otras opciones disponibles en el mercado se fundamenta en una combinación de factores clave que aportan ventajas tanto en el desarrollo como en la implementación final del sistema:

- **Amplia familia y escalabilidad:** ST ofrece una gran variedad de microcontroladores con diferentes capacidades (desde la serie F0 hasta la H7), lo que permite escalar el diseño sin cambiar de ecosistema. Esto facilita la migración de prototipos simples a productos comerciales más complejos.
- **Arquitectura ARM Cortex-M:** Todos los STM32 están basados en núcleos ARM Cortex-M, lo que garantiza compatibilidad con herramientas estándar de desarrollo, buena eficiencia energética y soporte a largo plazo.
- **Buen equilibrio rendimiento/consumo:** Especialmente en la serie F4, se logra un excelente compromiso entre potencia de procesamiento, consumo energético y capacidad de integración periférica.
- **Soporte extenso y comunidad activa:** Existe una gran cantidad de documentación oficial, bibliotecas (HAL/LL, CMSIS), foros y recursos *open source*, lo que reduce significativamente la curva de aprendizaje y facilita la resolución de problemas durante el desarrollo.
- **Compatibilidad con herramientas de desarrollo gratuitas:** ST proporciona herramientas como *STM32CubeMX* y *STM32CubeIDE*, que permiten configurar periféricos, generar código base y compilar el *firmare* sin necesidad de licencias pagas.
- **Integración de periféricos avanzados:** Los STM32 incorporan múltiples interfaces de comunicación (SPI, I2C, UART, CAN, USB, etc.), ADCs y DACs de alta resolución, temporizadores versátiles y soporte para RTC, entre otros. Esto reduce la necesidad de componentes externos y simplifica el diseño del hardware.
- **Disponibilidad y confiabilidad:** STM32 es una línea consolidada y ampliamente distribuida, lo que asegura disponibilidad a largo plazo, trazabilidad de componentes y confianza industrial.

5.3 Función del MCU

El MCU de ARISR en prototipo será un placa de desarrollo STM32F1, en la que ya satisface las necesidades que se requieren para poder trabajar de forma aislada. Dicha placa se harán las pruebas de *firmare* con conexión SPI al módulo RFM95 para validar todo lo desarrollado antes de proceder al diseño.

Durante esta etapa, el MCU se conectará al módulo RFM95 mediante la interfaz SPI, utilizando específicamente las interfaces SPI cero y SPI uno para establecer comunicación con dos módulos LoRa independientes, integrados dentro del sistema ARISR. Esta configuración permitirá realizar pruebas de enlace bidireccional y evaluar el comportamiento del protocolo en condiciones reales.

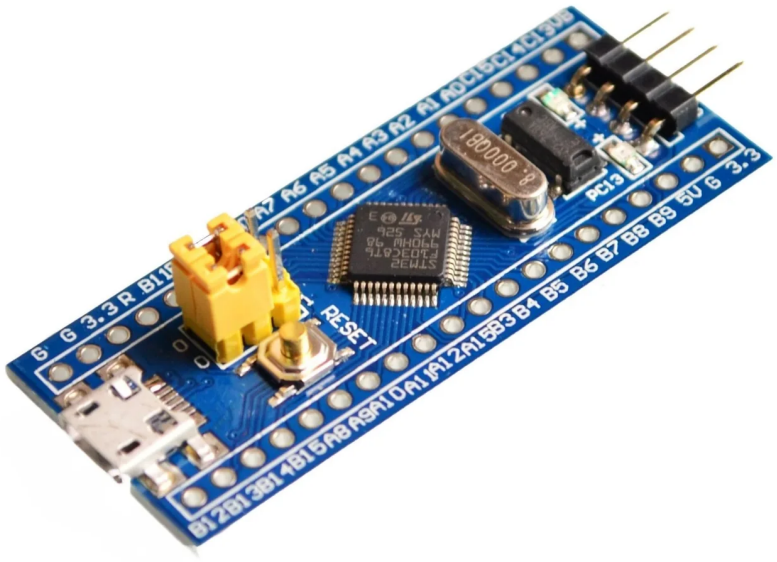

Figura 5.5: MCU STM32F1

El *firmare* del sistema ARISR, desarrollado sobre el MCU, será responsable de gestionar todos los periféricos de comunicación y el flujo de datos. En particular:

1. El MCU recibirá los datos proporcionados por el usuario a través de una interfaz SPI dedicada.

2. Los datos serán procesados y empaquetados utilizando la librería personalizada **lib_protoARISR_c**, que estructura la información de acuerdo con el protocolo, descrito en el capítulo siguiente.

3. Una vez empaquetada, la información será transmitida por radio mediante uno de los módulos LoRa, preferentemente por la interfaz cero, a menos que el usuario especifique otra salida.

El MCU simultáneamente operará en modo de recepción asíncrona, escuchando constantemente los paquetes entrantes por LoRa. Al recibir datos, estos serán:

1. Filtrados según su pertinencia (es decir, si están dirigidos al nodo local), según se describe en el Capítulo X.

2. Si el paquete es válido, el MCU lo entregará de inmediato a la salida del usuario mediante un sistema de interrupciones, garantizando baja latencia y respuesta rápida.

Es importante destacar que ARISR no implementa almacenamiento intermedio de paquetes cuando el usuario no puede procesar la interrupción de salida. Por lo tanto, es responsabilidad del usuario o del sistema receptor mantener un buffer de recepción adecuado para no perder datos si se encuentra ocupado o fuera de línea.

Esta arquitectura liviana, basada en la filosofía de procesamiento en tiempo real y sin retención local, se alinea con los principios de CCSDS sobre eficiencia energética y simplicidad operativa en aplicaciones IoT o escenarios espaciales.

A continuación se adjunta un esquema de procesos que hace el flujo de datos que circulan dentro del módulo ARISR:

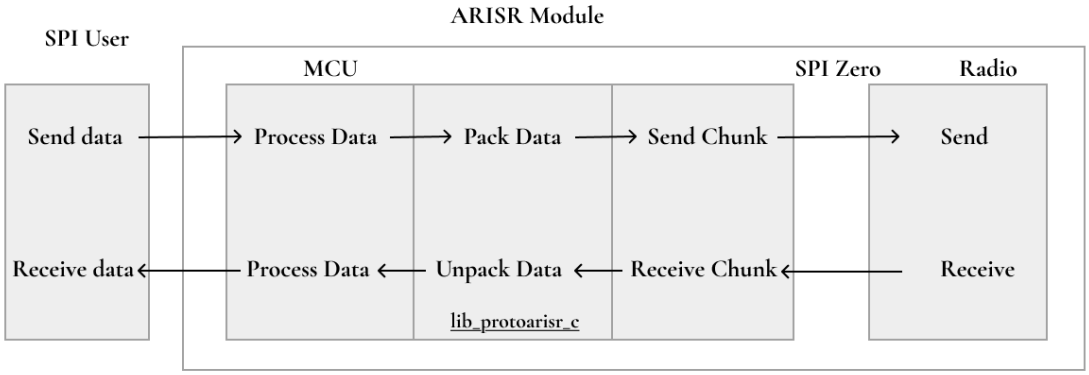

Figura 5.6: Diagrama de flujo de datos en el módulo ARISR.

5.4 Otros Hardware

Los hardwares que utilizaremos para hacer las pruebas del prototipo, tanto del diseño cómo del *firmare* desarrollado serán las siguientes:

- **ESP32 Placa:** Para hacer pruebas rápidas, de forma sencilla y con Arduino IDE.
- **STM32F1 Placa:** Para testear el *firmare* desarrollado de ARISR.
- **RF96 Módulo:** Donde contiene todo los componentes que gestionan LoRa.
- **ST-Link V2:** Sirve para programador o cargar el *firmare* al bootloader.

Durante el desarrollo del prototipo, la mayor parte del *hardware* necesario está integrado tanto en la placa como en el módulo. Sin embargo, para el diseño final de la placa, se ha requerido la adición de algunos componentes adicionales, los cuales se detallan a continuación:

- **Placa base y diseño del PCB:** La correcta disposición de las rutas de conexión y la implementación de un plano de tierra adecuado son esenciales para reducir interferencias y asegurar el funcionamiento estable de los componentes, especialmente las señales de alta frecuencia como SPI y las señales RF. Las rutas deben ser lo más cortas y anchas posible, y el diseño debe priorizar la separación de las señales de alta velocidad para minimizar el ruido.
- **Protección ESD:** Se han añadido diodos de protección ESD en las líneas de datos críticas, como MOSI, MISO, SCK y NSS, para proteger los circuitos de posibles descargas electrostáticas que podrían dañar los componentes sensibles.

- **Osciladores y cristales:** Para el SX1276, se ha incluido un cristal de 32 MHz para la sincronización de la señal de radiofrecuencia. El STM32F1 también requiere un oscilador adecuado (por lo general de 8 MHz o 12 MHz) para su funcionamiento correcto. Estos componentes son cruciales para mantener la estabilidad y la precisión en las comunicaciones.
- **Conexión USB para PC:** Aunque el módulo funciona a base comunicación SPI para exterior, se ha incluido un modelo extra para comunicación serial USB. Para facilitar la comunicación entre el STM32F1 y el PC, se ha integrado un convertidor USB a serie, basado en un chip como FTDI o CH340. Esto permite establecer una conexión directa con el PC para la programación y la comunicación de datos.
- **Filtro de alimentación:** Se ha incorporado un filtro de alimentación para garantizar que la fuente de energía sea estable y limpia, minimizando los ruidos que puedan afectar al rendimiento del SX1276, especialmente durante las transmisiones de RF.
- **Condensadores y resistencias:** Se han añadido condensadores de desacoplo (0.1 uF y 10 uF) cerca de los componentes clave para reducir el ruido eléctrico y estabilizar las señales de alimentación. Además, se incluyen resistencias para configurar correctamente los pines de control, como NSS/CS (*Chip Select*) y RESET, y también para garantizar el correcto funcionamiento de las señales de interrupción.
- **Antena:** Para la comunicación inalámbrica, se ha integrado un conector U.FL para una antena externa. Esta antena es fundamental para asegurar el alcance y la calidad de la señal de comunicación LoRa.

CAPÍTULO 6
Primeros pasos

Antes de comenzar con el desarrollo y diseño de las conexiones, se ha establecido una guía visual para facilitar la identificación rápida de los pines del módulo RF95. Esto evita la necesidad de consultar constantemente el *schematic* del fabricante y agiliza el proceso de prototipado.

En este capítulo se hablará todo lo relacionado a los primero que se ha hecho antes de comenzar con el trabajo.

6.1 Pines del módulo RF95

Para simplificar las conexiones, se soldaron los pines del módulo RF95 con cables de colores, asignando un color específico a cada función:

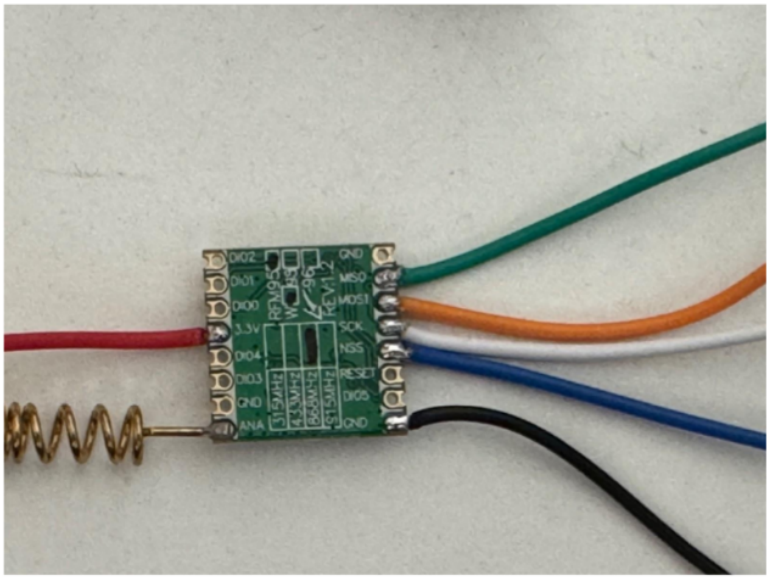

Figura 6.1: Ejemplo RFM95 con pines soldados y codificados por colores.

Color	Pin/Función	Descripción
Rojo	3.3V	Alimentación (3.3V)
Verde	MISO	Entrada de datos (LoRa → MCU)
Naranja	MOSI	Salida de datos (MCU → LoRa)
Blanco	SCK	Señal de reloj SPI
Azul	NSS (CS)	Chip Select (activación del módulo)
Negro	GND	Tierra (Ground)

Tabla 6.1: Conexiones SPI entre el MCU y el módulo LoRa

6.2 Pruebas iniciales y Durante el desarrollo

Antes de implementar el sistema completo, se adoptó una metodología de desarrollo incremental mediante pruebas modulares en Arduino IDE utilizando ESP32 donde consistía en crear algunas porciones de código de prueba para poder validar ciertos desarrollos sin tener que partir de cero cuando se necesitaba. A continuación, se detallan algunas pruebas creadas para ser usadas durante el desarrollo.

6.2.1. Prototipo de prueba

Primero de todo se creó un pequeño circuito compuesto por ESP32 y RF95 para cargar los códigos de prueba. Este circuito solo se usa para probar el módulo ARISR durante el desarrollo y verificar el correcto funcionamiento del flujo. Aunque se usa ESP32 también se ha usado un Arduino Nano durante el desarrollo ya que empezó a fallar la corriente.

Las conexiones del RF95 al microcontrolador son los siguientes:

```
/*
Board          MOSI        MISO        SCK         SS/CS (LoRa)
Arduino Uno    11          12          13          10 (or custom)
ESP32          23          19          18          5 (or custom)
ESP8266        D7 (13)     D6 (12)     D5 (14)     D8 (GPIO15)
*/

#include <SPI.h>
#include <LoRa.h>  // LoRa communication library

// LoRa module pins (adjust based on your hardware)
#define LORA_SS    5   // Chip Select (CS) pin
#define LORA_RST   14   // Reset pin (optional)
#define LORA_DIO0  2   // Interrupt pin (optional)
```

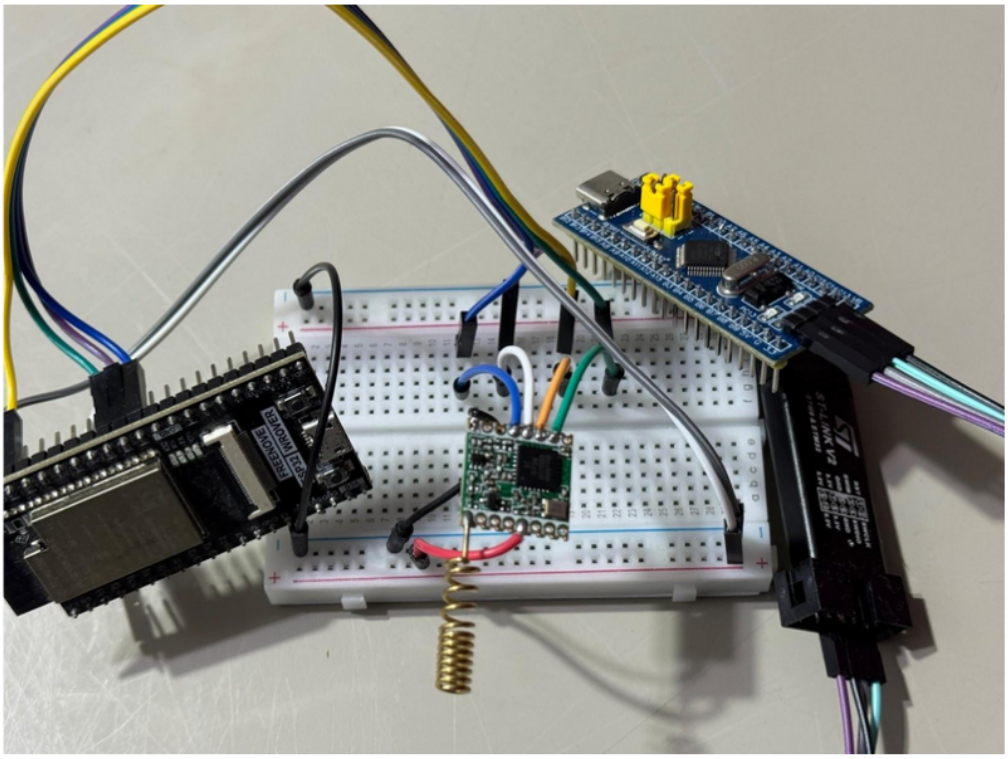

Figura 6.2: Circuito de prueba con ESP32 y módulo RF95.

6.2.2. Verificación de la comunicación

Las primeras pruebas se centraron en verificar la comunicación básica entre módulos LoRa. Se implementaron programas simples de transmisión y recepción para comprobar que los mensajes se enviaban y recibían correctamente. Estas pruebas iniciales utilizaban mensajes cortos como "Hello"para confirmar que la conexión se establecía sin errores.

Con esto nos facilita hacer pruebas íntegras del *firmare* para verificar sí el SPI que hemos creado ha sido correcto y se hacen los envíos de datos correspondiente.

