PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE INGENIERÍA

CARRERA DE: TECNOLOGÍAS DE LA INFORMACIÓN

Trabajo de Titulación

TEMA:

Propuesta y diseño de un prototipo de sistema de comunicación multi-nodo para microcontroladores ESP32 con LoRa en una red *mesh*

AUTOR:

Danilo Sebastián Arregui Almeida

DIRECTOR:

Juan Francisco Chafla Altamirano

QUITO DM, OCTUBRE DE 2024

DEDICATORIA

A mi familia, por su amor incondicional, su apoyo constante y por ser mi fuerza en cada paso. Esta meta también es suya.

AGRADECIMIENTO

Quiero expresar mi más sincero agradecimiento a mi querida madre, quien ha sido un pilar fundamental a lo largo de toda mi vida académica. Su apoyo constante, sus consejos y su confianza incondicional han sido esenciales para alcanzar esta meta.

A mi hermano, gracias por colaborar activamente en el proceso de evaluación y pruebas de este proyecto. Su disposición y ayuda fueron valiosas en momentos clave. Además, agradezco profundamente sus consejos, su experiencia en la elaboración de artículos científicos me brindó claridad y dirección en varios momentos del desarrollo de este trabajo.

Agradezco profundamente al Ing. Juan Francisco Chafla, mi tutor, por su compromiso, acompañamiento y disposición para guiarme, resolver mis inquietudes y asegurar el avance constante de este trabajo.

Finalmente, extiendo mi gratitud a todos los docentes de la Pontificia Universidad Católica del Ecuador, en especial a quienes conforman el área de Ingeniería, por su excelencia profesional y por brindar una formación centrada no sólo en la teoría, sino en el aprendizaje significativo y en experiencias prácticas que marcaron la diferencia en mi formación.

RESUMEN

LoRa es una tecnología de comunicación inalámbrica de largo alcance y bajo consumo energético, ampliamente empleada en soluciones de Internet de las Cosas (IoT). No obstante, su implementación convencional mediante LoRaWAN presenta restricciones operativas debido a su topología centralizada y dependencia de *gateways*. Ante esta limitación, se planteó el desarrollo de un sistema de comunicación descentralizado, funcional en dispositivos con recursos limitados. Se diseñó e implementó un sistema multi-nodo basado en ESP32 y modulación LoRa, bajo una topología *mesh* con mecanismos propios de enrutamiento por vecinos, confirmación *hop-by-hop* y retransmisiones controladas. El prototipo, estructurado modularmente, fue probado en tres escenarios con distintas condiciones, escenario ideal, con nodo intermedio caído y entorno urbano. Los resultados demostraron un comportamiento robusto, con tasas de entrega de paquetes (PDR) de hasta 95 % en entornos controlados y 75 % en condiciones urbanas. En el escenario con fallo, se alcanzó una tasa de recuperación ante fallo (PRR) promedio de 88,3 %, validando la capacidad de reconvergencia del sistema. Estos hallazgos confirman que es posible implementar un sistema de comunicación efectivo y tolerante a fallos sobre microcontroladores, con aplicaciones prácticas en redes de monitoreo ambiental, agricultura inteligente, sistemas IoT distribuidos, entre otros.

Palabras clave: LoRa, ESP32, red *mesh*, IoT.

CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

LISTA DE ABREVIATURAS

ACK	*Acknowledgment* (Acuse de recibo)
ADR	*Adaptive Data Rate* (Tasa de Datos Adaptativa)
ALT	*Alternative* (Alternativo)
AODV	*Ad hoc On-Demand Distance Vector Routing* (Enrutamiento de Vectores de Distancia Bajo Demanda ad hoc)
CSMA/CA	*Carrier Sense Multiple Access with Collision Avoidance* (Acceso Múltiple por Detección de Portadora con Prevención de Colisiones)
CSS	*Chirp Spread Spectrum* (Espectro ensanchado por *chirp*)
EXP	*Experimental bits* (Bits experimentales)
FIFO	*First In First Out* (Primero en Entrar, Primero en Salir)
GPIO	*General Purpose Input/Output* (Entrada/Salida de Propósito General)
IoT	*Internet of Things* (Internet de las Cosas)
LBT	*Listen Before Talk* (Escuchar Antes de Hablar)
LoRa	*Long Range* (Larga distancia)
LoRaWAN	*Long Range Wide Area Network* (Red de Área Amplia de Largo Alcance)
LPWAN	*Low Power Wide Area Network* (Red de Área Amplia de Bajo Consumo)
MPLS	*Multiprotocol Label Switching* (Conmutación de Etiquetas de Protocolos Múltiples)
OLSR	*Optimized Link State Routing* (Enrutamiento de Estado de Enlace Optimizado)
OSI	*Open Systems Interconnection* (Interconexión de Sistemas Abiertos)
PDR	*Packet Delivery Ratio* (Tasa de entrega de paquetes)
PHY	*Physical Layer* (Capa Física)
PLC	*Programmable Logic Controller* (Controlador Lógico Programable)
PRR	*Packet Restoration Ratio* (Tasa de recuperación ante fallo)
QoS	*Quality of Service* (Calidad de servicio)
RERR	*Router Error* (Error de Ruta)
RFID	*Radio Frequency Identification* (Identificación por Radiofrecuencia)
RREP	*Route Reply* (Respuesta de Ruta)
RREQ	*Route Request* (Requerimiento de Ruta)

RSSI	*Received Signal Strength Indicator* (Indicador de Fuerza de la Señal Recibida)
SF	*Spreading Factor* (Factor de Dispersión)
TCP/IP	*Transmission Control Protocol / Internet Protocol* (Protocolo de Control de Transmisión / Protocolo de Internet)
TDMA	*Time Division Multiple Access* (Acceso Múltiple por División de Tiempo)
TTL	*Times To Live* (Tiempo de vida)
USB	*Universal Serial Bus* (Bus Serie Universal)
Wi-Fi	*Wireless Fidelity* (Fidelidad Inalámbrica)

CAPÍTULO I: INTRODUCCIÓN

El constante avance del Internet de las Cosas (IoT) ha impulsado el desarrollo de sistemas de comunicación inalámbricos de bajo consumo energético y larga distancia, como la modulación *Long Range* (LoRa). Sin embargo, las soluciones existentes, como las conexiones punto a punto y el protocolo *Long Range Wide Area Network* (LoRaWAN), presentan limitaciones para aplicaciones que requieren redes multi-nodo ligeras y flexibles, especialmente cuando se utilizan microcontroladores con recursos limitados como el ESP32.

Este capítulo presenta la justificación del proyecto, el planteamiento del problema a resolver, los objetivos guía del trabajo y el alcance que delimita su ejecución, estableciendo así el marco inicial para el desarrollo de un prototipo de sistema de comunicación ligero y adaptable a las características de *hardware* del ESP32 utilizando LoRa

1.1. Justificación

Los microcontroladores ESP32 con módulos LoRa son muy utilizados en aplicaciones IoT de distintas industrias, tales como agrícola, energética, seguridad, etc., debido a su bajo costo y fácil configuración. Sin embargo, las alternativas de comunicación usando la modulación LoRa, tales como las conexiones punto a punto o el estándar LoRaWAN, no resultan adecuadas para proyectos que requieren escalabilidad y flexibilidad, especialmente cuando se trabaja con un *hardware* limitado. Actualmente, no existe un sistema de comunicación multi-nodo suficientemente ligero, simple y que, además, no sobrecargue los recursos de *hardware*.