Verificación de recepción:

Con esta prueba nos aseguramos que el *firmare* o el LoRa de recepción esté recibiendo datos. A su vez podemos modificar el *firmare* para que verifique sí se está recibiendo el texto “Hello” para poder ser enviada al usuario antes de aplicar la librería de protocolo ARISR

```
/*
Board          MOSI        MISO        SCK         SS/CS (LoRa)
Arduino Uno    11          12          13          10 (or custom)
ESP32          23          19          18          5 (or custom)
ESP8266        D7 (13)     D6 (12)     D5 (14)     D8 (GPIO15)
*/

#include <SPI.h>
#include <LoRa.h>
#define LORA_SS      5 // Chip Select (CS) pin
#define LORA_RST    14 // Reset pin (optional)
#define LORA_DIO0    2 // Interrupt pin (optional)
#define LED_BUILTIN 2
```

```
14
15 int counter = 0;
16
17 void setup()
18 {
19     Serial.begin(115200);
20     while (!Serial);
21
22     Serial.println("Initializing LoRa...");
23
24     pinMode(LED_BUILTIN, OUTPUT);
25
26     LoRa.setPins(LORA_SS, LORA_RST, LORA_DIO0);
27     Serial.println("Before LoRa.begin");
28     while (!LoRa.begin(868E6))
29     {
30         Serial.println(".");
31         delay(500);
32     }
33
34     Serial.println("LoRa initialized successfully!");
35 }
36
37 void loop()
38 {
39     Serial.print("Sending message: Hello ");
40     Serial.println(counter);
41     digitalWrite(LED_BUILTIN, HIGH);
42     delay(5000);
43
44     LoRa.beginPacket();  // Start packet
45     LoRa.print("Hello "); // Write payload
46     LoRa.print(counter);
47     LoRa.endPacket(true);     // Send packet
48
49     digitalWrite(LED_BUILTIN, LOW);
50     counter++;
51     delay(5000);
52 }
```

Verificación de envío:

Al igual que el código anterior también se ha creado un script de código para hacer la pruebas de envío del *firmare* de ARISR.

```
1  #include <SPI.h>
2  #include <LoRa.h>    // LoRa communication library
3  #define LORA_SS 5    // Chip Select (CS) pin
4  #define LORA_RST 14 // Reset pin (optional)
5  #define LORA_DIO0 2 // Interrupt pin (optional)
6  #define LED_BUILTIN 2
7
8  int counter = 0;
9
10 void setup()
11 {
12     Serial.begin(115200);
13     while (!Serial);
```

```
14
15    Serial.println("Initializing LoRa...");
16
17    pinMode(LED_BUILTIN, OUTPUT);
18
19    LoRa.setPins(LORA_SS, LORA_RST, LORA_DIO0);
20    Serial.println("Before LoRa.begin");
21    while (!LoRa.begin(868E6))
22    {
23        Serial.println(".");
24        delay(500);
25    }
26
27    Serial.println("LoRa initialized successfully!");
28 }
29
30
31 void loop() {
32  int packetSize = LoRa.parsePacket();
33  if (packetSize) {
34    digitalWrite(LED_BUILTIN, HIGH);
35    Serial.print("Received packet '");
36
37    while (LoRa.available()) {
38      String LoRaData = LoRa.readString();
39      Serial.print(LoRaData);
40    }
41
42    Serial.print("' with RSSI ");
43    Serial.println(LoRa.packetRssi());
44    delay(500);
45    digitalWrite(LED_BUILTIN, LOW);
46  }
47 }
```

Verificación del protocolo

Para verificar la correcta recepción del módulo ARISR se han creado unas pruebas que envían un dato de forma estática en un escenario de una red existente. El código es el siguiente:

```
1  void setup()
2  {
3      pinMode(LED_BUILTIN, OUTPUT);
4      LoRa.setPins(LORA_SS, LORA_RST, LORA_DIO0);
5
6      while (!LoRa.begin(868E6))
7      {
8          Serial.println(".");
9          delay(500);
10     }
11     LoRa.setSyncWord(0xA5); // ARISR Use SyncWord 0xA5
12 }
13
14
15 void loop()
16 {
17     dumpText(ARISR_MSG_RAW, sizeof(ARISR_MSG_RAW));
```

```
    digitalWrite(LED_BUILTIN, HIGH);
    delay(5000);

    LoRa.beginPacket();
    LoRa.write(ARISR_MSG_RAW, sizeof(ARISR_MSG_RAW));
    LoRa.endPacket(true);

    digitalWrite(LED_BUILTIN, LOW);
    delay(5000);
}
```

El contenido es el siguiente:

```
{
    .id = { 0x00, 0x11, 0x22, 0x33 },
    .aris = { 0x41, 0x52, 0x49, 0x53 },
    .ctrl = {
        .version = 1,
        .destinations = 15,
        .option = 0,
        .from = 0,
        .sequence = 1,
        .retry = 0,
        .more_data = 1,
        .identifier = 110,
        .more_header = 1
    },
    .origin = { 0x00, 0x1A, 0x2B, 0x3C, 0x4D, 0x5E },
    .destinationA = { 0xFA, 0x16, 0x3E, 0x2F, 0xEC, 0xA8 },
    .ctrl2 = {
        .data_length = 64
    },
    .feature = 0,
    .neg_answer = 0,
    .freq_switch = 0,
    .destinationsB = {
        { 0x00, 0x1A, 0x2B, 0x3C, 0x4D, 0x5E },
        { 0x00, 0x1B, 0x63, 0x84, 0x45, 0xE6 },
        { 0x00, 0x1C, 0x4F, 0x7D, 0x58, 0xA3 },
        { 0x00, 0x1D, 0x5A, 0x9E, 0x62, 0xB4 },
        { 0x00, 0x1E, 0x6B, 0xAF, 0x71, 0xC5 },
        { 0x00, 0x1F, 0x7C, 0xB0, 0x82, 0xD6 },
        { 0x00, 0x20, 0x8D, 0xC1, 0x93, 0xE7 },
        { 0x00, 0x21, 0x9E, 0xD2, 0xA4, 0xF8 },
        { 0x00, 0x22, 0xAF, 0xE3, 0xB5, 0x09 },
        { 0x00, 0x23, 0xB0, 0xF4, 0xC6, 0x1A },
        { 0x00, 0x24, 0xC1, 0x05, 0xD7, 0x2B },
        { 0x00, 0x25, 0xD2, 0x16, 0xE8, 0x3C },
        { 0x00, 0x26, 0xE3, 0x27, 0xF9, 0x4D },
        { 0x00, 0x27, 0xF4, 0x38, 0x0A, 0x5E },
        { 0x00, 0x28, 0x05, 0x49, 0x1B, 0x6F }
    },
    .data = {
        0x00, 0x11, 0x22, 0x33,
        0x40, 0x51, 0x48, 0x52,
        0x10, 0x20, 0x05, 0xDD,
        0x00, 0x1A, 0x2B, 0x3C, 0x4D, 0x5E,
        0xFA, 0x16, 0x3E, 0x2F, 0xEC, 0xA8,
        0x00, 0x1A, 0x2B, 0x3C, 0x4D, 0x5E,
```

```
47          0x00, 0x1B, 0x63, 0x84, 0x45, 0xE6,
48          0x02, 0x00, 0x00, 0x00,
49          0x4A, 0xCA,
50          0x06, 0x4F, 0x78, 0x11, 0x9D, 0x45,
51          0xAA, 0x4F, 0x9D, 0x91, 0x8D, 0xDA,
52          0x90, 0xD5, 0xDE, 0x8F,
53          0x98, 0x2E,
54          0x00, 0x11, 0x22, 0x33
55      }
56  };
```

CAPÍTULO 7
Comunicación Serial

En este capítulo, se aborda la comunicación entre los módulos, detallando los tres puntos clave donde es necesario establecer conexiones entre las diferentes partes, así como los protocolos de comunicación empleados en cada caso. Se explica el tipo de conexión utilizada y el protocolo específico para cada interacción. Las áreas donde se establece esta comunicación están destacadas en rojo en el diagrama que se presenta a continuación.

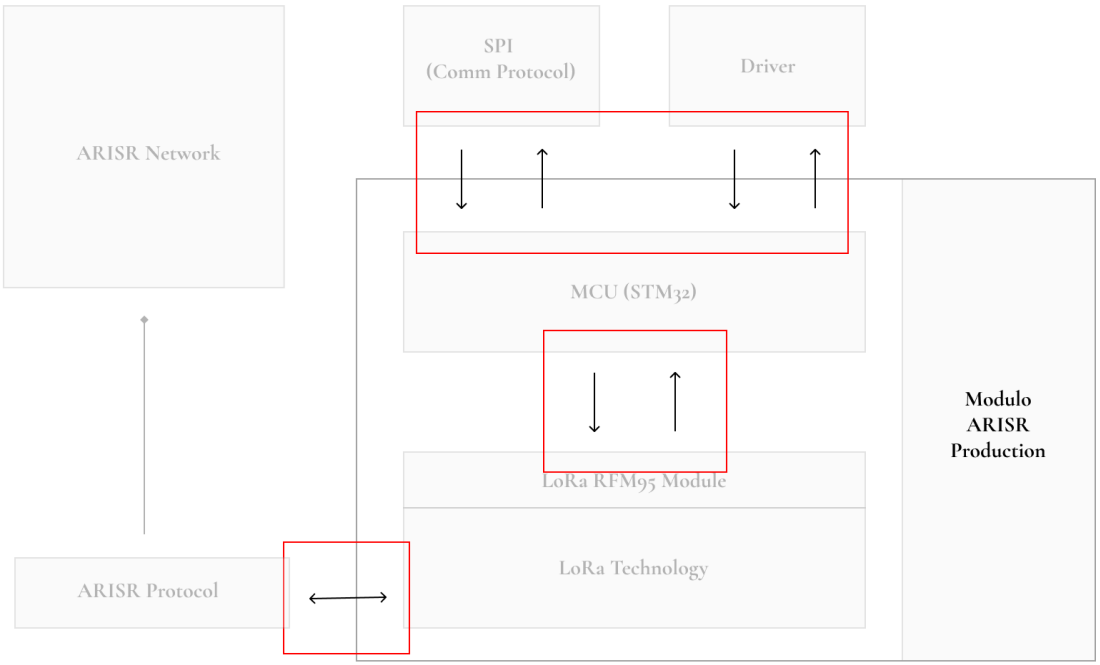

Figura 7.1: Diagrama de comunicación entre módulos en ARISR.

7.1 SPI LoRa

La comunicación entre el MCU y el módulo LoRa se realiza a través del bus SPI. De esta forma esta interfaz permite controlar el módulo LoRa mediante lectura y escritura directa en registros internos del chip.

El protocolo de comunicación no utiliza estructuras complejas ni marcos de datos estándar como UART con cabeceras. En su lugar:

- Se accede directamente a registros del chip LoRa utilizando direcciones predefinidas en la hoja de datos (*datasheet*), que puede consultarse en el Anexo X.
- Cada operación de lectura o escritura mediante la interfaz SPI consiste en un comando de 1 byte (dirección del registro) seguido de uno o más bytes de datos.
- El bit más significativo (MSB) del primer byte indica si se trata de una operación de lectura o escritura:
  - `0xxxxxxx` → lectura del registro `0xXX`
  - `1xxxxxxx` → escritura en el registro `0xXX`

Por tanto, sí se sigue un formato basado en registros, y todos los datos (inicialización, transmisión y recepción) se gestionan manipulando directamente dichos registros.

Un flujo completo de comunicación puede ser el siguiente:

1. Inicializar los GPIOs y la interfaz SPI.
2. Resetear el chip LoRa.
3. Leer el registro `RegVersion (0x42)` para confirmar la comunicación.
4. Poner el chip en modo LoRa + *Sleep* mediante el registro `RegOpMode`.
5. Configurar parámetros como frecuencia, potencia de transmisión, ancho de banda, entre otros.
6. Escribir los datos en el FIFO y cambiar el modo del dispositivo a *TX*.
7. Esperar la confirmación de transmisión.
8. Cambiar el dispositivo a modo *RX* y leer los datos del FIFO cuando se reciban.

Un ejemplo de código se muestra a continuación:

```
#include <SPI.h>
// Pines
#define NSS    10  // Chip select
#define RESET   9
#define DIO0    2

void lora_write_reg(uint8_t addr, uint8_t value) {
  digitalWrite(NSS, LOW);
  SPI.transfer(addr | 0x80);  // MSB=1 -> escritura
  SPI.transfer(value);
  digitalWrite(NSS, HIGH);
}

uint8_t lora_read_reg(uint8_t addr) {
  digitalWrite(NSS, LOW);
  SPI.transfer(addr & 0x7F);  // MSB=0 -> lectura
  uint8_t val = SPI.transfer(0x00);
  digitalWrite(NSS, HIGH);
  return val;
}

// Reset hardware
void lora_reset() {
```

```
  digitalWrite(RESET, LOW);
  delay(10);
  digitalWrite(RESET, HIGH);
  delay(10);
}

// Inicializacion basica del LoRa
bool lora_begin() {
  lora_reset();
  if (lora_read_reg(0x42) != 0x12) return false;  // RegVersion (SX1276)
  lora_write_reg(0x01, 0x80);  // Sleep + LoRa
  delay(10);

  // Frecuencia 868 MHz si acepta multi-frecuencia
  uint64_t frf = ((uint64_t)868E6 << 19) / 32000000;
  lora_write_reg(0x06, (frf >> 16) & 0xFF);
  lora_write_reg(0x07, (frf >> 8) & 0xFF);
  lora_write_reg(0x08, frf & 0xFF);

  // Potencia maxima
  lora_write_reg(0x09, 0xFF);

  // Configuracion basica de modulacion
  lora_write_reg(0x1D, 0x72); // BW 125kHz, CR 4/5
  lora_write_reg(0x1E, 0x74); // SF7, CRC ON
  lora_write_reg(0x26, 0x04); // Optimizacion

  // Standby
  lora_write_reg(0x01, 0x81);
  return true;
}

// Transmision simple
void lora_send(uint8_t *data, uint8_t len) {
  lora_write_reg(0x0E, 0x00); // FIFO TX base
  lora_write_reg(0x0D, 0x00); // FIFO puntero

  for (uint8_t i = 0; i < len; i++) {
    lora_write_reg(0x00, data[i]); // FIFO
  }

  lora_write_reg(0x22, len);     // Longitud
  lora_write_reg(0x01, 0x83);    // Modo TX

  while ((lora_read_reg(0x12) & 0x08) == 0); // Esperar TxDone
  lora_write_reg(0x12, 0x08);  // Limpiar bandera
}
```

Ejemplo: Para escribir 0x81 en RegOpMode (0x01):

```
SPI.transfer(0x81);  // 0x01 + MSB=1
SPI.transfer(0x81);  // Modo standby + LoRa
```

Para leer RegOpMode:

```
SPI.transfer(0x01);                // 0x01 + MSB=0
uint8_t val = SPI.transfer(0x00);  // Leer valor
```

Y un diagrama de flujo simplificado de las acciones sería el siguiente:

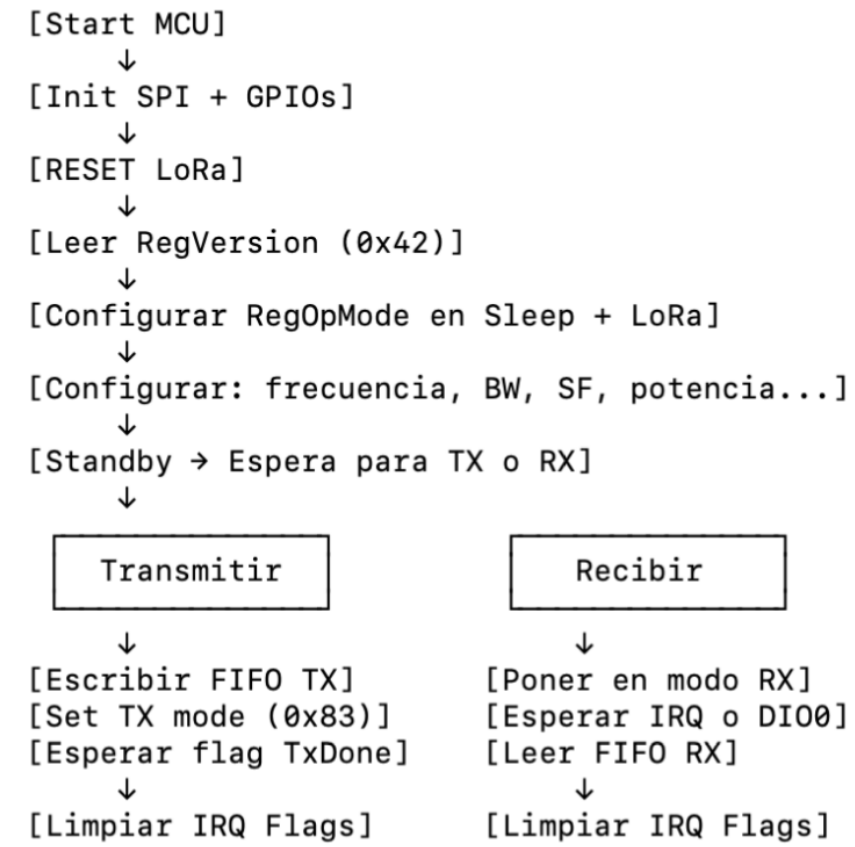

Figura 7.2: Diagrama de flujo de comunicación SPI con el módulo LoRa.

Para no estar haciendo polling constante, se puede usar el pin DIO0 como interrupción externa. Este pin se pone en alto cuando ocurre un evento, como:

- Transmisión completada (TxDone)
- Recepción de un paquete (RxDone)

Cómo activarlo de forma teorica:

- Configurás RegOpMode en modo RX continuo (0x85)
- Activás la interrupción externa en tu MCU sobre DIO0 (ej: INT0 en AVR, EXTI en STM32)
- Cuando DIO0 se activa, consultás los IRQFlags (0x12) para saber qué pasó

Y a continuación un ejemplo de pseudo-código:

```
void dio0_isr() {
  uint8_t irq = lora_read_reg(0x12);

  if (irq & 0x40) {  // RxDone
    uint8_t len = lora_read_reg(0x13); // RegRxNbBytes
```

```
    lora_write_reg(0x0D, lora_read_reg(0x10)); // puntero FIFO RX
    for (int i = 0; i < len; i++) {
      buffer[i] = lora_read_reg(0x00);
    }
    lora_write_reg(0x12, 0xFF); // limpiar todas las flags
  }

  if (irq & 0x08) { // TxDone
    lora_write_reg(0x12, 0xFF);
  }
}
```

7.1.1. Flags importantes (registro 0x12 – IRQFlags)

Bit	Nombre	Descripción
7	RxTimeout	Timeout sin recibir datos
6	RxDone	Paquete recibido correctamente
5	PayloadCrcError	Error de CRC en recepción
3	TxDone	Paquete transmitido
2	CadDone	Fin de detección de canal
0	RxValidHeader	Cabecera válida recibida

Tabla 7.1: Bits del registro de interrupciones

7.2 SPI ARISR

Para transmitir contenido mediante ARISR, el usuario se comunica con el módulo ARISR a través del bus SPI. En caso de requerir la conexión a un dispositivo externo (por ejemplo, un PC), es necesario utilizar un conversor de serie a USB. No obstante, el protocolo de comunicación basado en SPI y el formato de datos se mantienen sin cambios.

Las recepciones se tratan por interrupción, en todo caso se optará una estructura similar al LoRa de forma asincrónica

CAPÍTULO 8
Librerías y Open Source

En este capítulo se presenta de forma detallada toda la información relacionada con las librerías utilizadas o desarrolladas a lo largo del proyecto. También se incluye una descripción de las plantillas, componentes o fragmentos de código obtenidos de proyectos Open Source, especificando su procedencia, licencias y condiciones de uso.

Asimismo, se proporciona información relevante sobre el repositorio del proyecto ARISR, incluyendo su estructura, licenciamiento y consideraciones relacionadas con su distribución y reutilización. Este análisis garantiza la correcta atribución de los recursos utilizados y el cumplimiento de los términos legales establecidos por sus respectivos autores.

8.1 Librerías usadas

Durante el desarrollo del proyecto se ha optado por emplear únicamente las librerías básicas incluidas en el entorno de desarrollo Arduino IDE para el código de las pruebas del firmware. Las librerías utilizadas son las siguientes:

- **SPI.h**: Esta librería permite la comunicación mediante el protocolo SPI, fundamental para la transmisión de datos entre el microcontrolador y el módulo LoRa.

- **LoRa.h**: Proporciona una interfaz sencilla para la configuración y gestión de la comunicación a través del protocolo LoRa (*Long Range*). Esta librería facilita el envío y la recepción de paquetes de datos mediante módulos compatibles con la tecnología LoRa.

El uso exclusivo de estas librerías básicas garantiza la simplicidad del sistema, evitando dependencias externas innecesarias y favoreciendo la comprensión y replicabilidad del proyecto por parte de otros desarrolladores o investigadores. De está forma también se ahorra el tiempo de construir un código que se pueda portar y mantener por externos.

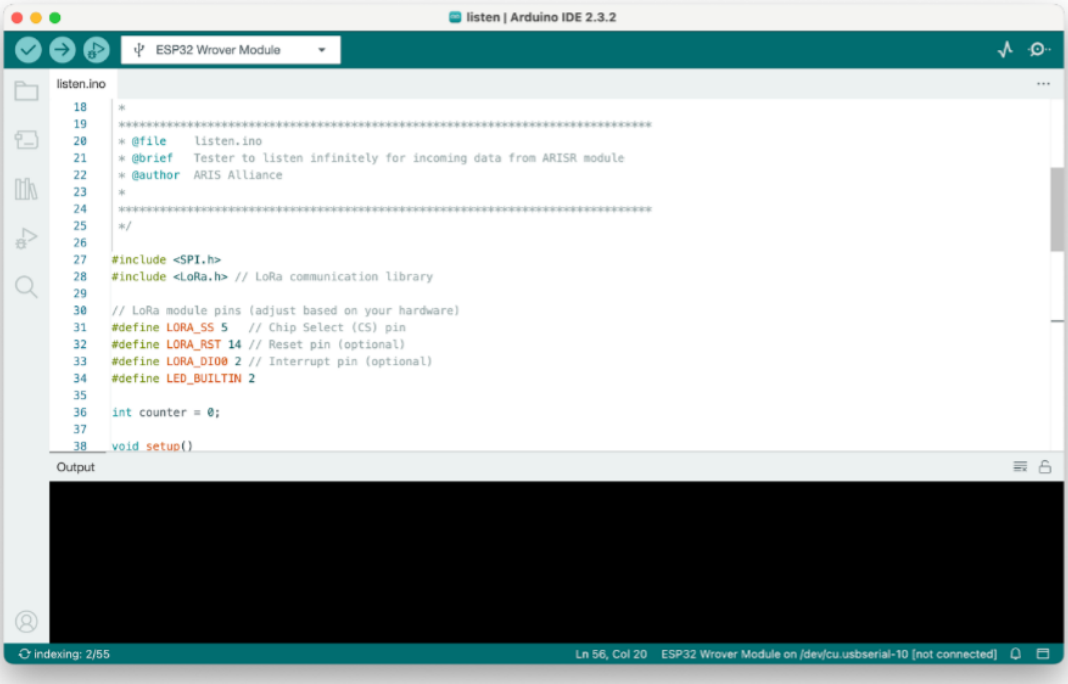

Figura 8.1: Interfaz Arduino IDE.

Con el objetivo de facilitar la implementación y el manejo del protocolo ARISR en aplicaciones desarrolladas en lenguaje C, se ha creado la librería lib-protoarisr-c. Esta biblioteca proporciona funciones esenciales para la serialización y deserialización de datos, permitiendo convertir fragmentos de información en estructuras compatibles con el protocolo ARISR (*chunks*).

El proyecto está alojado en GitHub y se encuentra disponible públicamente en el siguiente enlace:

`https://github.com/aris-radio/lib-protoarisr-c`

El repositorio incluye diversos directorios y archivos que estructuran el proyecto:

- **include/**: Contiene los archivos de encabezado (`.h`) necesarios para la interfaz pública de la librería.
- **source/**: Alberga los archivos fuente (`.c`) con la implementación de las funciones.
- **test/**: Incluye pruebas unitarias para verificar el correcto funcionamiento de la librería.
- **Makefile**: Facilita la compilación y construcción de la biblioteca mediante la herramienta `make`.

La estructura modular del proyecto permite una fácil integración en diferentes entornos y sistemas. Además, al estar escrita en C, la librería ofrece un alto rendimiento y eficiencia en la manipulación de datos.

El objetivo a largo plazo es extender esta librería a otros lenguajes de programación, promoviendo una mayor interoperabilidad y facilitando la adopción del protocolo ARISR en diversas plataformas y aplicaciones.

Un ejemplo de la interfaz de la librería es la siguiente:

```c
#ifndef LIB_ARISR_H
#define LIB_ARISR_H

#include <stdint.h>

#include "lib_arisr_base.h"
#include "lib_arisr_interface.h"
#include "lib_arisr_comm.h"
#include "lib_arisr_err.h"
#include "lib_arisr_crypt.h"
#include "lib_arisr.h"

/**
 * @brief Cleans (resets) the raw chunk buffer, freeing any allocated memory.
 */
#define ARISR_RAW_CLEAN_AND_RETURN(err_code) \
    do { \
        ARISR_proto_raw_chunk_clean(buffer); \
        return err_code; \
    } while (0)

/**
 * @brief Cleans (resets) the chunk buffer, freeing any allocated memory.
 */
#define ARISR_CLEAN_AND_RETURN(err_code) \
    do { \
        ARISR_proto_chunk_clean(buffer); \
        return err_code; \
    } while (0)


/**
 * @brief Cleans (resets) the raw chunk buffer, freeing any allocated memory.
 *
 * @param buffer Pointer to the ARISR_CHUNK_RAW structure.
 * @return kARISR_OK on success, or kARISR_ERR_GENERIC if buffer is NULL.
 */
ARISR_ERR ARISR_proto_raw_chunk_clean(ARISR_CHUNK_RAW *buffer);

/**
 * @brief Cleans (resets) the chunk buffer, freeing any allocated memory.
 *
 * @param buffer Pointer to the ARISR_CHUNK structure.
 * @return kARISR_OK on success, or kARISR_ERR_GENERIC if buffer is NULL.
 */
ARISR_ERR ARISR_proto_chunk_clean(ARISR_CHUNK *buffer);

/**
 * @brief Retrieves specific bits from a 32-bit control structure, using offset and mask.
 *
 * @param ctrl   Pointer to an ARISR_CHUNK_CTRL_RAW buffer.
```

```
52  * @param mask   Bit mask to apply after shifting.
53  * @param shift  Bit offset to start from to the last byte.
54  * @return The extracted bits as an 8-bit value.
55  */
56 ARISR_UINT8 ARISR_proto_ctrl_getField(const ARISR_UINT8 *ctrl,
        ARISR_UINT32 mask, ARISR_UINT8 shift);
57
58 /**
59  * @brief Set specific bits from a 32-bit control structure, using
        offset and mask.
60  *
61  * @param ctrl   Pointer to an ARISR_CHUNK_CTRL_RAW buffer.
62  * @param mask   Bit mask to apply after shifting.
63  * @param shift  Bit offset to start from to the last byte.
64  * @return kARISR_OK if set correctly.
65  */
66 ARISR_ERR ARISR_proto_ctrl_setField(ARISR_UINT8 *ctrl, ARISR_UINT8
        data, ARISR_UINT8 shift);
67
68 /**
69  * @brief Receives and parses raw data into an ARISR_CHUNK
        structure.
70  *
71  * This function reads the incoming data byte by byte, separating
        the protocol sections,
72  * allocating memory where needed, and checking the CRC values for
        both header and data decrypted.
73  *
74  * @param buffer [out] Pointer to the ARISR_CHUNK structure where
        parsed data will be stored and decrypted.
75  * @param data   [in]  Pointer to the raw input data buffer (e.g.,
        from the network or file).
76  * @param key    [in]  The AES-128 key used to decrypt the 'aris'
        section.
77  * @param id     [in]  The expected Network ID section to match the
         incoming data.
78  * @return kARISR_OK on success, or an error code for invalid
        parameters, CRC mismatch, etc.
79  *
80  * @note The caller is responsible for freeing the memory allocated
         for *buffer. With ARISR_proto_chunk_clean.
81  */
82 ARISR_ERR ARISR_proto_parse(ARISR_CHUNK *buffer, const ARISR_UINT8
        *data, const ARISR_AES128_KEY key, ARISR_UINT8 *id);
83
84 /**
85  * @brief Prepare and send from ARISR_CHUNK structure to a raw data
        .
86  *
87  * This function creates the raw data byte by byte, acording to the
         protocol,
88  * allocating memory where needed, and calculating the CRC values
        for both header and data.
89  *
90  * @param buffer [out] Pointer to the raw input data buffer
91  * @param length [out] Pointer to the size of the raw data buffer.
92  * @param data   [in]  Pointer to the struct output data buffer (e.
        g., from the sys).
```

```
93  * @param key    [in]  The AES-128 key used to encrypt the data
        section.
94  * @return kARISR_OK on success, or an error code for invalid
        parameters, CRC mismatch, etc.
95  *
96  * @note The caller is responsible for freeing the memory allocated
        for *buffer.
97  */
98 ARISR_ERR ARISR_proto_build(ARISR_UINT8 **buffer, ARISR_UINT32 *
    length, ARISR_CHUNK *data, const ARISR_AES128_KEY key);
99
100 /**
101  * @brief Those functions are used step by step to pack, send,
        receive and unpack the data.
102  *
103  * If you need this functios, please consider to define
        ARISR_PROTO_PARTIAL_FUNCTIONS
104  *
105  * Otherwise you can use the function ARISR_proto_parse to do all
        the process in one function.
106  * And also you can use the function ARISR_proto_build to do all
        the process in one function.
107  */
108 #if defined(ARISR_PROTO_PARTIAL_FUNCTIONS)
109 /**
110  * @brief Receives and parses raw data into an ARISR_CHUNK_RAW
        structure.
111  *
112  * This function reads the incoming data byte by byte, separating
        the protocol sections,
113  * allocating memory where needed, and checking the CRC values for
        both header and data.
114  *
115  * @param buffer Pointer to the ARISR_CHUNK_RAW structure where
        parsed data will be stored.
116  * @param data   Pointer to the raw input data buffer (e.g., from
        the network or file).
117  * @param key    The AES-128 key used to decrypt the 'aris' section
        .
118  * @param id     The expected Network ID section to match the
        incoming data.
119  * @return kARISR_OK on success, or an error code for invalid
        parameters, CRC mismatch, etc.
120  *
121  * @note The caller is responsible for freeing the memory allocated
        for *buffer only with return kARISR_OK.
122  * @note With ARISR_proto_raw_chunk_clean can free the memory.
123  * @note If any other error is returned, the buffer is not
        allocated.
124  */
125 ARISR_ERR ARISR_proto_recv(ARISR_CHUNK_RAW *buffer, const
    ARISR_UINT8 *data, const ARISR_AES128_KEY key, ARISR_UINT8 *id);
126
127 /**
128  * @brief Unpack and decrypt ARISR_CHUNK_RAW into an ARISR_CHUNK
        structure.
129  *
130  * This function parses the incoming data previosly received by
        ARISR_proto_recv,
```

```
131  * also decrypting the data section using the provided AES-128 key.
132  * And creating accessible fields for the user.
133  *
134  * @param buffer [out] Pointer to the ARISR_CHUNK structure where
         individual data will be stored.
135  * @param data   [in]  Pointer to the raw input data buffer (e.g.,
         from the network or file).
136  * @param key    [in]  The AES-128 key used to decrypt the data
         section.
137  * @return kARISR_OK on success, or an error code for invalid
         parameters, CRC mismatch, etc.
138  *
139  * @note The caller is responsible for freeing the memory allocated
          for *buffer only with return kARISR_OK.
140  * @note With ARISR_proto_chunk_clean can free the memory.
141  * @note If any other error is returned, the buffer is not
         allocated.
142  */
143 ARISR_ERR ARISR_proto_unpack(ARISR_CHUNK *buffer, ARISR_CHUNK_RAW *
        data, const ARISR_AES128_KEY key);
144
145 /**
146  * @brief Unpack and decrypt ARISR_CHUNK into an ARISR_CHUNK_RAW
         structure.
147  *
148  * This function wraps the outgoing data into a raw format,
         encrypting the data section
149  * using the provided AES-128 key. CRC are not calculated here, but
          in the send function.
150  * And creating raw struct to be ready to send.
151  *
152  * @param buffer [out] Pointer to the ARISR_CHUNK_RAW structure
         where the data will be store.
153  * @param data   [in]  Pointer to the struct output data buffer (e.
         g., from the sys).
154  * @param key    [in]  The AES-128 key used to encrypt the data
         section.
155  * @return kARISR_OK on success
156  *
157  * @note The caller is responsible for freeing the memory allocated
          for *buffer only with return kARISR_OK.
158  * @note With ARISR_proto_raw_chunk_clean can free the memory.
159  * @note If any other error is returned, the buffer is not
         allocated.
160  */
161 ARISR_ERR ARISR_proto_pack(ARISR_CHUNK_RAW *buffer, ARISR_CHUNK *
        data, const ARISR_AES128_KEY key);
162
163 /**
164  * @brief Prepare and send from ARISR_CHUNK_RAW structure to a raw
         data.
165  *
166  * This function creates the raw data byte by byte, acording to the
          protocol,
167  * allocating memory where needed, and calculating the CRC values
         for both header and data.
168  *
169  * @param buffer [out] Pointer to the raw input data buffer
```

```
170  * @param data   [in]  Pointer to the ARISR_CHUNK_RAW structure
         where parsed data will be stored.
171  * @param length [out] Pointer to the size of the raw data buffer.
172  * @return kARISR_OK on success, or an error code for invalid
         parameters, CRC mismatch, etc.
173  *
174  * @note The caller is responsible for freeing the memory allocated
          for *buffer only with return kARISR_OK.
175  * @note If any other error is returned, the buffer is not
         allocated.
176  */
177 ARISR_ERR ARISR_proto_send(ARISR_UINT8 **buffer, ARISR_CHUNK_RAW *
        data, ARISR_UINT32 *length);
178
179 #endif // ARISR_PROTO_PARTIAL_FUNCTIONS
180
181 #endif
182
183
184 /* COPYRIGHT ARIS Alliance */
```

Comúnmente debería usarse estás dos funciones.

```
1  /**
2   * @brief Receives and parses raw data into an ARISR_CHUNK
        structure.
3   *
4   * This function reads the incoming data byte by byte, separating
        the protocol sections,
5   * allocating memory where needed, and checking the CRC values for
        both header and data decrypted.
6   *
7   * @param buffer [out] Pointer to the ARISR_CHUNK structure where
        parsed data will be stored and decrypted.
8   * @param data   [in]  Pointer to the raw input data buffer (e.g.,
        from the network or file).
9   * @param key    [in]  The AES-128 key used to decrypt the 'aris'
        section.
10  * @param id     [in]  The expected Network ID section to match the
         incoming data.
11  * @return kARISR_OK on success, or an error code for invalid
        parameters, CRC mismatch, etc.
12  *
13  * @note The caller is responsible for freeing the memory allocated
         for *buffer. With ARISR_proto_chunk_clean.
14  */
15 ARISR_ERR ARISR_proto_parse(ARISR_CHUNK *buffer, const ARISR_UINT8
       *data, const ARISR_AES128_KEY key, ARISR_UINT8 *id);
16
17 /**
18  * @brief Prepare and send from ARISR_CHUNK structure to a raw data
       .
19  *
20  * This function creates the raw data byte by byte, acording to the
         protocol,
21  * allocating memory where needed, and calculating the CRC values
        for both header and data.
22  *
23  * @param buffer [out] Pointer to the raw input data buffer
```

```
24  * @param length [out] Pointer to the size of the raw data buffer.
25  * @param data   [in]  Pointer to the struct output data buffer (e.
        g., from the sys).
26  * @param key    [in]  The AES-128 key used to encrypt the data
        section.
27  * @return kARISR_OK on success, or an error code for invalid
        parameters, CRC mismatch, etc.
28  *
29  * @note The caller is responsible for freeing the memory allocated
         for *buffer.
30  */
31 ARISR_ERR ARISR_proto_build(ARISR_UINT8 **buffer, ARISR_UINT32 *
       length, ARISR_CHUNK *data, const ARISR_AES128_KEY key);
```

8.2 Firmware

El presente subapartado se centra en el desarrollo del *firmare* en la que pertenece en la capa 2 del proyecto que se muestra a continuación:

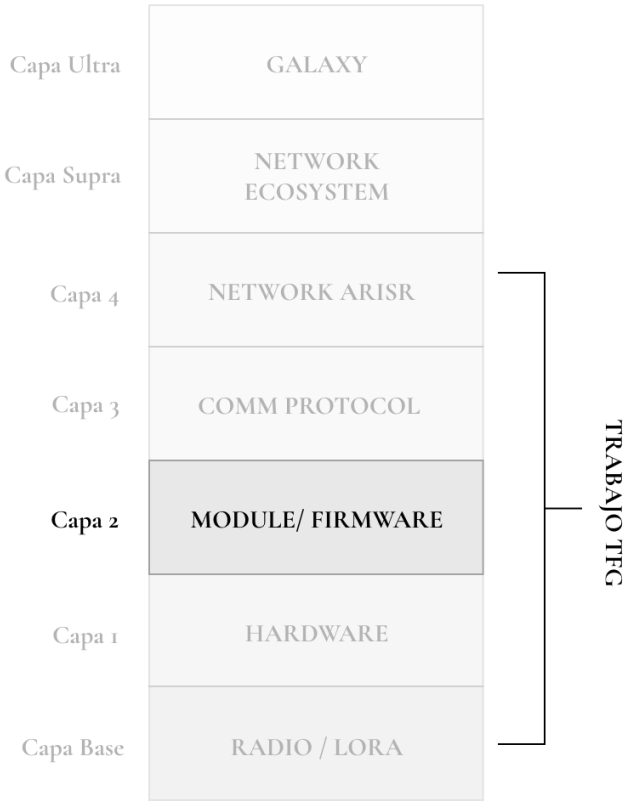

Figura 8.2: Diagrama de capas de firmware.

El desarrollo del *firmare* se basa en una plantilla previamente elaborada por el autor de este trabajo, la cual ha sido ampliada y actualizada a partir de una de las plantillas principales ofrecidas por STM32. Esta plantilla está disponible públicamente en el siguiente repositorio de GitHub:

https://github.com/ZhengLinLei/stm32f1xx-project

8.2.1. Estructura y Características de la Plantilla

La plantilla está diseñada para microcontroladores de la serie STM32F1 y permite la creación de nuevos proyectos sin la necesidad de un entorno de desarrollo integrado. Su estructura se organiza en los siguientes directorios:

- **source/**: Contiene los archivos fuente en C, incluyendo el archivo principal `main.c`.
- **include/**: Almacena los archivos de encabezado (`.h`) necesarios para el proyecto.
- **cmsis/** y **hal/**: Incluyen las bibliotecas CMSIS y HAL proporcionadas por STMicroelectronics, esenciales para la programación de bajo nivel y la abstracción de hardware.
- **BUILD.sh** y **FLASH.sh**: Scripts de automatización para la compilación y carga del *firmare* en el microcontrolador.
- **Makefile**: Facilita la compilación del proyecto utilizando herramientas de línea de comandos.

Para compilar el proyecto, es necesario instalar la herramienta `arm-none-eabi-gcc` y ejecutar el script `BUILD.sh`, previa configuración de los parámetros del MCU y CPU. El binario resultante se genera en el directorio `bin/`.