El desarrollo de este prototipo de sistema de comunicación proporcionará una solución ligera que permita la interacción eficiente entre varios nodos ESP32 en una red multi-nodo. Este trabajo de titulación no se limitará a una única aplicación, sino que proporcionará una base sólida que permita a futuros desarrolladores implementar capas de aplicación diversas sobre el sistema. Adicionalmente, se podrá adaptar a cualquier escenario IoT, llegando a ser útil para cualquier industria como la agricultura de precisión, monitoreo ambiental o automatización de infraestructuras.

El sistema se probará en una red *mesh* básica, lo cual asegurará su viabilidad en dispositivos con limitaciones de *hardware*. Además, este trabajo ofrecerá una alternativa escalable y personalizable para redes distribuidas. Así, se podrá establecer las bases técnicas para futuras expansiones y aplicaciones, las cuales requieran una capa de aplicación especifica, facilitando en consecuencia el desarrollo de soluciones IoT escalables y eficientes.

1.2. Planteamiento del problema

En las redes de comunicación que utilizan microcontroladores ESP32 con módulos LoRa, las opciones existentes están mayormente limitadas a conexiones estáticas punto a punto o a configuraciones específicas que no ofrecen la flexibilidad necesaria para redes de múltiples nodos y de propósitos varios. El uso del estándar LoRaWAN tampoco es una solución adecuada para proyectos con *hardware* limitado, debido a su complejidad y requisitos de recursos. Además, la falta de disponibilidad de un sistema de comunicación ligero que permita crear redes *mesh* escalables en estos microcontroladores de bajo costo, impide su uso en aplicaciones distribuidas más complejas.

Con base a esta problemática centrada en la limitación de opciones de comunicación Wireless LoRa, surge la pregunta principal para el trabajo:

- ¿Cuál sería un diseño adecuado para un prototipo de sistema de comunicación multi-nodo ligero y eficiente para microcontroladores ESP32 con LoRa que asegure la transmisión de datos en redes *mesh*, sin exceder las limitaciones de *hardware*?

Preguntas secundarias:

- ¿Qué limitaciones de *hardware* se deben considerar al desarrollar este prototipo de sistema para ESP32?
- ¿Cuáles son los principales desafíos al implementar un prototipo de sistema de comunicación multi-nodo en redes *mesh* con LoRa?

Este trabajo de titulación proporcionará una solución que permita la comunicación entre nodos en redes *mesh*, estableciendo una base sólida para la implementación de futuras aplicaciones IoT, sin barreras de conexiones estáticas y las limitaciones del *hardware* disponible en microcontroladores comerciales.

1.3. Objetivos

1.3.1. Objetivo General

Desarrollar un prototipo de sistema de comunicación multi-nodo para microcontroladores ESP32 con modulación LoRa, que permita la interacción entre nodos, transmisiones y recepciones de datos ligeros en una red *mesh*.

1.3.2. Objetivos Específicos

- Describir el marco teórico y conceptual relacionado con la modulación LoRa, el protocolo LoRaWAN, las características de los microcontroladores ESP32 y las redes *mesh*, estableciendo las bases técnicas para el diseño del sistema de comunicación.
- Diseñar la arquitectura y un modelo básico de comunicación que permita la interacción entre nodos y facilite la escalabilidad en la red *mesh*.
- Desarrollar un prototipo de sistema de comunicación multi-nodo que permita la transmisión y recepción de datos ligeros.
- Probar y analizar el prototipo en una red *mesh* básica, verificando el envío y recepción de distintos tipos de paquetes de datos y de control, así como la capacidad

de reconvergencia de la red, con el fin de confirmar la estabilidad y efectividad en la interacción de la red.

1.4. Alcance

En este trabajo de titulación se desarrollará un prototipo de sistema de comunicación multi-nodo para microcontroladores ESP32 utilizando modulación LoRa, y se probará en una topología de red *mesh* básica. Se describirá la selección del *hardware* ESP32 y su capacidad de conectividad, con el objetivo de contextualizar los recursos disponibles.

Posteriormente, se diseñará e implementará la arquitectura de red básica, compuesta por la selección del *hardware* de red, el desarrollo del sistema de comunicación y la topología, que permitirá la interacción entre varios nodos; lo cual facilitará la transmisión y recepción de datos en la red *mesh* propuesta. Se realizarán pruebas del prototipo en escenarios controlados para verificar el buen funcionamiento de la red. Se evaluarán aspectos fundamentales como el envío y recepción de distintos tipos de paquetes y la capacidad de reconvergencia de la red; lo cual asegurará la integridad y estabilidad de la comunicación.

No se abordarán mediciones de latencia, *jitter* ni tiempo de retransmisión, ya que estas requieren mecanismos de sincronización entre nodos para obtener resultados precisos, y el prototipo se enfoca en un sistema descentralizado que opera de forma asincrónica. Tampoco se discutirán aspectos avanzados como alteraciones de bits, pérdidas de información ni evaluaciones de Calidad de Servicio (QoS), debido a que implican procesos de análisis altamente demandantes en tiempo, instrumentación especializada y una complejidad que excede el alcance del presente proyecto.

Así mismo, no se evaluarán elementos relacionados con la física de la modulación de radiofrecuencia, propagación de ondas electromagnéticas, calidad de señal, interferencia, ruido en la transmisión, reflexiones, pruebas en entornos rurales o pruebas extendidas en un área amplia urbana, debido a limitaciones de recursos, tiempo y las propias restricciones del *hardware*.

Adicionalmente, no se evaluará la capacidad máxima de nodos que la red puede soportar, ya que el número de dispositivos disponibles para las pruebas es limitado. Finalmente, no se discutirá la selección de los microcontroladores, pues estos fueron elegidos únicamente en base a su disponibilidad en el comercio local.

CAPÍTULO II: MARCO TEÓRICO Y CONCEPTUAL

Este capítulo presenta el marco teórico y conceptual que fundamenta el desarrollo del sistema multimodo basado en microcontroladores ESP32 y modulación LoRa. Inicialmente se aborda la evolución del IoT como contexto ante la necesidad de sistemas inalámbricos flexibles y ligeros. Posteriormente se describe a la tecnología LoRa, una solución de radiofrecuencia para comunicaciones inalámbricas, utilizadas en IoT por su largo alcance y poco consumo energético.

También, se aborda el protocolo LoRaWAN, el protocolo de red de la modulación LoRa, que facilita la conexión de múltiples dispositivos a través de *gateways*. Sin embargo, por su complejidad y arquitectura centralizada puede no ser adecuado para sistemas que requieren descentralización. Además, se presenta la arquitectura de redes *mesh*, la misma que ofrece una comunicación distribuida y descentralizada entre varios nodos, aumentando la resiliencia y adaptabilidad para redes inalámbricas. Finalmente, se detallan las características de los microcontroladores ESP32, su integración con tecnologías de comunicación inalámbrica y su utilidad en entornos IoT.

2.1. Antecedentes

En los últimos años el área del IoT ha avanzado significativamente y ha ganado relevancia en el ámbito tecnológico y empresarial, al convertirse en una herramienta clave para la automatización, optimización de procesos y creación de servicios inteligentes. Cada año más dispositivos se acoplan a este entorno y se conectan entre ellos y con sistemas externos. En este contexto, si bien tecnologías como *Wireless Fidelity* (*Wi-Fi*) o *Bluetooth* han sido ampliamente utilizadas, han comenzado a surgir otras alternativas más adecuadas para ciertos entornos. Particularmente, tecnologías inalámbricas de baja potencia y largo alcance, como LoRa.