La carga del *firmare* se realiza mediante el script `FLASH.sh`, utilizando la herramienta `stlink`. Ambos scripts ofrecen opciones de ayuda accesibles mediante los comandos `BUILD.sh -h` y `FLASH.sh -h`.

8.3 Instalador de Librerías y Expansión del Repositorio

El *firmare* desarrollado incluirá un instalador que automatiza la incorporación de todas las librerías necesarias, tanto propias como externas, junto con el Makefile correspondiente. Este instalador permitirá compilar y expandir el repositorio de manera eficiente, facilitando su adaptación y escalabilidad para futuros desarrollos.

8.4 Dockerización del proyecto

Con el objetivo de garantizar un entorno de desarrollo uniforme, reproducible y portable, el proyecto de *firmare* para STM32F1xx ha sido completamente dockerizado. Esta estrategia permite evitar problemas relacionados con la instalación del *toolchain*, conflictos de dependencias o diferencias de configuración entre los equipos de los desarrolladores.

Objetivos

La configuración con Docker proporciona las siguientes ventajas:

- Un entorno de compilación preconfigurado con todas las herramientas necesarias.
- La posibilidad de compilar y flashear el *firmare* sin necesidad de instalar *software* adicional en el sistema anfitrión.

- Facilidad para nuevos desarrolladores, quienes solo necesitan un comando para comenzar.
- Reproducibilidad garantizada entre distintos sistemas operativos.

Contenido de la Imagen Docker

La imagen Docker está basada en Ubuntu 22.04 e incluye:

- **gcc-arm-none-eabi**: Toolchain oficial de ARM para compilar *firmare* destinado a microcontroladores Cortex-M.
- **make, cmake**: Herramientas de construcción necesarias para proyectos basados en `Makefile`.
- **stlink-tools**: Utilidades de línea de comandos para flashear dispositivos STM32.
- Herramientas comunes como `git`, `wget`, `curl` y `nano`.
- **Python 3 + pip**: útiles para scripts adicionales o automatización futura.

Además, se crea un usuario sin privilegios llamado `devuser`, lo que favorece la seguridad dentro del contenedor.

8.5 Open Source

El presente proyecto, denominado ARISR (perteneciente a la iniciativa ARIS Alliance), se desarrolla bajo una filosofía de *software* libre y de código abierto (Open Source). Con ello se busca no solo fomentar la transparencia y la colaboración, sino también permitir que otros desarrolladores, investigadores y entusiastas puedan estudiar, adaptar, modificar y redistribuir el código con total libertad tanto particular o fabricante, respetando siempre las condiciones de la licencia utilizada.

La idea del código abierto facilita la expansión del proyecto ARISR y la colaboración de todo los fabricantes de módulo ARISR, así mismo la transparencia del contenido y el *firmare* personalizado creado por cada empresa.

8.5.1. Licencia GNU General Public License v2

La mayoría de los componentes del proyecto está licenciado bajo los términos de la *GNU General Public License (GPL)*, versión 2, junio de 1991, una de las licencias más reconocidas y utilizadas en el ámbito del *software* libre. Esta licencia, promovida por la *Free Software Foundation (FSF)*, garantiza a los usuarios finales las cuatro libertades esenciales:

- **Libertad de uso**: el *software* puede ser utilizado para cualquier propósito, sin restricciones.
- **Libertad de estudio**: el código fuente está disponible para que cualquier persona pueda analizar su funcionamiento.
- **Libertad de modificación**: se permite alterar el código para adaptarlo a distintas necesidades.

- **Libertad de redistribución**: tanto el *software* original como sus versiones modificadas pueden ser compartidos libremente.

A diferencia de licencias más permisivas (como *MIT* o *BSD*), la GPL impone una condición importante: cualquier trabajo derivado que se distribuya debe conservar la misma licencia. Es decir, se debe publicar bajo los mismos términos de la GPL v2, lo que asegura que el *software* derivado también permanezca libre.

Esta característica se conoce como *copyleft* y ayuda a la comunidad a prevenir la aparición de implementaciones o fabricantes de *firmare* ARISR malintencionados.

8.5.2. Compromiso con la Comunidad

Al adoptar esta licencia, el proyecto ARISR manifiesta un compromiso activo con la comunidad de desarrollo de *software* libre, promoviendo prácticas de colaboración abierta, documentación accesible y mejora continua a través de aportes externos. Asimismo, se invita a otros desarrolladores a participar en el crecimiento del proyecto mediante contribuciones en el repositorio oficial o el reporte de errores.

Esta elección de licencia también permite que el proyecto pueda ser utilizado en contextos educativos, académicos y experimentales sin restricciones, lo cual resulta especialmente valioso en entornos como universidades, centros de investigación o proyectos de innovación tecnológica abierta.

CAPÍTULO 9
Protocolo de Comunicación

En este capítulo se describe el protocolo de comunicación propuesto para el proyecto, detallando su estructura y funcionamiento. Se explican todos los campos y subcampos definidos, con el objetivo de proporcionar una referencia clara sobre cómo construir el *chunk* de datos que debe enviarse y cómo interpretarlo correctamente.

Para facilitar la comprensión del avance del proyecto, se incluye el diagrama de componentes, en el cual se destaca el módulo que será analizado en este capítulo.

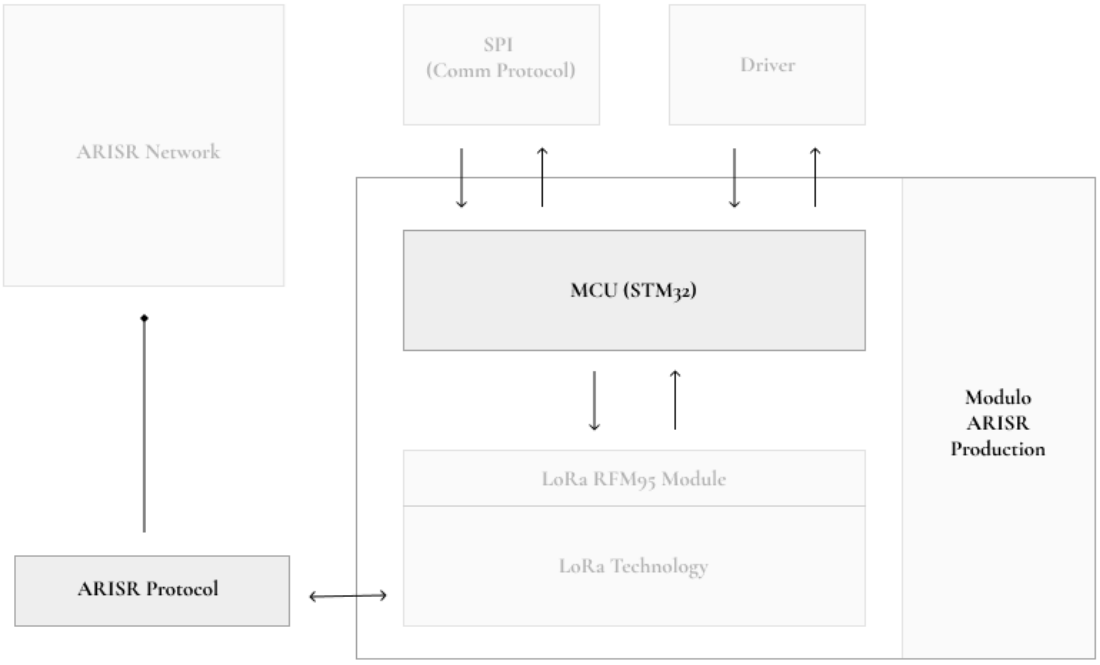

Figura 9.1: Diagrama de componentes de la capa protocolo.

Los dos módulos que aparecen en el diagrama corresponde a la Capa 3 de ARISR

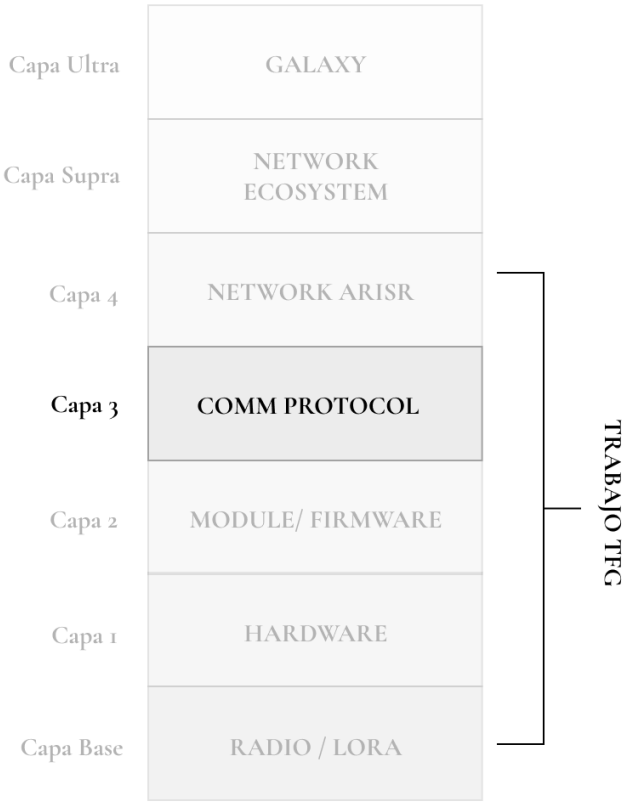

Figura 9.2: Diagrama de capas de protocolo.

9.1 Estructura del Protocolo

El protocolo de comunicación se basa en la construcción de un chunk de datos del mensaje completo a enviar, el cual se compone de los siguientes campos:

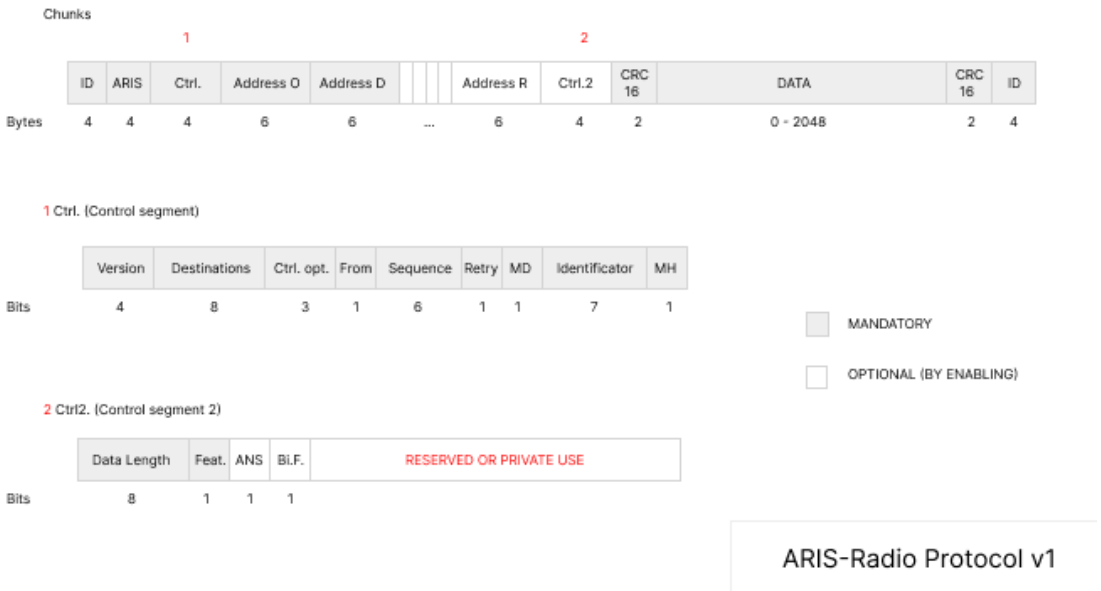

Figura 9.3: Estructura del protocolo de comunicación.

Conforme el modulo de ARISR va recibiendo el flujo del mensaje a enviar se va creando los *chunks* de datos, los cuales se van empaquetando el mensaje de 2048 bytes a 2048 bytes, y se van enviando a través de la capa física. En el proceso de construcción del chunk, se incluyen campos de control, datos y CRC para garantizar la integridad y seguridad de la comunicación.

Para la construcción de los *chunks*, se requieren varias informaciones como el ID de red, una llave de cifrado AES-128 si se requiere cifrado extra por hardware, el tipo de mensaje que se va a enviar, el mensaje a enviar y el tamaño del mensaje a enviar proporcionado por la capa superior.

9.1.1. Campos del Protocolo

- **ID**: Identificador de red, utilizado para distinguir entre diferentes redes o grupos de dispositivos.
- **ARIS**: Campo *Magic Number* que indica el inicio de un mensaje ARISR, al mismo tiempo hace de uso como integridad a que un dispositivo pertenece realmente a la red.
- **Ctrl**: Campo de control que contiene los metadatos del chunk, como el tipo de mensaje, flags de control, etc.
- **Address O**: Dirección MAC del origen, que identifica al dispositivo que envía el mensaje.
- **Address D**: Dirección MAC del destino, que identifica al dispositivo que recibe el mensaje.
- **Address X**: Dirección MAC del destino multiple
- **Address R**: Dirección MAC del relay, si el chunk ha sido reenviado por un nodo intermedio.
- **Ctrl 2**: Campo de control adicional preparado para futuras expansiones del protocolo y funcionalidades del fabricante.
- **CRC Header**: Código de verificación de redundancia cíclica para el encabezado del chunk. Se profundiza a continuación en la sección de seguridad.
- **Data**: Campo que contiene la carga útil del mensaje, es decir, los datos que se desean transmitir.
- **CRC Data**: Código de verificación de redundancia cíclica para la carga útil, utilizado para detectar errores en la transmisión de los datos. Se profundiza a continuación en la sección de seguridad.
- **ID**: Identificador de red, utilizado para distinguir entre diferentes redes o grupos de dispositivos. En este caso significa fin de mensaje

Para poder analizar en detalle cada campo, se adjunta un ejemplo de chunk de datos con valores ficticios, el cual se utilizará como referencia para explicar la función y el formato de cada campo dentro del protocolo. Dicho ejemplo ha sido *formateado* con la librería **lib-protoarisr-c** desarrollada en este proyecto, lo que permite una visualización clara y estructurada de los datos contenidos en el chunk.

```
[2025-02-16 00:49:58] [INFO] [ID]           00 11 22 33
[2025-02-16 00:49:58] [INFO] [ARIS]         41 52 49 53
[2025-02-16 00:49:58] [INFO] [CTRL]
[2025-02-16 00:49:58] [INFO]   [VER]           2
[2025-02-16 00:49:58] [INFO]   [DEST]          2
[2025-02-16 00:49:58] [INFO]   [OPT]           0
[2025-02-16 00:49:58] [INFO]   [FROM]          0
[2025-02-16 00:49:58] [INFO]   [SEQ]           1
[2025-02-16 00:49:58] [INFO]   [RET]           0
[2025-02-16 00:49:58] [INFO]   [MD]            1
[2025-02-16 00:49:58] [INFO]   [ID]            110
[2025-02-16 00:49:58] [INFO]   [MH]            1
[2025-02-16 00:49:58] [INFO] [ORIGIN]       00 1A 2B 3C 4D 5E
[2025-02-16 00:49:58] [INFO] [DEST A]       FA 16 3E 2F EC A8
[2025-02-16 00:49:58] [INFO] [DEST B]
[2025-02-16 00:49:58] [INFO]   [000]           00 1A 2B 3C 4D 5E
[2025-02-16 00:49:58] [INFO]   [001]           00 1B 63 84 45 E6
[2025-02-16 00:49:58] [INFO] [CTRL2]
[2025-02-16 00:49:58] [INFO]   [DL]            41
[2025-02-16 00:49:58] [INFO]   [FEAT]          0
[2025-02-16 00:49:58] [INFO]   [NEG]           0
[2025-02-16 00:49:58] [INFO]   [FREQ]          0
[2025-02-16 00:49:58] [INFO] [CRC H]        D5 F1
[2025-02-16 00:49:58] [INFO] [CRC D]        D0 1F
[2025-02-16 00:49:58] [INFO] [END]          00 11 22 33
[2025-02-16 00:49:58] [INFO]
[2025-02-16 00:49:58] [INFO] [DATA]
[2025-02-16 00:49:58] [INFO] 0000: 00 01 02 03 |....|
[2025-02-16 00:49:58] [INFO] 0004: 04 05 06 07 |....|
[2025-02-16 00:49:58] [INFO] 0008: 08 09 0A 0B |....|
[2025-02-16 00:49:58] [INFO] 000C: 0C 0D 0E 0F |....|
[2025-02-16 00:49:58] [INFO] 0010: 10 11 12 13 |....|
[2025-02-16 00:49:58] [INFO] 0014: 14 15 16 17 |....|
[2025-02-16 00:49:58] [INFO] 0018: 18 19 1A 1B |....|
[2025-02-16 00:49:58] [INFO] 001C: 1C 1D 1E 1F |....|
[2025-02-16 00:49:58] [INFO] 0020: 20 21 22 23 | !=#|
[2025-02-16 00:49:58] [INFO] 0024: 24 25 26 27 |$%&%|
[2025-02-16 00:49:58] [INFO] 0028: 28          |(   |
```

Identificación de Red

El campo de identificación de red se emplea para determinar a qué red o grupo de dispositivos pertenece un determinado mensaje. Este mecanismo resulta especialmente relevante en entornos donde coexisten múltiples redes, ya que permite a los dispositivos filtrar y procesar únicamente los mensajes que son pertinentes para su dominio de comunicación.

El identificador de red es configurable por el usuario y no constituye un valor estático, lo que proporciona una elevada flexibilidad en la gestión de las comunicaciones dentro de distintos contextos o aplicaciones. Este campo posee una longitud de 4 bytes, permitiendo un total de 2^{32} identificadores únicos (4.294.967.296 valores posibles). Dichos identificadores pueden reutilizarse en diferentes dominios o zonas sin generar conflictos operacionales.

El identificador de red se incluye tanto al inicio como al final del mensaje, actuando como un delimitador lógico de trama. Esta redundancia facilita los procesos de detección, sincronización y validación de los mensajes durante la transmisión.

Con el fin de garantizar la seguridad e integridad de la comunicación, el campo de identificación de red se utiliza conjuntamente con mecanismos criptográficos y de verificación descritos en la sección correspondiente a la seguridad del protocolo.

Como ejemplo de aplicación, el identificador de red puede configurarse para representar distintos grupos funcionales dentro de un sistema de IoT, tales como sensores, actuadores o dispositivos de monitorización.

En el ejemplo considerado se emplea el identificador de red `00 11 22 33`, el cual será utilizado en las secciones posteriores para ilustrar el funcionamiento del protocolo.

Llave de Seguridad ARIS

El campo ARIS actúa como un *magic number* o firma de protocolo que posee una longitud de 4 bytes, cuyo propósito es verificar que un mensaje pertenece a una red compatible con el sistema ARIS. Este campo se deriva de la representación ASCII de la palabra *ARIS* y constituye un mecanismo adicional de control de pertenencia a la red.

En configuraciones de acceso público, este campo puede mantenerse sin modificación, adoptando directamente los valores ASCII correspondientes:

$$A = \texttt{41}, \quad R = \texttt{52}, \quad I = \texttt{49}, \quad S = \texttt{53}$$

De esta forma, cualquier dispositivo que reciba un mensaje con dicha firma podrá identificarlo como perteneciente a una red ARIS válida, siempre que el identificador de red coincida.