La modulación LoRa es comúnmente utilizada con microcontroladores como el ESP32, pero para asegurar la comunicación de estos en entornos IoT, las implementaciones se limitan mayormente a configuraciones punto a punto o al uso del estándar LoRaWAN. Si bien LoRaWAN es una solución efectiva para ciertas aplicaciones, su complejidad y requerimiento de recursos la hacen inapropiada para entornos de *hardware* con limitaciones estrictas, como los microcontroladores ESP32.

Actualmente, existe una carencia de sistemas ligeros y eficientes que permitan una comunicación multi-nodo en una red *mesh* utilizando estos microcontroladores. Un ejemplo de ello es la propia librería proporcionada por Heltec Automation (2019) para gestionar las comunicaciones LoRa, la cual se centra en funciones básicas de envío y recepción de datos punto a punto, así como en la configuración del módulo LoRa. Sin embargo, no contempla formas más complejas o flexibles de comunicación necesarias para entornos dinámicos y con múltiples nodos.

La falta de flexibilidad y escalabilidad en las soluciones existentes limita el uso de ESP32 con LoRa a aplicaciones simples o que requieren un manejo centralizado, impidiendo su implementación en proyectos más complejos de redes distribuidas. Esto se traduce en restricciones para desarrolladores e investigadores que desean aplicar estas tecnologías en escenarios como la agricultura de precisión, el monitoreo ambiental y la automatización de infraestructuras, donde la interoperabilidad y la robustez de la red son esenciales.

Por otra parte, una opción muy viable y ampliamente utilizada en entornos IoT son las redes malla, como las redes ad hoc, que generalmente emplean *Wi-Fi* como medio de comunicación. Las redes *mesh* han demostrado su eficacia como una alternativa sólida para

la comunicación resiliente y distribuida en redes IoT, proporcionando una estructura adaptable y robusta para la interacción entre múltiples nodos (Qiu et al., 2017). Estas características hacen de las redes *mesh* una opción ideal para superar las limitaciones de las configuraciones tradicionales en microcontroladores con recursos limitados. Sin embargo, no existe una solución equivalente ampliamente adoptada para microcontroladores de bajo costo que empleen tecnología LoRa. Por esta razón se evidencia una necesidad de exploración y desarrollo en este ámbito.

Este trabajo de titulación busca llenar ese vacío mediante el diseño de un prototipo de sistema de comunicación multi-nodo para ESP32 con LoRa, sirviendo como una base modular que optimice la transmisión de datos en redes *mesh* y permita su implementación en escenarios que requieren flexibilidad y bajo uso de *hardware*. Además, se pretende que este prototipo actúe como una capa de comunicación que pueda ser empleada por futuros desarrollos, permitiendo a los desarrolladores integrar y adaptar diversas capas de aplicación para escenarios específicos de IoT.

2.2. Internet de las Cosas (IoT)

El internet de las cosas es un paradigma tecnológico vanguardista que busca integrar objetos físicos cotidianos dentro de las redes digitales, de tal forma que estos objetos puedan identificarse, comunicarse y cooperar entre sí para lograr metas comunes tales como automatización de procesos, ofrecer servicios o simplemente mejorar la funcionalidad de dichos objetos. Este paradigma tiene tres enfoques principales: enfoque orientado a los objetos, enfoque orientado a la red y enfoque orientado a lo semántico.

El enfoque orientado a los objetos principalmente abarca al término de lo que se conoce como objetos inteligentes que son aquellos que tienen la capacidad de recolectar datos con sensores, reaccionar mediante actuadores e identificarse ya sea mediante etiquetas de radiofrecuencia, identificadores universales u otros. Por otro lado, el segundo enfoque que se centra en la red tiene como objetivo asegurar la conectividad de los objetos a través de protocolos y redes eficientes. Este enfoque abarca el concepto de conectividad ubicua que hace relación a la conectividad de cualquier objeto sin importar su tamaño y potencia, involucrando el uso de tecnologías de comunicación de bajo consumo y la integración con redes existentes como lo es *Wi-Fi*, *Bluetooth*, LoRa, *Zigbee*, etc. Finalmente, el enfoque semántico busca dar sentido, estructura e inteligencia a los datos generados por los objetos conectados, mediante búsqueda, filtrado y organización de dichos datos (Atzori et al., 2010).

En cuanto al impacto del IoT, este se ha establecido como una de las tendencias y paradigmas tecnológicos más transformadores de la última década, generando impactos positivos en distintas áreas de la industria y mercados. El crecimiento de la densidad de dispositivos IoT ha permitido mejorar operaciones, crear nuevos modelos de negocio y aumentar eficiencia de sectores como la manufactura, salud, transporte, agrícola y servicios. Además, esta tendencia no sólo se enfoca en los dispositivos físicos sino también en la analítica en la nube y herramientas de visualización (Dahlqvist et al., 2019).

Debido a este paradigma, se ha visto la incorporación de distritos tipos de dispositivos al ecosistema IoT, ya sea equipos de usuario final como celulares, laptops y *tablets*, hasta dispositivos como *gateways*, infraestructura en la nube o servidores. No obstante, también destaca la inclusión de microcontroladores, los cuales son útiles para implementar sistemas embebidos simples en un solo *chip*, con un alto rendimiento y un costo muy bajo. Esto ampliando las posibilidades del IoT, añadiendo una capa de inteligencia a tareas de automatización, comunicación entre máquinas y hasta recolección de datos en tiempo real (Wu et al., 2020).

2.3. Microcontroladores ESP32

El ESP32 es un microcontrolador versátil y potente que se diferencia por su integración *Wi-Fi* y *Bluetooth* en un *chip* único, convirtiéndolo en una solución viable para distintas aplicaciones dentro del ecosistema IoT. Diseñado para poder funcionar incluso en entornos industriales (Espressif Systems, 2025b), este microcontrolador está equipado con un procesador de dos núcleos Tensilica Xtensa LX6, que permite procesamiento de tareas demandantes o múltiples.

También, incorpora conectividad *Wi-Fi* y *Bluetooth* de forma nativa, compatible con los estándares 802.11 b/g/n, así como con *Bluetooth Classic* y *Bluetooth Low Energy*. Adicionalmente, posee varios pines de entrada y salida de propósito general (GPIO) para conexión de módulos externos, cuya distribución puede visualizarse en la Figura 1. El módulo está diseñado para poder ser programado principalmente en C++ ya sea con Arduino IDE o con PlatformIO (Hercog et al., 2023).

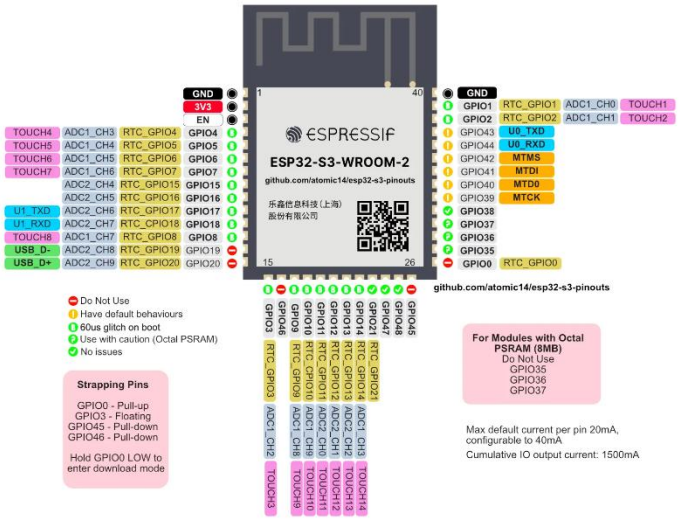

Figura 1 Mapa de Pines GPIO del módulo ESP32. Tomado de Greening (2023)

En primer lugar, debido a sus características técnicas, el ESP32 se ha convertido en una solución ampliamente utilizada en proyectos IoT, a pesar de poseer menos potencia y ser mucho más económica que una computadora convencional. Es ideal para cualquier industria que requiera automatización, monitoreo remoto o lectura de sensores en tiempo real debido a su flexibilidad en cuanto a conectividad inalámbrica, bajo consumo energético y su simplicidad en cuanto a configuración y programación. Es comúnmente utilizado en sistemas de domótica y casas inteligentes, automatización industrial, estaciones meteorológicas, monitoreo, agricultura y hasta educación (Samodya, 2023).

Adicionalmente, existe una amplia gama de variedades de módulos como el ESP32 clásico, ESP32-S2, ESP32-S3, ESP32-C, ESP32-H. Cada una de estas variedades posee características específicas que permiten su integración en diversas aplicaciones, ya sea conectores de antena externos, soporte nativo para USB, *chips* aceleradores para inteligencia artificial, entre otras (Espressif Systems, 2025a).

Es importante mencionar que la introducción de dispositivos ESP32 en distintas industrias y empresas se ha visto influenciada por los altos costos de los controladores lógico-programables (PLC) disponibles en el mercado, así como el *software* licenciado restrictivo que suelen requerir (Murillo Oviedo et al., 2024). Por esta razón los ESP32 se han convertido en una solución viable por su bajo costo, no depender de *software* licenciado para usarlos o programarlos, la amplia gama de módulos disponibles, además de la disponibilidad de proyectos de código libre compatibles con dichos microcontroladores.

2.4. Tecnología LoRa

LoRa es una tecnología inalámbrica que se ha popularizado en la última década debido a su versatilidad y eficacia. Su nombre proviene del inglés *Long Range* ya que su diferenciador principal con otro tipo de tecnologías Wireless presentes en el mercado es su capacidad de poder transmitir datos a largas distancias, ya sea en entornos urbanos o en entornos rurales, su bajo consumo de energía y la eficiencia para trasmitir pequeños volúmenes de datos (Semtech, 2024a). LoRa inicio su desarrollo entre 2009 y 2010 por la empresa francesa Cycleo para luego ser comprada por Semtech quienes completaron su desarrollo. Siendo esta última empresa quien impulsaría el uso de LoRa para acelerar y simplificar el desarrollo de soluciones IoT y convertir a esta tecnología en parte crucial del ecosistema IoT (Slats, 2020).

2.4.1. Principios

Basándose en el modelo de Interconexión de Sistemas Abiertos (OSI), LoRa trabaja solamente en la capa física (PHY), encargándose únicamente de la transmisión de *bits* a través del aire, siendo este su medio. No se requiere cableado físico ya que se emplean ondas de radio frecuencia para transportar información entre dispositivos IoT y *gateways*, siendo esta una interfaz de comunicación física pura. Lora emplea una técnica de modulación de espectro expandido conocida como *Chirp Spread Spectrum* (CSS), la cual consiste en usar señales llamadas *chirps* que son ondas cuya frecuencia aumenta o disminuye gradualmente durante el tiempo (Semtech, 2024a). Cabe recalcar que LoRa opera en las bandas de frecuencia libres o ISM, específicamente en 433 MHz, 868 MHz o 915 MHz, estas frecuencias varían según la región debido a regulaciones locales. También, su característica principal de funcionamiento a largas distancias se da debido emplear frecuencias menores a 1 GHz, a su vez esto le otorga a LoRa una buena penetración a obstáculos (Torres et al., 2021).

A nivel físico, cada trasmisión de LoRa está compuesta por múltiples *chirps* que recorren de forma cíclica toda la banda de frecuencia seleccionada como se puede ver en la Figura 2. Cada símbolo se codifica mediante una secuencia de *chirps*, cuya variación en frecuencia y posición representa los *bits* transmitidos. Específicamente, la información se codifica en la posición de salto dentro del *chirp*, lo cual permite representar diferentes combinaciones de *bits*. La cantidad de información que se puede transmitir por cada símbolo depende del *Spreading Factor* (SF), que determina cuantos *chirps* se usan para representar

un solo símbolo. Cuanto mayor es el SF, más *chirps* se usan y el símbolo dura más tiempo, reduciendo la velocidad, pero aumentando el alcance de la señal.

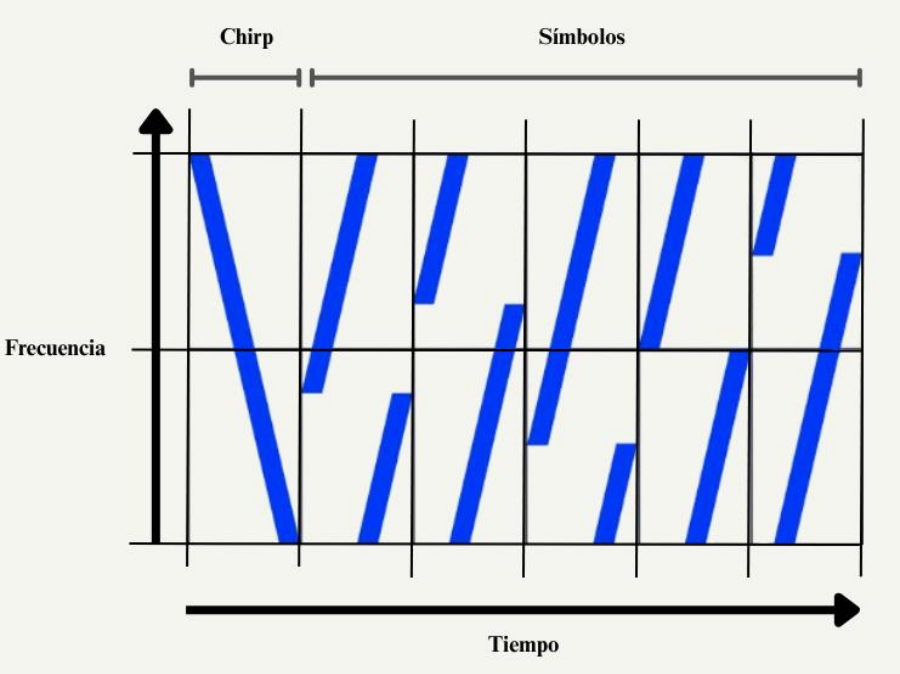

Figura 2 Representación gráfica de chirps en LoRa. Cada chirp corresponde a una variación lineal de frecuencia en función del tiempo. Adaptado de Huang (2020).