En redes privadas o con requisitos de seguridad más estrictos, este campo puede modificarse mediante un proceso de derivación basado en la clave AES-128 asociada a la red. Concretamente, el valor ASCII base de *ARIS* se incrementa de forma cíclica utilizando el último byte de la clave criptográfica.

Por ejemplo, si el último byte de la clave AES-128 es `12`, el campo ARIS se calcula como:

$$\texttt{41 52 49 53} + \texttt{12 12 12 12} = \texttt{53 64 61 65}$$

lo que corresponde a la cadena ASCII `Sdae`. Este procedimiento permite establecer una firma diferenciada para redes privadas, facilitando la validación del origen del mensaje y añadiendo una capa adicional de control de acceso.

En el capítulo siguiente se describirá en detalle la arquitectura criptográfica asociada a la llave de seguridad ARIS, así como la justificación de su diseño e integración dentro del protocolo. El valor especifico configurado en una red privada se aplica a la base de la red pública con el valor de ARIS más sumando de forma cíclica el último byte de la llave AES-128 de la red. Un ejemplo es por ejemplo si mi clave acaba en 12 pues el valor que se debe añadir al campo es el siguiente: `41 52 49 53 + 12 12 12 12 = 53 64 61 65`, lo que hace que el campo de ARIS se convierta en `Sdae` en ascii, lo que hace que cualquier dispositivo que reciba un mensaje con ese campo pueda identificarlo como perteneciente a la red privada si el valor del identificador coincide. En el capitulo siguiente verá en detalle toda la información que hay detrás de la llave de seguridad ARIS y porque el uso de ella.

Campo de Control

El campo de control (Ctrl) que posee de 4 bytes es un componente fundamental del protocolo de comunicación, ya que contiene información esencial para la gestión y el procesamiento de los mensajes. Este campo se estructura en varios subcampos que permiten definir el formato, la dirección, la secuencia y otras características del mensaje.

El campo de control se compone de los siguientes subcampos que son divididos en bits:

- **Versión**: 4 bits que indican la versión del protocolo utilizado, lo que permite la compatibilidad hacia atrás y la evolución del sistema.
- **Destinarions**: 8 bits que especifican el número de destinatarios a los que quieren enviar el mensaje. Con un máximo de 255 destinatarios, lo que permite una comunicación eficiente en redes con múltiples nodos. A esto se le suma que se puede crear direcciones MAC virtuales para que un solo destino pertenezca a un grupo de dispositivos, lo que facilita la gestión de comunicaciones grupales.
- **Opciones**: 3 bits reservados para futuras expansiones. (No me acuerdo en profundidad lo que se acordó en el diseño del protocolo)
- **From**: 1 bit que indica si el mensaje es un mensaje reenviado de un relay o no, `0` para mensajes originales y `1` para mensajes reenviados, lo que permite a los dispositivos identificar la procedencia del mensaje y aplicar políticas de procesamiento adecuadas.
- **Secuencia**: 8 bits que representan el número de secuencia del mensaje, lo que facilita la detección de mensajes duplicados o fuera de orden, mejorando la fiabilidad de la comunicación. La secuencia comienza en 0 y se incrementa con cada chunk enviado, reiniciándose a 0 después de alcanzar el valor máximo de 64.
- **Retransmisión**: 1 bit que indica si el mensaje requiere retransmisión en caso de no ser recibido correctamente, lo que permite implementar mecanismos de confiabilidad en la comunicación.
- **MD (More Data)**: 1 bit que indica si hay más datos por enviar después del chunk actual, si esta a `1` indica que debería esperar más *chunks* del mismo identificador de mensaje para completar la recepción del mensaje completo.
- **ID**: 7 bits que representan el identificador del mensaje, cuando el usuario decide enviar un mensaje todo el conjunto de chunk que pertenece al mismo mensaje tiene el mismo identificador.
- **MH (More Header)**: 1 bit que indica si hay más campos de encabezado por venir después del campo de control actual, si está activo se debería leer el campo de control 2 para obtener más información.

El campo `Ctrl` se codifica en formato *big-endian*, de modo que el bit más significativo se transmite en primer lugar. En consecuencia, la numeración de bits adoptada en ARISR sigue el rango 31–0, donde el bit 31 corresponde al bit más significativo del primer byte y el bit 0 al bit menos significativo del último byte.

Dirección de Origen y Destino

Los dos campos siguientes corresponden a la dirección de origen (Address O) y la dirección de destino (Address D), cada una con una longitud de 6 bytes. Estas direcciones se utilizan para identificar de manera única a los dispositivos que participan en la comunicación dentro de la red.

Cabe destacar que Origen y Destino no son mutuamente excluyentes, es decir, un dispositivo puede actuar como origen en un mensaje y como destino en otro, dependiendo del contexto de la comunicación. Esto permite una flexibilidad significativa en la interacción entre los nodos de la red. Además existe la posibilidad de añadir direcciones virtuales tanto como origen como destino que facilita la gestión del multicast y la comunicación grupal, lo que es especialmente útil en escenarios de IoT donde múltiples dispositivos pueden necesitar recibir el mismo mensaje. Al estilo **MQTT**. (Pero mejor :D)

Podemos ver que en el ejemplo se ha configurado la dirección de origen como `00 1A 2B 3C 4D 5E` y la dirección de destino como `FA 16 3E 2F EC A8`. Estas direcciones pueden ser asignadas de manera estática o dinámica, dependiendo de las necesidades de la red y los dispositivos involucrados. En redes con un gran número de dispositivos, es común utilizar un esquema de asignación de direcciones que facilite la gestión y evite conflictos, como por ejemplo la asignación basada en el identificador de red o el uso de direcciones MAC virtuales para grupos de dispositivos.

Dirección Destino Multiple

Los campos siguientes se activan con el subcampo Destinations del campo de opciones. En caso de que el número de destinatarios sea mayor a 1, se activa el campo de dirección destino múltiple (Address X), el cual puede contener hasta 255 direcciones adicionales, cada una con una longitud de 6 bytes. Este mecanismo permite enviar un mismo mensaje a múltiples destinatarios de manera eficiente, sin necesidad de duplicar el mensaje para cada destino individual. En el ejemplo se muestra que el campo de dirección destino múltiple contiene dos direcciones adicionales: `00 1A 2B 3C 4D 5E` y `00 1B 63 84 45 E6`, lo que indica que el mensaje será enviado a tres destinatarios en total (incluyendo la dirección de destino principal).

Dirección de Relay

Este campo se activa cuando el subcampo From del campo de control indica que el mensaje es un mensaje reenviado por un nodo intermedio (relay). En este caso, la dirección de relay (Address R) contiene la dirección MAC del nodo que ha reenviado el mensaje, esto nos permite rastrear la ruta que ha seguido el mensaje a través de la red y aplicar políticas de procesamiento específicas para mensajes reenviados. En el ejemplo, el campo de dirección de relay no está presente, lo que indica que el mensaje es un mensaje original enviado directamente por el dispositivo de origen.

Campo de Control 2

El campo de control 2 (Ctrl 2) es un componente adicional del protocolo que se activa cuando el subcampo MH (More Header) del campo de control indica que hay más campos de encabezado por venir. Este campo tiene una longitud de 4 bytes y está diseñado para proporcionar información adicional sobre el mensaje, así como para permitir futuras expansiones del protocolo sin afectar la compatibilidad con versiones anteriores.

Tambien se puede usar para que cada fabricante pueda añadir campos personalizados sin necesidad de modificar el protocolo base, lo que facilita la integración de funcionalidades específicas para cada caso de uso o aplicación.

El campo de control 2 se estructura en los siguientes subcampos:

- **DL (Data Length)**: 8 bits que indican la longitud de la carga útil (Data) del mensaje, lo que permite a los dispositivos saber cuántos bytes de datos se están transmitiendo. Si se deja en blanco significa que el campo data esta completa por lo que se puede prenscindir activar el campo de control 2 a menos de que se quiera usar para otras funciones.
- **FEAT (Features)**: 1 bit que activa los campos siguietes (Sinceramente se puede omitir)
- **NEG (Negotiation)**: 1 bits (Ya no me acuerdo)
- **FREQ (Frequency)**: 1 bits (Ya no me acuerdo)

Campos de CRC Cabeceras

Este campo se aplica el algoritmo de CRC-16-CCITT para calcular sobre el encabezado del chunk, que incluye todos los campos desde el identificador de red hasta el campo de control 2 (si está presente). El resultado del cálculo se almacena en el campo CRC Header, que tiene una longitud de 2 bytes. Este mecanismo permite detectar errores en la transmisión del encabezado, asegurando que los dispositivos puedan validar la integridad de esta parte crítica del mensaje antes de procesarlo.

Datos

Este campo es donde contiene el dato final, que puede estar encriptado o no dependiendo de la configuración del mensaje con AES-128 que se meciono anteriormente con la llave de serguridad ARIS. La longitud de este campo puede variar dependiendo del los datos que se desea enviar, y su tamaño está determinado por el campo DL (Data Length) del control 2. En el ejemplo, se muestra una carga útil de 41 bytes.

Campos de CRC Datos

Al igual que el campo de CRC para el encabezado, este campo se calcula utilizando el algoritmo de CRC-16-CCITT, pero en este caso se aplica sobre la carga útil del chunk. El resultado del cálculo se almacena en el campo CRC Data, que también tiene una longitud de 2 bytes. Este mecanismo permite detectar errores en la transmisión de los datos, asegurando que los dispositivos puedan validar la integridad de la carga útil antes de procesarla o entregarla a la capa superior.

Fin de chunk

Finalmente, el campo de fin de chunk se utiliza para marcar el final de un chunk de datos dentro del mensaje. Este campo tiene una longitud de 4 bytes y suele contener el mismo valor que el identificador de red.

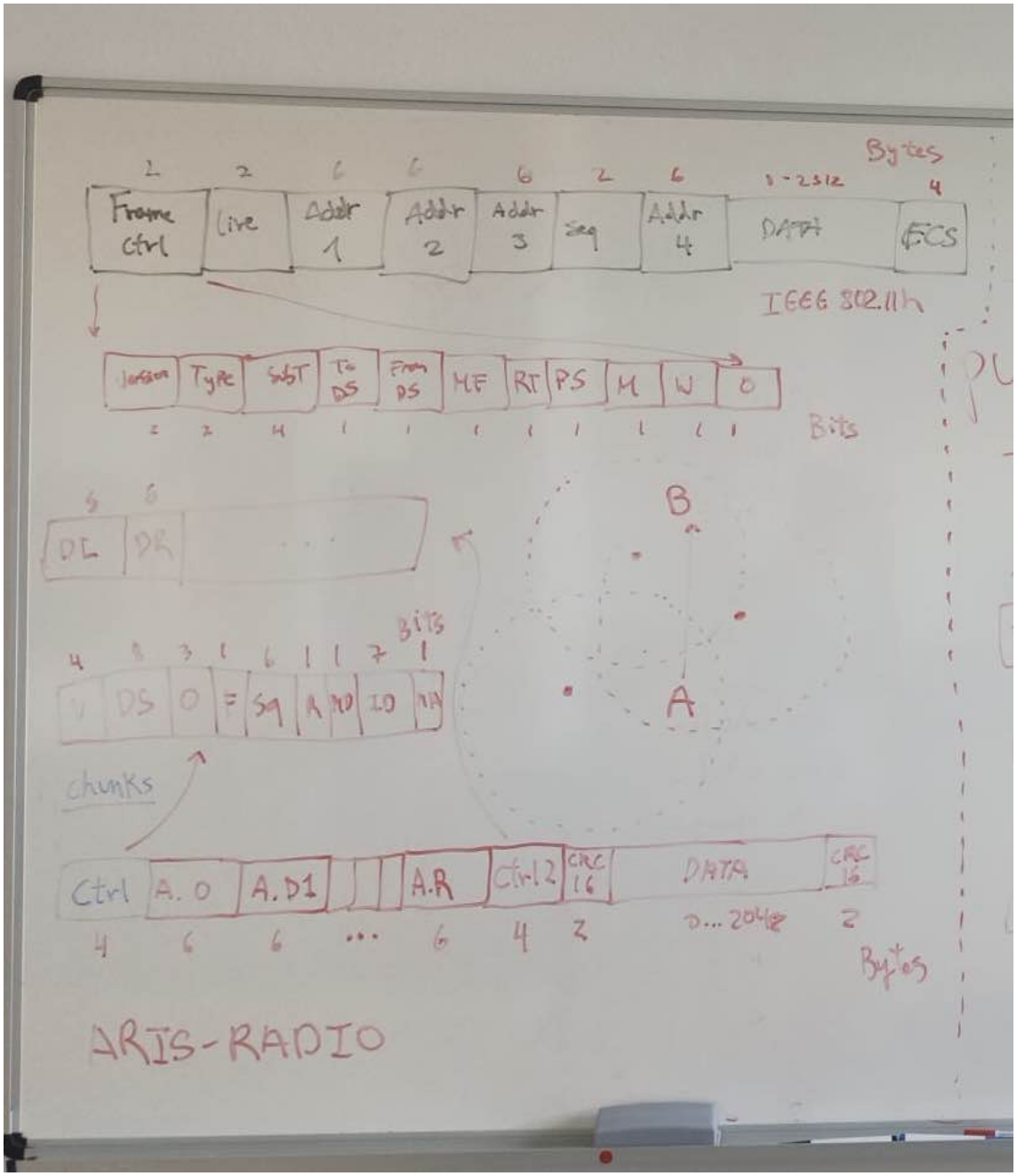

Figura 9.4: Diseño del protocolo de comunicación.

En el apartado de anexos puede encontrar una tabla resumen de todos los campos del protocolo, con su descripción, longitud y formato, lo que facilita la referencia rápida.

9.2 Máquina de Estados del Protocolo

La operación del protocolo se basa en un modelo de máquina de estados tanto en el nodo emisor como en el nodo receptor, con el objetivo de garantizar la correcta construcción, transmisión, validación y reensamblado de los mensajes fragmentados en *chunks*. Este enfoque permite estructurar el flujo de procesamiento de forma determinista, facilitando la implementación del protocolo en sistemas embebidos y entornos distribuidos.

En el nodo emisor, la máquina de estados se encarga de la construcción progresiva de cada *chunk* a partir del flujo de datos proporcionado por la capa superior. En primer lugar, se inicializa la cabecera del mensaje con el identificador de red y la firma del protocolo. Posteriormente, se genera el campo de control principal, en el cual se configuran los parámetros relativos a la versión del protocolo, número de destinatarios, número de secuencia y flags de control. En caso de requerirse cabeceras extendidas, se añade el campo de control adicional correspondiente.

Una vez construida la cabecera, el emisor procede a calcular el CRC de cabecera, tras lo cual se inserta la carga útil del *chunk*, cuya longitud puede ser fija o variable dependiendo de la configuración asignada. Finalmente, se calcula el CRC de datos y se añade el delimitador de fin de *chunk*. Este proceso se repite de manera iterativa hasta completar la transmisión del mensaje completo, incrementando el número de secuencia en cada fragmento generado.

En el nodo receptor, la máquina de estados sigue un proceso secuencial de validación y ensamblado. Inicialmente, se verifica la presencia del identificador de red que actúa como delimitador de inicio de *chunk*. A continuación, se comprueba la firma del protocolo (campo ARIS), con el fin de determinar la pertenencia del mensaje a la red configurada. Una vez superada esta verificación, el receptor analiza el campo de control para determinar la estructura del *chunk*, incluyendo la posible presencia de cabeceras opcionales.

Tras la lectura completa de la cabecera, se valida el CRC correspondiente. En caso de ser correcto, se procede a la lectura de la carga útil y a la verificación del CRC de datos. Finalmente, se comprueba la coherencia del delimitador de fin de *chunk*. Si todas las verificaciones resultan satisfactorias, los datos son entregados a la capa superior o almacenados temporalmente en el buffer de reensamblado, en función del estado del bit *More Data*. Este proceso continúa hasta la recepción completa del mensaje fragmentado.

9.3 Tratamientos de Errores

Con el objetivo de garantizar la robustez del protocolo frente a condiciones adversas de transmisión, se definen una serie de mecanismos de detección y tratamiento de errores. Estos mecanismos permiten a los nodos participantes adoptar decisiones coherentes ante la recepción de información corrupta o inconsistente.

En particular, si el CRC de cabecera no es válido, el *chunk* debe ser descartado inmediatamente sin procesar la carga útil. De forma análoga, si el CRC de datos resulta incorrecto, el receptor podrá descartar el fragmento o solicitar su retransmisión en función del estado del bit de control correspondiente. Asimismo, la discrepancia entre el identificador de inicio y el identificador de fin de *chunk* se considera una condición de error crítico, implicando la invalidación del fragmento completo.

Otros casos de error contemplados incluyen la recepción de un valor no reconocido en el campo ARIS, lo cual implica que el mensaje no pertenece a la red configurada, así como la detección de números de secuencia inesperados, que pueden indicar pérdida de fragmentos o duplicación de datos. Del mismo modo, el valor cero en el subcampo de destinatarios se considera reservado y debe interpretarse como una condición de *chunk* inválido.

La correcta definición de estas políticas de tratamiento de errores contribuye a mejorar la fiabilidad del protocolo y a reducir la probabilidad de propagación de información inconsistente hacia capas superiores del sistema.

CAPÍTULO 10
Seguridad de la Comunicación

ARISR incorpora un conjunto de mecanismos de seguridad diseñados para proteger la integridad, confidencialidad y autenticidad de las comunicaciones dentro de la red. Estos mecanismos se implementan a través de campos específicos en el protocolo, así como mediante la integración de algoritmos criptográficos robustos.

En primer lugar, el campo ARIS actúa como una firma de protocolo que permite verificar la pertenencia de un mensaje a una red compatible. En configuraciones privadas, este campo se deriva de la clave AES-128 asociada a la red, lo que añade una capa adicional de control de acceso y dificulta la suplantación de identidad por parte de actores malintencionados. Además, el uso de CRC para el encabezado y la carga útil permite detectar errores en la transmisión, asegurando que los dispositivos puedan validar la integridad de los mensajes antes de procesarlos.

A continuación, se describen en detalle los mecanismos de seguridad.

10.1 Integridad de los Mensajes: CRC-16-CCITT

Todo los *chunks* llevan encima dos CRC, uno para la cabecera o preambulo del chunk y otro para la carga útil o datos del chunk. El algoritmo utilizado para calcular ambos CRC es el CRC-16-CCITT, que es un estándar ampliamente utilizado en sistemas de comunicación para detectar errores en la transmisión de datos. Este algoritmo se basa en la división polinómica de los datos por un polinomio generador específico, lo que permite generar un código de verificación que puede ser utilizado para validar la integridad de los mensajes. En el caso de ARISR, el CRC de cabecera se calcula sobre todos los campos desde el identificador de red hasta el campo de control 2 (si está presente), mientras que el CRC de datos se calcula exclusivamente sobre la carga útil del chunk. La inclusión de estos mecanismos de verificación contribuye a mejorar la fiabilidad de la comunicación, permitiendo a los dispositivos detectar y descartar mensajes corruptos antes de procesarlos o entregarlos a capas superiores del sistema.