Además, LoRa aparte del SF permite ajustar parámetros como ancho de banda y tasa de corrección de errores. El ancho de banda determina que tan amplia es la banda de frecuencia que usa cada *chirp* y la tasa de corrección de errores permite añadir *bits* extra para corregir problemas causados por interferencias. Gracias a esta modulación, se tiene una alta resistencia al ruido, no se requiere relojes de alta precisión en los dispositivos y se tiene resistencia a los cambios de frecuencia por movimiento. Por estas razones LoRa puede detectar señales muy débiles hasta -130 dBm, haciéndola ideal para comunicaciones de largo alcance y bajo consumo (Augustin et al., 2016).

2.4.2. Limitaciones

La principal limitación que posee la tecnología LoRa es que utiliza transceptores *half-duplex*, lo que significa que no puede enviar y recibir al mismo tiempo. Al trabajar sólo en la capa física según el modelo OSI, lo que limita las técnicas de control y corrección, existe una alta probabilidad de que ocurran transmisiones simultáneas y colisiones. LoRa usa un esquema tipo ALOHA, el cual no tiene control de acceso al canal y se transmite sin verificar que el canal esté libre, por lo que cualquier transmisión que empiece antes o durante

otra puede destruir el mensaje. Esto también provoca que en redes muy densas o que incorporen nuevos dispositivos de forma variable, las colisiones ocurran con más frecuencia, afectando la eficiencia y fluidez de la comunicación en la red (Xiao et al., 2024).

Otras limitaciones importantes están relacionadas con las condiciones variables e implicaciones de la comunicación a larga distancia. A pesar de que LoRa es una tecnología diseñada para redes WAN, sus señales pueden debilitarse o incluso perderse debido a la interferencia o a características del entorno, como paredes gruesas, estructuras metálicas o vegetación densa. Si bien LoRa opera en frecuencias bajas y tiene mejor capacidad de penetración frente a obstáculos, esto no significa que todos los obstáculos sean irrelevantes. La acumulación de barreras físicas atenúa progresivamente la señal. En particular, materiales de construcción como el concreto, el vidrio y el metal pueden absorberla o reflejarla, y elementos como el espesor de las paredes, refuerzos o ductos pueden propiciar su pérdida (Islam et al., 2024). Este problema se agrava en entornos urbanos, donde la arquitectura densa y alta interferencia electromagnética reducen su capacidad de mantener comunicación a larga distancia.

Por otro lado, como consecuencia de las pérdidas de señal, reflexiones y colisiones, los dispositivos se ven obligados a reenviar paquetes o actuar distinto a su funcionamiento simple de envío y recepción, dependiendo de cómo están configurados. Esto genera que los dispositivos consuman más energía de la que deban por dichos eventos imprevistos, afectando su autonomía y posiblemente su vida útil (Ayoub Kamal et al., 2023).

Finalmente, debido a la susceptibilidad a colisiones y los problemas de cobertura, la escalabilidad de LoRa presenta importantes desafíos. A medida que se agregan más nodos a una red, especialmente en aquellas con alta densidad de estos, aumenta la posibilidad de transmisiones simultáneas y colisiones. Además, los obstáculos físicos como paredes, árboles o estructuras metálicas, así como la interferencia electromagnética típica de entornos urbanos, dificultan la expansión efectiva de la red. Estas limitaciones reducen la viabilidad de desplegar redes LoRa de gran densidad.

2.5. Tecnología LoRaWAN

LoRaWAN es un protocolo de comunicación para redes amplias y de baja potencia, que funciona en base a la capa física y modulación de radiofrecuencia LoRa. Este protocolo se ha convertido en un estándar para las redes de área amplia de bajo consumo (LPWAN) y

para los dispositivos que manejan modulación LoRa, su especificación está abierta para que cualquiera pueda configurar e interactuar con una red LoRA (AWS, 2025).

El protocolo LoRaWAN fue creado por una organización sin fines de lucro llamada LoRa Alliance, esta fue fundada en 2015 por proveedores y empresas de tecnología como Semtech, Cisco, IBM. Su objetivo era crear un protocolo y establecerlo como estándar abierto para redes IoT de bajo consumo y largo alcance, de esta forma se evitaría que cada empresa manejara protocolos o sistemas propios y se garantizaría la interoperabilidad. Esta organización aprovecho las ventajas de LoRa como su bajo consumo, comunicación a larga distancia y definió una arquitectura basada en nodos finales, *gateways* y servidores (LoRa Alliance, 2025a).

2.5.1. Principios

En primer lugar, al estar basado en la modulación de radiofrecuencia LoRa, LoRaWAN mantiene sus mismas características físicas de transmisión. Sin embargo, este se trata de un protocolo de comunicación que incorpora funcionalidades características de niveles superiores del modelo OSI, como es la gestión de dispositivos, estructuración de trafico de red y control de tasas de datos mediante un esquema conocido como Adaptative Data Rate (ADR). En la Figura 3 se puede observar gráficamente en qué capas del modelo OSI opera LoRaWAN.

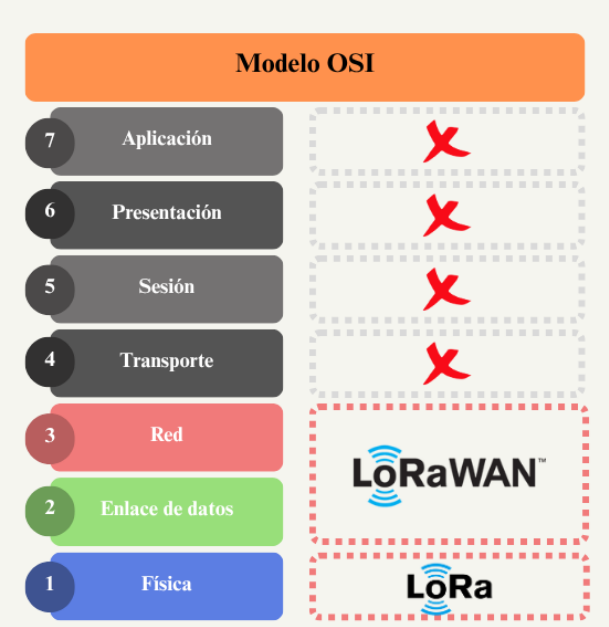

Figura 3 Representación de LoRaWAN en el modelo de referencia OSI. Adaptado de Baltuille (2023)

Este mecanismo permite que un servidor ajuste dinámicamente la potencia de transmisión y la tasa de datos de cada nodo de la red, en función de su distancia, condiciones del canal y señal. De esta forma se asigna tasas más altas a dispositivos cercanos y más bajas a aquellos alejados, reduciendo el tiempo de las transmisiones y disminuyendo la probabilidad de colisiones. Además, optimiza el consumo energético de los dispositivos al evitar el uso innecesario de potencia de transmisión.

Como se muestra en la Figura 4, LoRaWAN se caracteriza por trabajar con una arquitectura preestablecida, esta se basa en una topología estrella de estrellas, donde los dispositivos finales o nodos no se comunican entre ellos, sino mandan los datos a *gateways* LoRa. Estos *gateways* trabajan como intermediarios y trasmiten los datos por conexiones IP a un servidor de red, este servidor centraliza la gestión de tráfico, identificación de emisor y envío de datos al servidor de aplicación correspondiente. La comunicación que emplea LoRa se centra en nodos y *gateways* que es de un solo salto y por lo general los nodos son los que envían datos, es decir se tiene una comunicación ascendente. Sólo cuando el servidor necesita realizar ajustes se envía instrucciones a los nodos LoRa, así lo hacen los procesos asociados al esquema ADR (LoRa Alliance, 2025b).

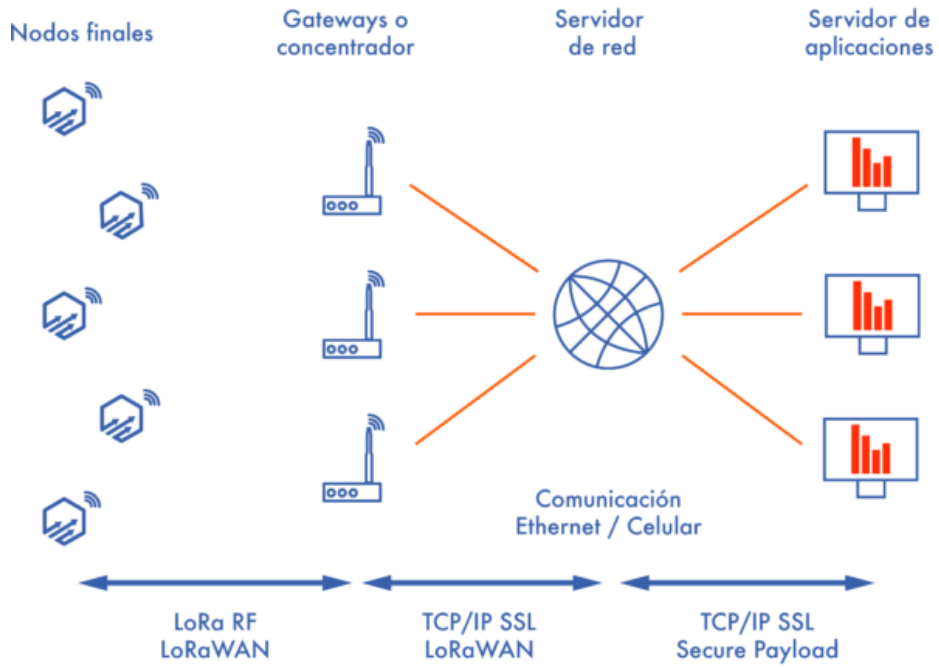

Figura 4 Arquitectura general de una red LoRaWAN. Tomado de Easy Automation (2023).

En cuanto a los dispositivos finales o nodos, LoRaWAN define tres tipos de configuración para los dispositivos finales denominados Clases A, B, C. Estas clases y su

comportamiento está definido por el *firmware* del dispositivo y no puede ser modificado por medio del servidor de red o aplicación (Semtech, 2024b). Cabe mencionar que cada una está enfocada a distintos requerimientos energéticos, fluidez y sentido de comunicación como se muestra en la Figura 5.

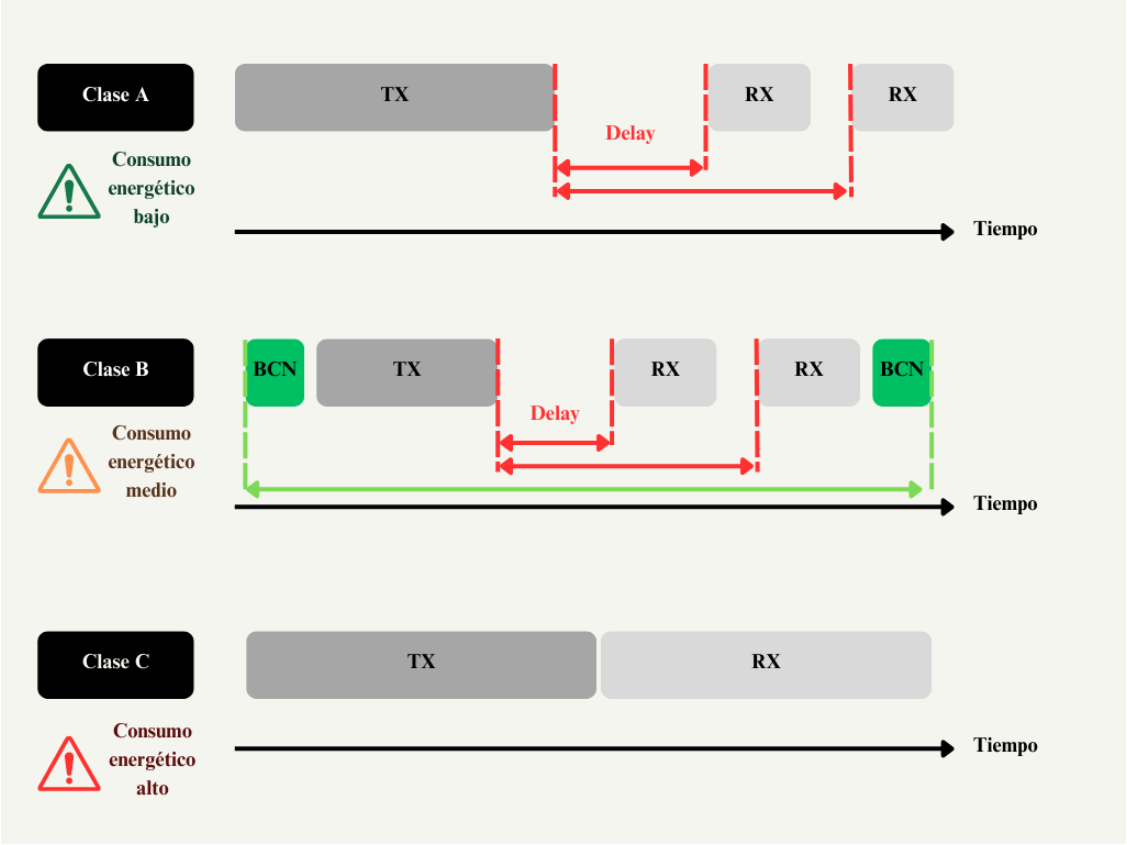

Figura 5 Funcionamiento de las clases LoRaWAN, donde TX: Transmisión, RX: Recepción, BCN: Beacon de sincronización. Elaboración propia.

- **Clase A:** Caracterizada por la transmisión en función de las necesidades del dispositivo y por la apertura de dos pequeñas ventanas de recepción tras haber trasmitido, esto hace que consuma muy poca energía debido a que permanecen la mayor parte del tiempo en estado de suspensión. Esta es muy útil para redes que no requieran comunicación descendente frecuente o que las transmisiones hacia los nodos no sean críticas.
- **Clase B:** Similar a la Clase A, pero agrega ventanas de recepción configurables, los dispositivos finales reciben señales de sincronización llamados *Beacons* desde los *gateways* para abrir ventanas de recepción, así permitiendo una mejor sincronización para los mensajes descendentes. Muy útil para implementaciones que requieran

comunicación regular con los dispositivos finales y donde se busque un consumo energético moderado.

- **Clase C:** Caracterizada por mantener una ventana de recepción abierta en casi todo momento, sólo deja de escuchar al canal cuando está trasmitiendo. Esta clase es útil para aplicaciones que requieran una comunicación bidireccional fluida, pero es la clase con mayor consumo energético ya que no se entra en estado de suspensión en ningún momento.

2.5.2. Limitaciones

LoRaWAN, al estar basado sobre la capa física de LoRa, hereda sus principales limitaciones técnicas. Entre estas se encuentra el uso de transceptores *half-duplex*, que impiden trasmitir y recibir al mismo momento; la falta de control de acceso al medio y el uso de esquemas de transmisión sin verificar el canal. Estas limitaciones afectan también a la eficiencia de la comunicación, el consumo energético y escalabilidad de las redes LoRaWAN.