La ventaja que nos da de este CRC es que es facil de implementar en hardware, lo que permite que el cálculo del CRC se realice de manera eficiente y rápida aparte de su beneficio de ahorro de espacio en el mensaje al ser un CRC de 2 bytes. Sin embargo, es importante destacar que el CRC-16-CCITT no proporciona protección contra ataques malintencionados, ya que un atacante podría modificar los datos y recalcular el CRC para que coincida con el nuevo contenido. Por esta razón, el CRC se utiliza principalmente como un mecanismo de detección de errores accidentales en la transmisión, mientras que la confidencialidad y autenticidad de los mensajes se abordan mediante otros mecanismos criptográficos, como el cifrado AES-128.

10.1.1. Teoría del CRC-16-CCITT

El algoritmo consiste en dividir el polinomio correspondiente al mensaje entre un polinomio generador predefinido. El residuo de dicha división constituye el valor del CRC, el cual se adjunta al mensaje antes de su transmisión. En el receptor, se repite el mismo proceso de división incluyendo el CRC recibido; si el residuo resultante es nulo, se considera que el mensaje no ha sufrido errores de transmisión.

En el caso del CRC-16-CCITT, el polinomio generador utilizado es:

$$G(x) = x^{16} + x^{12} + x^5 + 1 \tag{10.1}$$

cuya representación binaria es:

$$\texttt{0x1021} \tag{10.2}$$

En la implementación adoptada en el protocolo ARISR, se emplean los siguientes parámetros:

- Polinomio generador: `0x1021`
- Valor inicial: `0xFFFF`
- Sin reflexión de bits de entrada ni salida
- Sin inversión final del resultado

Este conjunto de parámetros corresponde a una de las variantes más extendidas del CRC-16-CCITT en sistemas de comunicación embebidos.

Ejemplo de cálculo del CRC

Para ilustrar el proceso de cálculo, considérese una secuencia de datos compuesta por los siguientes bytes:

$$\texttt{01 02 03 04} \tag{10.3}$$

El procedimiento de cálculo puede describirse de forma simplificada mediante los siguientes pasos:

1. Inicializar el registro CRC con el valor `0xFFFF`.
2. Procesar cada byte del mensaje mediante operaciones de desplazamiento y XOR con el polinomio generador.
3. Tras procesar todos los bytes, el valor final del registro constituye el CRC calculado.

Aplicando el algoritmo CRC-16-CCITT a la secuencia anterior, se obtiene el siguiente resultado:

$$\texttt{CRC} = \texttt{0x89C3} \tag{10.4}$$

Por tanto, el bloque transmitido incluiría los datos originales seguidos del valor de verificación:

$$\texttt{01 02 03 04 89 C3} \tag{10.5}$$

10.1.2. Librería de Cálculo de CRC de ARISR

La librería de **lib-protoarisr-c** tiene un ejemplo de código para el calculo del CRC super simple

```
ARISR_UINT16 ARISR_crypt_crc16_calculate(const ARISR_UINT8 *data,
    const ARISR_UINT32 length)
{
     ARISR_UINT16 crc = CRC16_INITIAL_VALUE;
     ARISR_UINT32 i;

     for (i = 0; i < length; i++) {
          ARISR_UINT8 index = (crc >> 8) ^ data[i];  // Get index
          crc = (crc << 8) ^ crc16_table[index]; // XOR with table
              value
     }
     return crc;
}
```

10.2 Confidencialidad: Cifrado AES-128-ECB

Por otra parte para prevenir la interceptación y lectura no autorizada de los datos, aunque en capas superiores se pueden implementar mecanismos de cifrado más robustos, el protocolo ARISR permite la opción de cifrar la carga útil de los mensajes utilizando el algoritmo AES-128 en modo ECB (Electronic Codebook). Este algoritmo es un estándar de cifrado simétrico que utiliza una clave de 128 bits para cifrar bloques de datos de 16 bytes. En el contexto de ARISR se utiliza principalmente para proporcionar una capa básica de confidencialidad en mensajes cortos o fragmentados. Y su principal función es poder distinguir la verdadera identidad de un nodo en una supuesta red privada, ya que el campo de ARIS se deriva de la clave.

De esta forma, los dispositivos que reciben un mensaje cifrado pueden intentar descifrar la carga útil utilizando la clave AES-128 configurada para la red. Si el descifrado es exitoso y el contenido resultante es coherente con el formato esperado, se puede considerar que el mensaje es legítimo y proceder a su procesamiento. En caso contrario, el mensaje puede ser descartado o tratado como sospechoso.

La ventaja de usar AES-128 es que es un algoritmo ampliamente reconocido por su seguridad y eficiencia, lo que proporciona una protección adecuada para la mayoría de las aplicaciones de IoT. Sin embargo, es importante tener en cuenta que el modo ECB no es el más seguro para cifrar grandes cantidades de datos o datos con patrones repetitivos, por lo que se recomienda utilizarlo con precaución y considerar alternativas como el modo CBC o GCM para futuras versiones con *hardware* más potente.

Para más información sobre la implementación del cifrado AES-128 en ARISR, se puede consultar en el Anexo: **Implementación de AES-128 en ARISR**, donde se detalla el proceso de cifrado y descifrado, así como ejemplos de código para su integración en dispositivos embebidos.

La librería de **lib-protoarisr-c** tiene un ejemplo de código para el encriptado

```
ARISR_ERR ARISR_aes_data_encrypt(const ARISR_AES128_KEY key,
                                 const ARISR_UINT8 *input,
                                 ARISR_UINT32 input_len,
                                 ARISR_UINT8 **output,
                                 ARISR_UINT32 *output_len)
{
    ARISR_UINT32 offset;
    ARISR_UINT8 *padded_data;
    // Validate input parameters
    if (!input || input_len == 0 || !output || !output_len) {
        return kARISR_ERR_INVALID_ARGUMENT;
    }

    // Calculate PKCS#7 padding requirements
    // Always add padding (even if input is block-aligned) per RFC 5652
    const ARISR_UINT8 pad_value = AES_BLOCKLEN - (input_len % AES_BLOCKLEN);
    const ARISR_UINT32 padded_len = input_len + pad_value;

    // Security boundary check: prevent integer overflow
    if (padded_len < input_len) {
        return kARISR_ERR_BUFFER_OVERFLOW;
    }

    // Allocate secure buffer for padded data
    padded_data = (ARISR_UINT8 *)malloc(padded_len);
    if (!padded_data) {
        return kARISR_ERR_GENERIC;
    }

    // Prepare plaintext with padding
    memcpy(padded_data, input, input_len);
    memset(padded_data + input_len, pad_value, pad_value);

    // Initialize AES context
    struct AES_ctx ctx;
    AES_init_ctx(&ctx, key);

    // Encrypt using ECB mode (warning: ECB is insecure for most real-world use)
    // Process each block independently
    for (offset = 0; offset < padded_len; offset += AES_BLOCKLEN) {
        AES_ECB_encrypt(&ctx, padded_data + offset);
    }

    // Set output parameters - transfer ownership of buffer to caller
    *output = padded_data;
    *output_len = padded_len;

    return kARISR_OK;
}
```

Y un ejemplo sencillo para el rotato de la palabra ARIS para crear la firma de la red privada

```
ARISR_ERR ARISR_aes_aris_encrypt(const ARISR_AES128_KEY key,
    ARISR_UINT8 *aris)
{
    ARISR_UINT8 i;

    if (!aris) {
        return kARISR_ERR_GENERIC;
    }

    // Las byte (16 bytes -> index 15)
    ARISR_UINT8 last_byte = (ARISR_AES_IS_ZERO_KEY(key)) ? 0 : (
        ARISR_UINT8)key[ARISR_AES128_BLOCK_SIZE - 1];

    // Module increment
    if (last_byte)
    for (i = 0; i < ARISR_PROTO_ARIS_SIZE; i++) {
        aris[i] = (ARISR_UINT8)(aris[i] + last_byte);
    }

    // No hay verificacion aqui, pues es cifrado. Simplemente
        retornamos OK.
    return kARISR_OK;
}
```

CAPÍTULO 11
Arquitectura de Red

Ahora viene la mejor parte y la más importante del protocolo, la arquitectura de red constituye uno de los pilares fundamentales del protocolo ARISR, definiendo cómo los diferentes nodos interactúan entre sí, cómo se organizan las comunicaciones y qué reglas rigen el flujo de información dentro del sistema. Este capítulo aborda en detalle los roles que pueden adoptar los dispositivos en la red, los principios de diseño que guían la arquitectura y los mecanismos de filtrado que garantizan la seguridad y la eficiencia de las comunicaciones.

11.1 Principios de Diseño de Red

Anteriormente ya se detallo los principios de diseño de red de ARISR pero se procederá a profundizar en más detalles. La arquitectura de red de ARISR se fundamenta en un conjunto de principios que garantizan su robustez, escalabilidad y seguridad. Estos principios han sido establecidos para asegurar que la red pueda adaptarse a diferentes escenarios de aplicación, desde despliegues IoT de baja densidad hasta sistemas críticos con múltiples nodos distribuidos.

11.2 Roles de los Nodos en la Red

- **Nodos Activos**: Son los dispositivos que generan y envían mensajes dentro de la red. Estos nodos pueden ser sensores, actuadores, dispositivos de control o cualquier otro tipo de dispositivo que necesite comunicarse con otros nodos para compartir información o recibir comandos. Los nodos activos son responsables de iniciar las comunicaciones y pueden actuar como origen o destino de los mensajes.
- **Nodos Pasivos**: Son los dispositivos que esperan a recibir mensajes pero no generan mensajes por sí mismos. Estos nodos pueden ser relays o repeaters que se encargan de retransmitir mensajes de otros nodos para extender la cobertura de la red, o pueden ser nodos de monitorización que simplemente escuchan el tráfico de la red sin participar activamente en la generación de mensajes. Los nodos pasivos son esenciales para mantener la conectividad y la integridad de la red, especialmente en entornos con obstáculos o largas distancias.

La arquitectura ARISR define diferentes roles que pueden adoptar los nodos dentro de la red, cada uno con responsabilidades específicas que contribuyen al funcionamiento global del sistema.

11.2.1. Nodo

Un nodo es cualquier dispositivo conectado a la red, capaz de participar en la transmisión de datos. Los nodos pueden incluir una amplia variedad de dispositivos, desde satélites y rovers planetarios hasta dispositivos IoT y ordenadores personales. Cada nodo desempeña un papel en el soporte de la funcionalidad de la red, ya sea enviando, recibiendo o retransmitiendo datos.

11.2.2. Relay

Un relay es un dispositivo que puede operar en modo de espera o activo, aunque típicamente permanece en estado de espera. Su propósito es extender la cobertura de transmisión permitiendo que los *chunks* se transmitan de manera fiable a mayores distancias o en entornos donde las señales directas son débiles. Cuando está en modo activo, el relay recibe y retransmite los *chunks*, asegurando una conexión estable y continua dentro de la red inalámbrica.

11.2.3. Repeater

El repeater tiene la función específica de repetir la señal desde el origen hasta el destino. A diferencia del relay, que puede implementar lógica adicional, el repeater actúa de manera más simple, simplemente amplificando y retransmitiendo la señal recibida. Puede instalarse con o sin funcionalidades de relay adicionales.

11.2.4. Courier

Un courier es un nodo que opera simultáneamente en dos o más redes, funcionando como un puente de mensajes. Este tipo de nodo facilita la comunicación transfiriendo datos entre redes, permitiendo una conectividad fluida y el flujo de datos a través de sistemas que de otra manera estarían separados.

11.2.5. Ghost Node

Un ghost node es un nodo que existe y participa activamente en todos los eventos pero no pertenece a una red específica. Desde la perspectiva de un nodo dentro de una red determinada, un ghost node se define como cualquier nodo que no forma parte de esa red. Sin embargo, sigue siendo un nodo activo para cualquier otra red a la que sí pertenezca.

Hay que destacar que un nodo pasivo con role de repeater o relay no puede generar mensajes por sí mismo, sino que se limita a retransmitir los mensajes que recibe de otros nodos. Esto implica que un nodo con rol de relay no puede actuar como origen o destino en la red, de está forma se garantiza una clara separación de funciones y se evita la posibilidad de que un nodo pueda generar mensajes no autorizados o actuar como un punto de ataque dentro de la red. A su vez mejora la eficiencia de la red al permitir que los relays se centren exclusivamente en la tarea de retransmisión, sin necesidad de gestionar la generación o procesamiento de mensajes.

Sin embargo un nodo activo puede actuar como origen, destino y relay al mismo tiempo. Lo que deja claro que un nodo pasivo puede tener una complejidad de implementación menor que un nodo activo.

11.3 Aislamiento de Redes

Uno de los principios más importantes es el aislamiento completo entre redes. Este principio establece que cualquier chunk recibido por un nodo o relay que no pertenezca a la red designada debe ser descartado inmediatamente. Este mecanismo de filtrado estricto previene la propagación de datos no autorizados, irrelevantes o potencialmente maliciosos, salvaguardando la integridad y eficiencia de la comunicación.

El aislamiento se implementa mediante un proceso de filtrado en múltiples capas que todo nodo de retransmisión debe seguir:

1. **Identificadores Únicos de Red**: Cada red ARISR dispone de un identificador único (4 bytes) incrustado en cada chunk transmitido. Los nodos inspeccionan este identificador y lo comparan con el suyo propio. Si no coinciden, el chunk es descartado inmediatamente.

2. **Verificación de Dirección de Origen**: Los nodos mantienen una caché de direcciones de nodos conocidos y confiables dentro de la red. La dirección de origen del chunk se compara con esta caché para verificar su legitimidad.

3. **Cifrado y Autenticación**: Todos los nodos operan bajo una clave de cifrado compartida, asegurando que solo dispositivos autorizados puedan encriptar o desencriptar los datos transmitidos.

11.4 Descentralización y Autonomía

ARISR promueve una arquitectura descentralizada donde los nodos pueden operar de manera autónoma sin depender de una infraestructura centralizada. Esta característica es fundamental para aplicaciones en entornos remotos o adversos donde la conectividad con un servidor central no puede garantizarse.

La red ARISR puede formar topologías en malla auto-organizadas, donde los nodos se conectan directamente entre sí, extendiendo la cobertura de manera orgánica a medida que se incorporan nuevos dispositivos.

11.4.1. Auto-Organización

Los nodos en la red ARISR pueden descubrirse mutuamente y establecer rutas de comunicación de forma dinámica. Esto permite que la red se adapte a cambios en la topología, como la adición o eliminación de nodos, sin necesidad de intervención manual. La auto-organización se logra mediante algoritmos de descubrimiento de vecinos y establecimiento de rutas que permiten a los nodos identificar y comunicarse con otros nodos dentro de su alcance.

El proceso de integración de un nuevo nodo en la red ARISR implica los siguientes pasos:

1. **Descubrimiento de Vecinos**: El nuevo nodo emite mensajes de descubrimiento para identificar otros nodos dentro de su alcance. Estos mensajes contienen información básica sobre el nodo, como su identificador y capacidades. Teniendo en cuenta que el nuevo nodo tenga asignado el identificador y la clave de cifrado correcta, los nodos existentes responderán con mensajes de bienvenida que incluyen su propia

información y la confirmación de que el nuevo nodo es parte de la red. (Puede que CTRL Opt sirva para esto, algún número destinaría a mensaje de descubrimiento)

2. **Establecimiento de Rutas**: Una vez que el nuevo nodo ha sido reconocido por los nodos existentes, se inicia el proceso de establecimiento de rutas. El nuevo nodo puede solicitar información sobre las rutas disponibles hacia otros nodos en la red, y los nodos existentes pueden proporcionar esta información basada en su conocimiento de la topología actual. El nuevo nodo selecciona las rutas más eficientes para comunicarse con otros nodos, lo que permite una integración fluida en la red. En este paso tiene en cuenta si los *chunks* recibidos han sido asignados con campo From o no, al mismo tiempo las respuestas contienen las coordenadas de los nodos para que el nuevo nodo pueda construir un mapa de la red y optimizar sus rutas.

3. **Sincronización de Estado**: Para garantizar la coherencia de la red, el nuevo nodo puede sincronizar su estado con los nodos existentes. Esto incluye la actualización de tablas de enrutamiento, cachés de direcciones y cualquier otra información relevante para el funcionamiento de la red. Esta sincronización se realiza mediante mensajes específicos que permiten al nuevo nodo obtener una visión completa del estado actual de la red.

11.5 Topologías de Red Soportadas

ARISR es compatible con múltiples topologías de red, lo que permite adaptar el despliegue a los requisitos específicos de cada aplicación.

11.5.1. Topología Punto a Punto

La configuración más simple, donde dos nodos se comunican directamente sin intermediarios. Ideal para enlaces directos entre dispositivos dentro del rango de cobertura.

11.5.2. Topología Estrella

Múltiples nodos se comunican con un nodo central (que puede actuar como *relay* o *gateway*). Aunque ARISR no depende de infraestructura centralizada, esta topología puede implementarse cuando existe un punto de coordinación natural.

11.5.3. Topología en Malla

La topología más característica de ARISR, donde los nodos se conectan entre sí formando una red descentralizada. Las ventajas incluyen:

- **Redundancia**: Si un nodo falla, los mensajes pueden ser redirigidos a través de otros nodos, aumentando la resiliencia de la red.

- **Escalabilidad**: La red puede crecer orgánicamente a medida que se añaden nuevos nodos sin necesidad de reconfiguración centralizada.

- **Flexibilidad**: Permite una comunicación eficiente en entornos dinámicos donde la topología puede cambiar con frecuencia.