Aunque LoRaWAN es un protocolo de comunicación que permite gestionar la red, organizar el tráfico y coordinar los dispositivos finales, no implementa mecanismos eficaces para evitar colisiones, como el método *Listen Before Talk* (LBT). En su lugar, mantiene el uso del esquema ALOHA, en el cual los dispositivos transmiten sin verificar si el canal está libre, lo que provoca colisiones frecuentes, especialmente en escenarios con alta densidad de nodos (Adelantado et al., 2017). A pesar de que LoRaWAN organiza el tráfico por canales y permite distintas tasas de datos mediante ADR, este enfoque no resuelve completamente la congestión en entornos con muchos nodos o tráfico considerable.

Otra de sus grandes limitaciones es su dependencia de una topología fija de estrella de estrellas, donde los dispositivos finales únicamente se comunican con los *gateways* LoRa cercanos, con solo un salto (LoRa Alliance, 2025b). Esta arquitectura quita flexibilidad al protocolo LoRaWAN, impidiendo que los nodos se comuniquen directa y necesariamente deban pasar por un único *gateway*, el cual luego redirige los datos a través de una infraestructura IP. Esto denota que la red LoRaWAN no es autónoma ni descentralizada, en realidad es una red hibrida, cuyo primer segmento emplea enlaces LoRa y el segundo se gestiona Protocolos de Control de Transmisión/Internet (TCP/IP). Esto restringe la flexibilidad y hace a LoRaWAN menos adaptable a escenarios varios y generando un posible punto único de falla entre el Gateway y el servidor de red.

También, como establece Magrin et al. (2019) el despliegue de redes IoT puede resultar costoso y complejo. En este contexto, el hecho de que LoRaWAN requiera obligatoriamente *gateways* especializados y servidores de red dedicados incrementa los costos de implementación y limita su utilidad en entornos con pocos recursos. Esto representa un obstáculo significativo en áreas rurales o en proyectos de bajo presupuesto, donde la infraestructura necesaria puede ser inviable. Así, pese a ser un protocolo eficiente para soluciones de ámbito general, su rigidez arquitectónica y dependencia de componentes centrales restringen su utilidad en escenarios autónomos, flexibles o descentralizados.

2.6. Redes *Mesh*

Las redes *mesh* o redes malla son una topología cada vez más utilizada en distintas aplicaciones, siendo la más popular en la actualidad las redes IoT, donde la conectividad entre varios dispositivos es fundamental. Aunque este tipo de red es muy usada en la actualidad, esta no es una tecnología nueva. Sus inicios se remontan a los años 70, cuando comenzaron investigaciones acerca de topologías de red no centralizadas, con instituciones como el Laboratorio Naval de Investigación y DARPA de los Estados Unidos. Estas instituciones desarrollaron proyectos como *Packet-Radio* e *Intra-Task Force Network*, enfocados en aplicaciones militares que requieran redes descentralizadas.

Durante los años 80, las bases de las investigaciones previas condujeron al desarrollo de redes malla primitivas como lo es ad hoc. Este siendo un proyecto que sentó las bases de la tecnología de redes malla, especialmente su capacidad de gestión autónoma entre nodos y la transmisión de datos por múltiples saltos. Gracias a estos avances se tiene disponible las bases tecnológicas que hoy en día se utilizan en las redes inalámbricas e IoT (Ephremides, 2024).

2.6.1. Características

Las redes mallas poseen dos elementos clave en su topología, los enlaces y nodos. Un nodo es cualquier dispositivo que es capaz de recibir, enviar y retransmitir datos, mientras que un enlace es la conexión entre dos nodos, esto quiere decir que mientras más nodos existan más enlaces se establecerán. Sin embargo, incluso en configuraciones donde no todos los nodos están conectados entre sí existen múltiples rutas que permiten mantener conectividad.

Esta estructura distribuida favorece una alta escalabilidad, ya que permite incorporar nuevos nodos sin necesidad de reorganizar toda la red, y a su vez, proporciona una notable fiabilidad. Gracias a sus múltiples caminos de transmisión y a la capacidad de los nodos para decidir de manera autónoma por dónde redirigir el tráfico, las redes malla se distinguen por su tolerancia a fallos (Held et al., 2005).

Cabe destacar que estas redes se caracterizan por su descentralización, lo que significa que no dependen de un punto de acceso o infraestructura central para funcionar correctamente. Cada nodo puede actuar como un enrutador, gestionando el envío y recepción de datos de forma autónoma. Esta característica otorga una flexibilidad significativa, ya que permite modificar la estructura de la red, incorporar nuevos nodos o desconectarlos según las necesidades específicas. La popularidad de este tipo de redes se centra precisamente en su capacidad de adaptación frente a la rigidez de las infraestructuras tradicionales.

Otra característica relevante es su capacidad para mantener la conectividad en entornos poco convencionales, como zonas rurales o áreas urbanas con múltiples obstrucciones, donde las redes tradicionales no funcionarían adecuadamente o ni siquiera podrían ser implementadas. Además, los nodos no requieren una línea de vista directa con el destino final, ya que la comunicación se realiza mediante múltiples saltos entre nodos intermedios, mejorando así la cobertura y evitando la dependencia de un único camino (Sichitiu, 2005). Debido a estas propiedades de flexibilidad, descentralización, autonomía y resiliencia, las redes malla son ampliamente utilizadas en aplicaciones de IoT y en entornos donde el acceso a infraestructura convencional resulta limitado o inviable.

2.6.2. Topologías

Las redes *mesh* no están limitadas a una topología física por sus cualidades de flexibilidad, fácil adición de nuevos nodos y la capacidad de estos en estar conectados con otros por varios enlaces, pero se pueden clasificar a todas las variantes en tres topologías *mesh* principales como lo son *mesh* completa, *mesh* parcial e hibrida *star-mesh* (Espina et al., 2014).

Como se observa en la Figura 6, todas las topologías comparten una característica fundamental, la existencia de múltiples enlaces entre nodos, aunque se diferencian por la cantidad de enlaces disponibles y el número de saltos necesarios para alcanzar un nodo especifico.

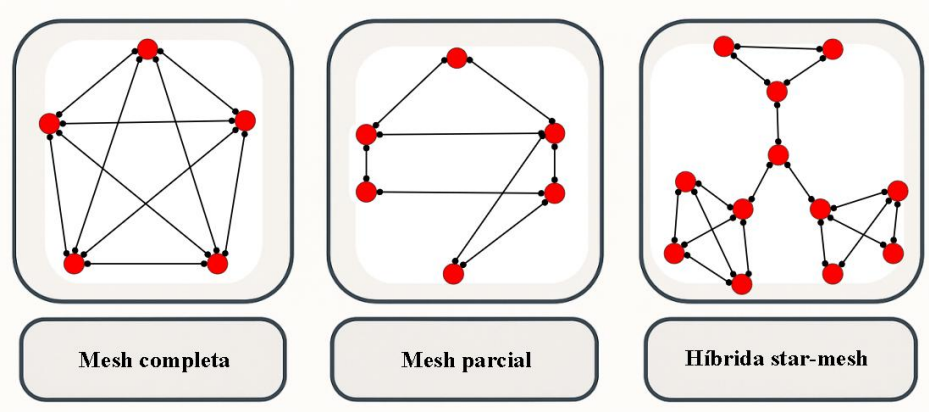

Figura 6 Tipos de topología mesh. Elaboración propia.