11.6 Diagramas de Arquitectura

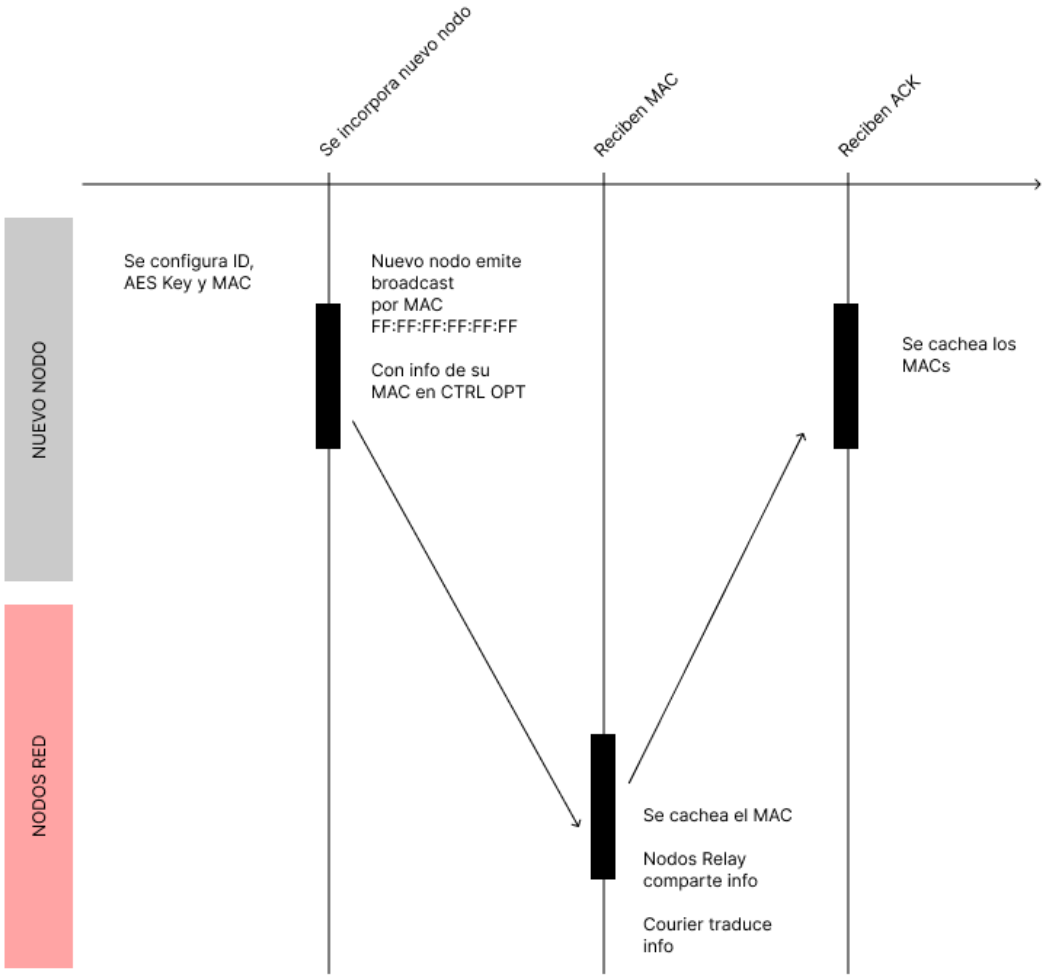

Figura 11.1: Diagrama descubrimiento e incorporación

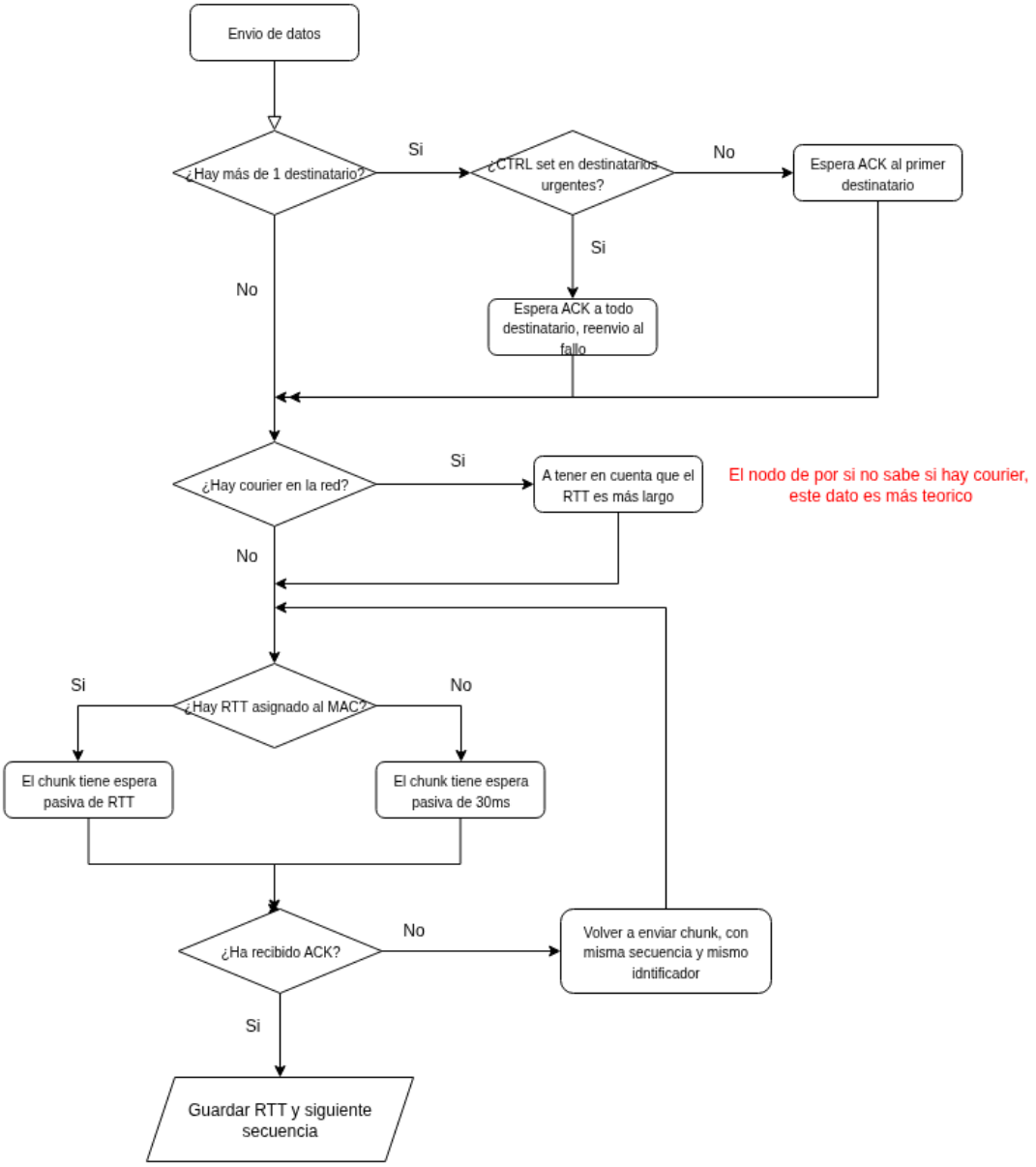

Figura 11.2: Diagrama de reenvío de mensajes

La comuniación entre redes se hace mediante nodos con rol de courier, que actuan como puente entre redes. Estos nodos pueden recibir mensajes de una red, verificar su legitimidad y retransmitirlos a otra red a la que también pertenezcan, facilitando así la interconexión entre diferentes sistemas. Se habilita con el CTRL OPT a 0b003 y el mensaje solo lo recibe los couriers para retransmitirlo a la otra red, el mensaje se construye con el mismo formato pero con el campo ARIS de la otra red para que los nodos de esa red puedan identificarlo como un mensaje legítimo.

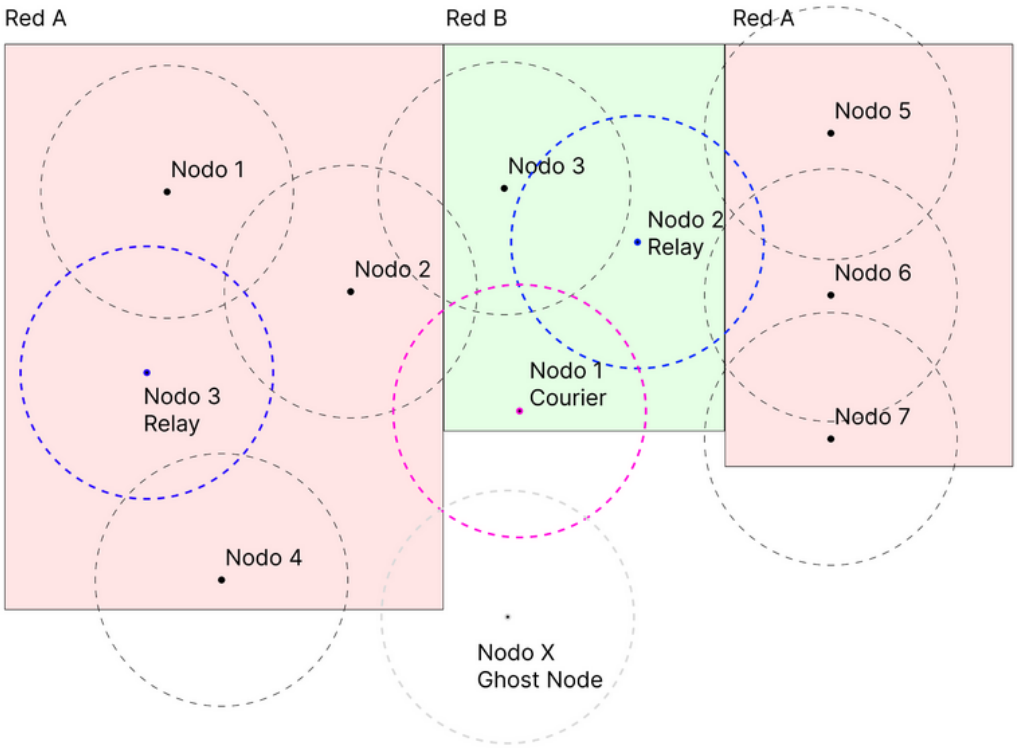

Figura 11.3: Dibujo de comunicación entre redes mediante nodos courier.

CAPÍTULO 12
Implementación y Resultados

Para validar el diseño del protocolo ARISR, se ha llevado a cabo una implementación de referencia utilizando la librería **lib-protoarisr-c**, partiendo de la plantilla de código **firmware-stm32f1xx**.

https://github.com/aris-radio/firmware-stm32f1xx

Se ha hecho una pequeña prueba de concepto para verificar la correcta construcción y transmisión de mensajes entre dos nodos utilizando el protocolo ARISR. En esta prueba, se han configurado dos nodos STM32F103C8T6 para comunicarse entre sí a través de una comunicación LoRa utilizando el módulo SX1276. Uno de los nodos actúa como emisor, generando mensajes ARISR con diferentes configuraciones de control, mientras que el otro nodo actúa como receptor, validando la integridad de los mensajes recibidos mediante los mecanismos de CRC y autenticación implementados.

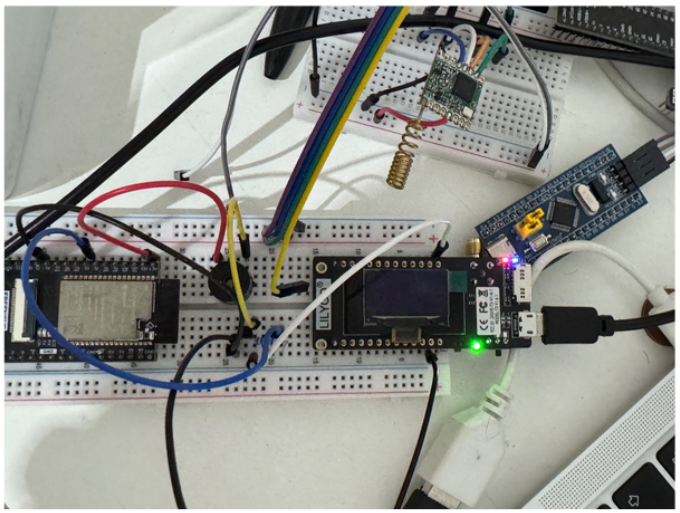

Figura 12.1: Mini prueba de conceptos rápidos

A pesar que la prueba realiza ha sido una breve simulación de envios y recepciones (asincronos) han podido verificarse los siguientes aspectos:

- La correcta construcción de los mensajes ARISR, incluyendo la generación de los campos de control y la inserción de la carga útil.
- La validación de los mensajes recibidos mediante el cálculo y verificación de los CRC para el encabezado y la carga útil.

- La capacidad del receptor para descartar mensajes corruptos o no pertenecientes a la red configurada, demostrando el funcionamiento del mecanismo de aislamiento de redes.

Las pruebas han sido de topología simple (mensaje de punto a punto) sin integración de ninguna red, pero por falta de tiempo no se han podido realizar pruebas más complejas como topología en malla o integración de múltiples nodos para verificar la auto-organización.

Se han realizado diferentes trazas de mensajes enviados y recibidos como las que se muestrasn a continuación:

Figura 12.2: Trazas de mensajes enviados y recibidos.

En detalle, la prueba que se ha realizado se ha seguido este esquema. Se ha construido dos nodos donde cada nodo hay un circuito donde simulan un hardaware conectado a LoRa con firmware sencillo de ARISR y otro MCU que actua de cliente. El cliente en esta prueba hace un envio de datos en *loop* cada 5 segundos, en la que los datos se envian activando al hardware ARISR e internamente se construye el paquete con la librería para

luego ser transmitido por el modulo LoRa. El nodo receptor recibe los mensajes, valida su integridad con la configuración **ARISR Key** configurada y activa al cliente por medio de interrupción. Una vez recibida el mensaje se va sustrayendo el chunk y enviandoselo al cliente para que este pueda procesarlo. En esta prueba el cliente simplemente imprime por consola el mensaje recibido, pero en una aplicación real podría realizar cualquier tipo de procesamiento o acción en función del contenido del mensaje.

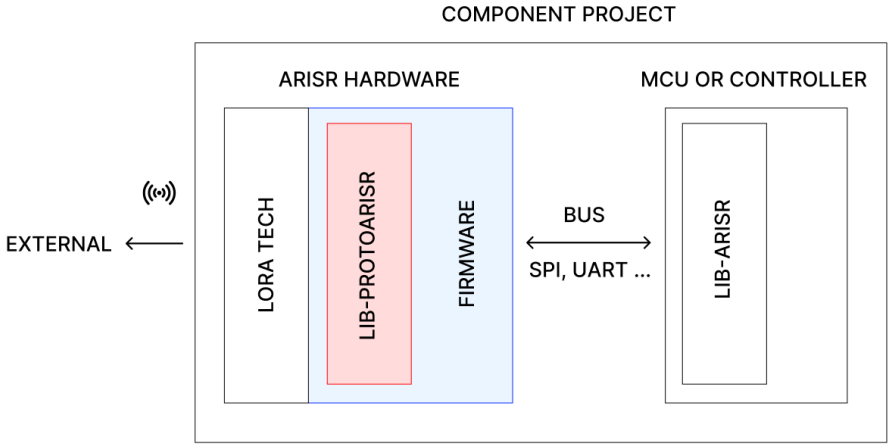

Figura 12.3: Esquema de la prueba de concepto.

Figura 12.4: Pruebas de funcionamiento.

CAPÍTULO 13
Conclusiones

El presente Trabajo de Fin de Grado ha permitido abordar el diseño de un sistema de comunicación inalámbrica orientado a entornos exigentes, como aplicaciones espaciales, sistemas autónomos e infraestructuras IoT distribuidas. A lo largo del desarrollo del proyecto ARIS Radio, se han alcanzado los objetivos principales relacionados con el diseño conceptual y técnico del sistema, sentando una base sólida para futuras evoluciones.

En primer lugar, se ha logrado diseñar un protocolo de comunicación propio, adaptable, eficiente y orientado a entornos con recursos limitados. Este protocolo contempla aspectos clave como la estructuración de los mensajes, la detección de errores y mecanismos básicos de seguridad, lo que garantiza una comunicación robusta en condiciones adversas.

Asimismo, se ha desarrollado el diseño del módulo hardware, integrando tecnologías como LoRa y comunicación SPI, lo que demuestra la viabilidad de implementar soluciones de largo alcance con bajo consumo energético. Las pruebas iniciales realizadas han permitido validar la comunicación entre dispositivos y confirmar el correcto funcionamiento del sistema en un entorno controlado.

Otro aspecto relevante es la definición de una arquitectura de red descentralizada, capaz de soportar múltiples topologías (punto a punto, estrella y malla), lo cual representa una ventaja significativa frente a soluciones tradicionales más centralizadas. Este enfoque permite mayor flexibilidad, escalabilidad y resiliencia en escenarios dinámicos o sin infraestructura.

No obstante, este trabajo se ha centrado principalmente en las fases iniciales de diseño y validación, dejando aspectos clave como la implementación completa del sistema, la optimización del driver y el desarrollo avanzado de la arquitectura de red como líneas de trabajo futuro.

En conclusión, este TFG no solo cumple con los objetivos planteados, sino que también establece una base tecnológica y conceptual sólida sobre la que construir sistemas de comunicación más avanzados, demostrando el potencial del enfoque propuesto en contextos reales de alta exigencia.

13.1 Líneas futuras

Con el fin de facilitar la continuidad del proyecto por parte de futuros desarrolladores o investigadores, se identifican las siguientes líneas de trabajo:

- **Implementación completa del sistema:**

  - Desarrollo completo del firmware del protocolo.
  - Integración del driver multiplataforma (PC, móvil, sistemas embebidos).
  - Validación en entornos reales.
- **Optimización del rendimiento:**
  - Mejora de la eficiencia en transmisión (latencia y throughput).
  - Optimización del consumo energético.
  - Reducción del overhead del protocolo.
- **Evolución de la capa física:**
  - Desarrollo de un sistema de radio propio.
  - Incremento del ancho de banda.
  - Adaptación a entornos espaciales reales.
- **Seguridad:**
  - Sustitución de AES-ECB por modos más seguros (CBC o GCM).
  - Implementación de autenticación robusta.
  - Gestión segura de claves.
- **Arquitectura de red avanzada:**
  - Implementación de redes malladas dinámicas.
  - Desarrollo de algoritmos de routing adaptativo.
  - Gestión de nodos (descubrimiento, sincronización, tolerancia a fallos).
- **Escalabilidad y validación:**
  - Simulación de redes con gran número de nodos.
  - Pruebas en escenarios reales (drones, rovers, IoT).
  - Evaluación en condiciones extremas.
- **Ecosistema open source:**
  - Publicación y mantenimiento del repositorio.
  - Mejora de la documentación.
  - Fomento de la comunidad de desarrollo.

13.2 Aportaciones del trabajo

Este trabajo proporciona una base sólida para futuros desarrollos, destacando las siguientes aportaciones:

- Un protocolo de comunicación funcional y extensible.
- Un diseño hardware validado para sistemas IoT y aplicaciones distribuidas.
- Una arquitectura de red descentralizada y escalable.
- Documentación detallada que facilita la reutilización y ampliación del sistema.

En consecuencia, este proyecto puede servir como punto de partida para futuras investigaciones, desarrollos académicos o aplicaciones industriales en el ámbito de las comunicaciones inalámbricas avanzadas.

Bibliografía

[1] Augustin, A., Yi, J., Clausen, T., Townsley, W. A Study of LoRa: Long Range & Low Power Networks for the Internet of Things. *Sensors*, vol. 16, no. 9, 2016. doi: 10.3390/s16091466

[2] Raza, U., Kulkarni, P., Sooriyabandara, M. Low Power Wide Area Networks: An Overview. *IEEE Communications Surveys & Tutorials*, vol. 19, no. 2, pp. 855–873, 2017. doi: 10.1109/COMST.2017.2652320

[3] Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., Ayyash, M. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015. doi: 10.1109/COMST.2015.2444095

[4] Bor, M., Vidler, J., Roedig, U. LoRa for the Internet of Things. *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks*, 2016.

[5] Georgiou, O., Raza, U. Low Power Wide Area Network Analysis: Can LoRa Scale? *IEEE Wireless Communications Letters*, vol. 6, no. 2, pp. 162–165, 2017. doi: 10.1109/LWC.2016.2647247

[6] Gislason, D. ZigBee Wireless Networking. *Newnes (Elsevier)*, 2008.

[7] Jang, Y., Yang, H., Kim, H. Performance Evaluation of Bluetooth Mesh Networks. *IEEE Access*, vol. 7, pp. 143410–143422, 2019. doi: 10.1109/ACCESS.2019.2944983

[8] Cerf, V., Burleigh, S., Hooke, A., et al. Delay-Tolerant Network Architecture. *IEEE Communications Magazine*, vol. 44, no. 11, pp. 128–136, 2006.

[9] Burleigh, S., Hooke, A., Torgerson, L., et al. Delay-Tolerant Networking: An Approach to Interplanetary Internet. *IEEE Communications Magazine*, vol. 41, no. 6, pp. 128–136, 2003.

[10] Croce, D., Gucciardo, M., Mangione, S., Santaromita, G., Tinnirello, I. Impact of LoRa Imperfect Orthogonality: Analysis of Link-Level Performance. *IEEE Communications Letters*, vol. 22, no. 4, pp. 796–799, 2018.

[11] Atzori, L., Iera, A., Morabito, G. The Internet of Things: A Survey. *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[12] Mekki, K., Bajic, E., Chaxel, F., Meyer, F. A Comparative Study of LPWAN Technologies for Large-Scale IoT Deployment. *ICT Express*, vol. 5, no. 1, pp. 1–7, 2019.

[13] Adelantado, F., Vilajosana, X., Tuset-Peiró, P., et al. Understanding the Limits of LoRaWAN. *IEEE Communications Magazine*, vol. 55, no. 9, pp. 34–40, 2017.

[14] Akyildiz, I., Wang, X. Wireless Mesh Networks: A Survey. *Computer Networks*, vol. 47, no. 4, pp. 445–487, 2005.

[15] Semtech Corporation. SX1276/77/78/79 LoRa Transceiver Datasheet. Documento técnico del transceptor LoRa utilizado en módulos como RFM95. https://cdn-shop.adafruit.com/product-files/3179/sx1276_77_78_79.pdf

[16] HopeRF. RFM95/96/97/98W LoRa Transceiver Module Datasheet. Hoja técnica del módulo LoRa RFM95 utilizado en prototipos IoT. https://cdn.sparkfun.com/assets/learn_tutorials/8/0/4/RFM95_96_97_98W.pdf

[17] STMicroelectronics. STM32F4 Series Datasheet. Especificaciones técnicas del microcontrolador STM32F4 basado en ARM Cortex-M4. https://www.st.com/resource/en/datasheet/stm32f407vg.pdf

[18] LoRa Alliance. LoRaWAN Specification v1.0.4. Especificación oficial del protocolo LoRaWAN. https://lora-alliance.org/wp-content/uploads/2021/11/LoRaWAN-Link-Layer-Specification-v1.0.4.pdf

[19] Bluetooth SIG. Bluetooth Mesh Networking Specifications. Especificación oficial para redes malladas Bluetooth Low Energy. https://www.bluetooth.com/specifications/mesh-specifications

[20] Consultative Committee for Space Data Systems. CCSDS Space Data Link Protocols. Estándares para comunicación y manejo de datos en misiones espaciales. https://public.ccsds.org

APÉNDICE A
Tabla de Campos del Protocolo

Campo	Tamaño	Oblig.	Descripción
ID (inicio)	4 B	Sí	Identificador de red y delimitador de inicio de chunk
ARIS	4 B	Sí	Firma del protocolo (*magic number*)
Ctrl	4 B	Sí	Campo de control principal (versión, secuencia, flags, etc.)
Address O	6 B	Sí	Dirección MAC del origen
Address D	6 B	Sí	Dirección MAC del destino principal
Address X	6·N B	No	Direcciones MAC adicionales (si DEST >1)
Address R	6 B	No	Dirección MAC del relay (si FROM = 1)
Ctrl2	4 B	No	Campo de control extendido (si MH = 1)
CRC Header	2 B	Sí	CRC-16-CCITT de la cabecera
Data	Variable	Sí	Carga útil del chunk
CRC Data	2 B	Sí	CRC-16-CCITT de la carga útil
ID (fin)	4 B	Sí	Identificador de red como delimitador de fin

Tabla A.1: Estructura general del chunk del protocolo ARISR

Bits	Subcampo	Tamaño	Descripción
31–28	VER	4 bits	Versión del protocolo. Permite compatibilidad entre distintas revisiones.
27–20	DEST	8 bits	Número de destinatarios del chunk. Si su valor es mayor que 1, se habilitan direcciones adicionales en el campo Address X.
19–17	OPT	3 bits	Bits reservados para opciones futuras del protocolo.
16	FROM	1 bit	Indica si el chunk ha sido reenviado por un relay. Valor 0 para mensaje original y 1 para mensaje reenviado.
15–8	SEQ	8 bits	Número de secuencia del chunk dentro del mensaje completo. Permite ordenar y detectar pérdidas o duplicados.
7	RET	1 bit	Indica si el chunk requiere retransmisión en caso de error o pérdida.
6	MD	1 bit	*More Data*. Indica si existen más *chunks* asociados al mismo mensaje.
5–1	ID	5 bits	Identificador lógico del mensaje al que pertenece el chunk.
0	MH	1 bit	*More Header*. Indica si existe una cabecera extendida y, por tanto, si debe leerse el campo Ctrl 2.

Tabla A.2: Subcampos del campo Ctrl del protocolo ARISR.

Bits	Subcampo	Tamaño	Descripción
31–24	DL	8 bits	*Data Length*. Longitud de la carga útil del chunk en bytes.
23	FEAT	1 bit	Activa el uso de características extendidas o campos adicionales definidos por el fabricante.
22	NEG	1 bit	Bit reservado para funciones de negociación del protocolo.
21	FREQ	1 bit	Bit reservado para funciones relacionadas con frecuencia o configuración de transmisión.
20–0	RES	21 bits	Bits reservados para futuras ampliaciones del protocolo. Deben transmitirse a 0 en la versión actual.

Tabla A.3: Subcampos del campo Ctrl 2 del protocolo ARISR.

APÉNDICE B
AES-128 en ARISR

El cifrado AES-128 en modo ECB constituye uno de los mecanismos fundamentales de seguridad implementados en el protocolo ARISR para garantizar la confidencialidad de los datos transmitidos. Este algoritmo de cifrado simétrico, ampliamente reconocido y estandarizado, utiliza una clave de 128 bits para proteger la carga útil de los mensajes, asegurando que únicamente los dispositivos autorizados que posean la clave correspondiente puedan acceder al contenido original de la información.

B.1 Fundamentos del cifrado AES-128-ECB

El AES es un algoritmo de cifrado por bloques que opera sobre bloques de datos de tamaño fijo de 16 bytes. En el modo ECB, cada bloque de texto plano se cifra de manera independiente utilizando la misma clave, sin ningún mecanismo de realimentación o encadenamiento entre bloques.

B.1.1. Características del modo ECB

- **Independencia de bloques**: Cada bloque de 16 bytes se cifra de forma aislada, lo que permite procesar los datos en paralelo y simplifica la implementación en sistemas con recursos limitados.
- **Determinismo**: Dado un mismo bloque de texto plano y una misma clave, el bloque cifrado resultante será siempre idéntico. Esta propiedad puede ser explotada en ataques de análisis de patrones.
- **Simplicidad**: La implementación es directa y requiere menos recursos computacionales que otros modos como CBC o GCM.

B.1.2. Limitaciones del modo ECB

Es importante destacar que el modo ECB no es recomendado para aplicaciones que requieran un alto nivel de seguridad, especialmente cuando se transmiten grandes volúmenes de datos con patrones repetitivos. En dichos escenarios, la falta de difusión entre bloques puede revelar información sobre la estructura del mensaje original. Sin embargo, en el contexto de ARISR, este modo se utiliza principalmente para:

- Cifrar mensajes de longitud reducida, típicos en aplicaciones IoT.
- Derivar la firma de protocolo ARIS en redes privadas.

- Proporcionar una capa básica de confidencialidad en entornos con recursos computacionales limitados.

B.2 Esquema de cifrado en ARISR

El proceso de cifrado en ARISR sigue un esquema estructurado que garantiza la correcta manipulación de los datos, incluyendo el manejo de longitudes variables mediante el esquema de relleno PKCS#7.

B.2.1. Relleno PKCS#7

Dado que AES opera exclusivamente sobre bloques de 16 bytes, los mensajes de longitud arbitraria deben ser rellenados para alcanzar un múltiplo de dicho tamaño. ARISR implementa el esquema PKCS#7, ampliamente utilizado en estándares criptográficos, que añade tantos bytes como sean necesarios, cada uno con el valor igual al número de bytes añadidos.

```
1 // Calcular el valor de relleno segun PKCS#7
2 const ARISR_UINT8 pad_value = AES_BLOCKLEN - (input_len %
      AES_BLOCKLEN);
3 const ARISR_UINT32 padded_len = input_len + pad_value;
4
5 // Aplicar el relleno al final del mensaje
6 memcpy(padded_data, input, input_len);
7 memset(padded_data + input_len, pad_value, pad_value);
```

Por ejemplo, si un mensaje tiene 13 bytes, se añadirán 3 bytes con valor 0x03. Si el mensaje ya es múltiplo de 16, se añade un bloque completo de 16 bytes con valor 0x10, lo que permite distinguir inequívocamente el final del mensaje durante el descifrado.

B.2.2. Proceso de cifrado

El cifrado de datos en ARISR sigue los siguientes pasos:

1. **Validación de parámetros**: Se verifica que los punteros de entrada y salida sean válidos y que la longitud del mensaje sea mayor que cero.
2. **Cálculo del relleno**: Se determina el número de bytes necesarios para completar un bloque de 16 bytes según PKCS#7.
3. **Asignación de memoria**: Se reserva espacio para el mensaje rellenado.
4. **Aplicación del relleno**: Se copia el mensaje original y se añaden los bytes de relleno.
5. **Cifrado bloque a bloque**: Se procesa cada bloque de 16 bytes independientemente utilizando el modo ECB.
6. **Devolución del resultado**: Se transfiere la propiedad del buffer cifrado al llamante.

B.2.3. Implementación del cifrado

La función de cifrado implementada en ARISR es la siguiente:

```
ARISR_ERR ARISR_aes_data_encrypt(const ARISR_AES128_KEY key,
                                 const ARISR_UINT8 *input,
                                 ARISR_UINT32 input_len,
                                 ARISR_UINT8 **output,
                                 ARISR_UINT32 *output_len)
{
    ARISR_UINT32 offset;
    ARISR_UINT8 *padded_data;

    // Validacion de parametros de entrada
    if (!input || input_len == 0 || !output || !output_len) {
        return kARISR_ERR_INVALID_ARGUMENT;
    }

    // Calcular relleno PKCS#7
    const ARISR_UINT8 pad_value = AES_BLOCKLEN - (input_len %
       AES_BLOCKLEN);
    const ARISR_UINT32 padded_len = input_len + pad_value;

    // Verificar desbordamiento de buffer
    if (padded_len < input_len) {
        return kARISR_ERR_BUFFER_OVERFLOW;
    }

    // Asignar memoria para datos con relleno
    padded_data = (ARISR_UINT8 *)malloc(padded_len);
    if (!padded_data) {
        return kARISR_ERR_GENERIC;
    }

    // Preparar texto plano con relleno
    memcpy(padded_data, input, input_len);
    memset(padded_data + input_len, pad_value, pad_value);

    // Inicializar contexto AES
    struct AES_ctx ctx;
    AES_init_ctx(&ctx, key);

    // Cifrar en modo ECB (bloque a bloque)
    for (offset = 0; offset < padded_len; offset += AES_BLOCKLEN) {
        AES_ECB_encrypt(&ctx, padded_data + offset);
    }

    // Transferir propiedad del buffer al llamante
    *output = padded_data;
    *output_len = padded_len;

    return kARISR_OK;
}
```

B.3 Esquema de descifrado en ARISR

El proceso de descifrado es simétrico al cifrado, siguiendo los pasos inversos y realizando una validación rigurosa del relleno PKCS#7 para garantizar la integridad de los datos recuperados.

B.3.1. Proceso de descifrado

1. **Validación de parámetros**: Se verifica que los punteros sean válidos y que la longitud del mensaje cifrado sea múltiplo de 16 bytes.
2. **Asignación de memoria**: Se reserva espacio para los datos descifrados.
3. **Descifrado bloque a bloque**: Se procesa cada bloque de 16 bytes independientemente.
4. **Validación del relleno**: Se extrae el valor del último byte y se verifica que todos los bytes de relleno sean consistentes.
5. **Recorte del relleno**: Se eliminan los bytes añadidos durante el cifrado.
6. **Devolución del resultado**: Se transfiere la propiedad del buffer descifrado al llamante.

B.3.2. Implementación del descifrado

La función de descifrado implementada en ARISR es la siguiente:

```
ARISR_ERR ARISR_aes_data_decrypt(const ARISR_AES128_KEY key,
                                 const ARISR_UINT8 *input,
                                 ARISR_UINT32 input_len,
                                 ARISR_UINT8 **output,
                                 ARISR_UINT32 *output_len)
{
    ARISR_UINT32 i, original_len;
    ARISR_UINT8 pad, *decrypted_data, *resized;

    // Validacion estricta de argumentos
    if (!input || input_len == 0 || input_len % AES_BLOCKLEN != 0
        || !output || !output_len) {
        return kARISR_ERR_INVALID_ARGUMENT;
    }

    // Asignar memoria para datos descifrados
    decrypted_data = malloc(input_len);
    if (!decrypted_data) {
        return kARISR_ERR_GENERIC;
    }

    // Copiar datos cifrados al buffer
    memcpy(decrypted_data, input, input_len);

    // Inicializar contexto AES
    struct AES_ctx ctx;
    AES_init_ctx(&ctx, key);

    // Descifrar en modo ECB (bloque a bloque)
    for (ARISR_UINT32 i = 0; i < input_len; i += AES_BLOCKLEN) {
        AES_ECB_decrypt(&ctx, decrypted_data + i);
    }

    // Validacion de relleno PKCS#7
    // ----------------------------
    // 1. Obtener valor de relleno del ultimo byte
    pad = decrypted_data[input_len - 1];
```

```
37
38      // 2. Validar rango del relleno (1-16 para AES-128)
39      if (pad == 0 || pad > AES_BLOCKLEN) {
40          free(decrypted_data);
41          return kARISR_ERR_INVALID_PADDING;
42      }
43
44      // 3. Verificar que todos los bytes de relleno coincidan con el
            valor
45      for (i = 1; i <= pad; ++i) {
46          if (decrypted_data[input_len - i] != pad) {
47              free(decrypted_data);
48              return kARISR_ERR_INVALID_PADDING;
49          }
50      }
51
52      // Calcular longitud real sin relleno
53      original_len = input_len - pad;
54
55      // Optimizar uso de memoria redimensionando el buffer
56      resized = realloc(decrypted_data, original_len);
57      if (!resized) {
58          free(decrypted_data);
59          return kARISR_ERR_GENERIC;
60      }
61
62      // Establecer parametros de salida
63      *output = resized;
64      *output_len = original_len;
65
66      return kARISR_OK;
67  }
```

B.4 Ejemplo de uso

A continuación se presenta un ejemplo completo de cómo utilizar las funciones de cifrado y descifrado en una aplicación ARISR:

```
1   #include "lib_arisr_crypt.h"
2   #include <stdio.h>
3   #include <string.h>
4
5   int main() {
6       ARISR_ERR err;
7       ARISR_AES128_KEY key;
8       ARISR_UINT8 *ciphertext = NULL;
9       ARISR_UINT8 *plaintext = NULL;
10      ARISR_UINT32 ciphertext_len, plaintext_len;
11
12      // Mensaje original a cifrar
13      const char *original_msg = "Mensaje secreto para ARISR";
14      ARISR_UINT32 original_len = strlen(original_msg) + 1; //
            Incluir terminador nulo
15
16      // Clave AES-128 de ejemplo (16 bytes)
17      const ARISR_AES128_KEY example_key = {
18          0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6,
```

```
        0xab, 0xf7, 0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c
    };
    memcpy(key, example_key, AES_BLOCKLEN);

    printf("Mensaje original (%d bytes): %s\n", original_len,
        original_msg);

    // Cifrar el mensaje
    err = ARISR_aes_data_encrypt(key, (ARISR_UINT8*)original_msg,
        original_len,
                                 &ciphertext, &ciphertext_len);
    if (err != kARISR_OK) {
        printf("Error al cifrar: %d\n", err);
        return 1;
    }

    printf("Mensaje cifrado (%d bytes): ", ciphertext_len);
    for (ARISR_UINT32 i = 0; i < ciphertext_len; i++) {
        printf("%02x ", ciphertext[i]);
    }
    printf("\n");

    // Descifrar el mensaje
    err = ARISR_aes_data_decrypt(key, ciphertext, ciphertext_len,
                                 &plaintext, &plaintext_len);
    if (err != kARISR_OK) {
        printf("Error al descifrar: %d\n", err);
        free(ciphertext);
        return 1;
    }

    printf("Mensaje descifrado (%d bytes): %s\n", plaintext_len,
        plaintext);

    // Liberar memoria
    free(ciphertext);
    free(plaintext);

    return 0;
}
```

B.5 Consideraciones de seguridad

Al implementar cifrado AES-128-ECB en ARISR, es fundamental tener en cuenta las siguientes consideraciones:

- **Gestión segura de claves**: Las claves deben almacenarse en zonas de memoria protegidas y, en sistemas con mayores requisitos de seguridad, pueden derivarse mediante mecanismos como PBKDF2.
- **Limitaciones del modo ECB**: Para aplicaciones que requieran transmitir grandes volúmenes de datos con patrones repetitivos, se recomienda evaluar la migración a modos más seguros como CBC o GCM en futuras versiones.
- **Validación del relleno**: La validación PKCS#7 debe realizarse con cuidado para evitar vulnerabilidades de tipo *padding oracle*.

- **Contexto de uso**: El cifrado en ARISR está diseñado como una capa básica de confidencialidad. En escenarios de alta seguridad, se recomienda complementarlo con mecanismos de autenticación adicionales en capas superiores.

B.6 Integración con el protocolo ARISR

Dentro del flujo completo del protocolo ARISR, el cifrado AES-128 se aplica exclusivamente al campo `Data` de cada *chunk*. Esta aproximación permite:

- Proteger la información sensible del usuario sin afectar los campos de control necesarios para el encaminamiento.
- Mantener la interoperabilidad entre dispositivos que no soporten cifrado, ya que el campo `ARIS` y los CRC permanecen sin cifrar.
- Derivar la firma de red privada a partir de la misma clave, creando un mecanismo unificado de autenticación y confidencialidad.

La implementación presentada en este capítulo forma parte de la librería `lib-protoarisr-c` y ha sido validada mediante pruebas unitarias que verifican tanto el correcto funcionamiento del cifrado como la robustez frente a entradas malformadas o ataques de relleno.