- ***Mesh* completa:** En esta topología todos los nodos están interconectados entre sí, formando una red de alta redundancia debido a todos los enlaces disponibles. Esta estructura garantiza comunicación desde cualquier nodo de la red mediante un solo salto, mejorando fiabilidad, tolerancia a fallas y evitando cuellos de botella. Puede resultar compleja su administración si la red crece demasiado y por ende sus enlaces crecen exponencialmente.
- ***Mesh* parcial:** Posee una topología basada en la *mesh* completa, pero sólo unos pocos nodos tienen múltiples conexiones, mientras que otros se enlazan a pocos vecinos. Es ideal para redes donde no se requiera conectividad total. Con esta estructura se mantiene cierta redundancia evitando la creación exponencial de enlaces cada que un nuevo nodo se adiciona a la red. Para este tipo de red es necesario protocolos de enrutamiento para asegurar la comunicación entre nodos.
- **Híbrida *star-mesh*:** Este tipo de topología combina la simplicidad de las redes tipo estrella con las redes *mesh* completa o parcial. Su principal ventaja es su escalabilidad ya que permite la integración de nuevos nodos sin que tengan que incorporarse a la malla, sólo necesitan incorporarse a un nodo centralizado de dicha malla. Sin embargo, debido a sus regiones con estructura de estrella existen nodos centrales que en el caso de fallo aislarían parte de la red.

2.6.3. Comunicación en redes *mesh Wireless*

Las redes *mesh Wireless* heredan características de la comunicación inalámbrica. En primer lugar, tenemos que estas igual sufren de alta probabilidad de colisiones, esto debido a su operación en un medio compartido, el espectro de radiofrecuencia. En este medio compartido todos pueden escuchar, pero sólo un dispositivo puede enviar en un determinado

tiempo, esto debido a que las transmisiones simultáneas generan colisiones (Schirn, 2023). Por otro lado, la comunicación en redes *mesh* requiere de protocolos de enrutamiento para poder comunicar nodos que estén a varios saltos de distancia. Estos protocolos pueden emplear distintos métodos para reconocer vecinos, escoger rutas óptimas y adaptarse a cambios en la topología, como desconexiones o congestión. Sin embargo, los protocolos de enrutamiento para redes malla inalámbricas han sido poco estudiados en comparación con el enrutamiento tradicional, debido a los problemas propios de la comunicación *mesh*, como colisiones, cuellos de botella o limitaciones en los recursos disponibles en la red (Waharte et al., 2006).

En cuanto a la problemática de la alta probabilidad de colisiones debido a que la comunicación se da en un medio compartido hay varios mecanismos para evitar colisiones y mejorar el uso del canal. Cabe recalcar que estos mecanismos no son estándares de las redes *mesh*, pero pueden ser adaptados para su uso y la decisión de su aplicación depende de la finalidad de la red. Entre los más relevantes destacan:

- ***Carrier Sense Multiple Access with Collision Avoidance*** **(CSMA/CA):** Es un protocolo diseñado para reducir colisiones. En este los nodos verifican que el canal esté libre y utilizan un *backoff* variable para minimizar la probabilidad de trasmitir mientras otro nodo inicia su transmisión o durante la transmisión de otro nodo (Schirn, 2023).
- ***Listen Before Talk*** **(LBT):** Implementa un esquema similar a CSMA/CA. Este verifica el canal por un cierto tiempo para determinar si está libre antes de transmitir, y si el canal no se encuentra libre espera un determinado tiempo antes de volver a verificar. Este esquema es muy usado en redes de identificación por radiofrecuencia (RFID) (López-Matencio et al., 2020).
- ***Time Division Multiple Access*** **(TDMA):** Este es un mecanismo que asigna intervalos de tiempo dedicados a cada nodo para trasmitir, lo cual elimina la probabilidad de colisiones, pero requiriendo sincronización precisa entre nodos. Requiere de una red que gestione planificación de transmisiones y todos sus nodos se sincronicen, esto añadiendo latencia a su comunicación. Este esquema es muy usado en redes industriales (Pérez, 1998).

Además de los mecanismos aplicables a las redes malla para mitigar las colisiones, este tipo de red requiere protocolos o mecanismos de enrutamiento que permitan la

comunicación entre nodos distantes, adaptándose a la estructura flexible de las topologías *mesh*. A diferencia de las redes tradicionales, los protocolos para *mesh* deben considerar número de saltos, calidad de enlace, nodos disponibles, incluso interferencia. Entre los protocolos de enrutamiento aplicables para redes *mesh* se encuentran:

- ***Ad hoc On-Demand Distance Vector Routing* (AODV):** Es un protocolo reactivo, que sólo busca rutas cuando lo necesita. Para esto emplea mensajes de requerimiento de ruta (RREQ) y mensajes de respuesta de ruta (RREP). Con este esquema de enrutamiento cada nodo tiene una tabla de enrutamiento temporal construida en función de la demanda. En redes muy grandes los mensajes de control pueden generar sobrecarga en el medio de transmisión y un incremento considerable en las latencias al solicitar una ruta (Waharte et al., 2006).
- ***Flood Routing*:** Este esquema de enrutamiento es caracterizado por su simpleza. En este cada paquete se reenvía a todos los nodos vecinos excepto el nodo que originalmente envió el mensaje. Este método garantiza que el mensaje llegue siempre y cuando haya un camino, evitando cálculos complejos o parámetros a considerar al enrutar (Rahman et al., 2005). Cabe mencionar que este método puede provocar una gran sobrecarga a la red, desperdiciando ancho de banda y energía.
- ***Optimized Link State Routing* (OLSR):** Es un protocolo proactivo, donde todos los nodos mantienen rutas actualizadas hacia todos los posibles destinos. Esto se logra tras el intercambio de información constante sobre el estado de los enlaces entre vecinos. Básicamente, cada nodo tiene un mapa completo de la red y con esta información y varios algoritmos se elige la ruta de menor costo. Este protocolo consume mucho ancho de banda y no es ideal para dispositivos con *hardware* limitado (Clausen et al., 2003).
- ***Neighbor-Based Routing*:** Es un esquema de enrutamiento que aún se encuentra en estudio y su funcionamiento se basa en la recolección de información local de los nodos vecinos para las decisiones de enrutamiento. Este enfoque busca combatir con la sobrecarga de mensajes de control de AODV. Entre las técnicas recientes destaca el uso de probabilidades basadas en la densidad de vecinos, donde los nodos en zonas densas reducen el reenvío y los de zonas dispersas lo aumentan para mantener la conectividad (Ejmaa et al., 2016).

CAPÍTULO III: Diseño de Comunicación y Arquitectura

Este capítulo presenta el diseño del sistema de comunicación multi-nodo, el cual se basa en una arquitectura de red *mesh* utilizando la tecnología LoRa y microcontroladores ESP32. A partir de las limitaciones anteriormente mencionadas de protocolos existentes como LoRaWAN y la necesidad de soluciones descentralizadas para entornos IoT, se plantea un modelo de red que permita a los nodos comunicarse de forma directa entre sí y reenviar mensajes a través de múltiples saltos, sin requerir una infraestructura centralizada.

El diseño considera las restricciones inherentes a los enlaces LoRa, tales como el canal compartido y la ausencia de mecanismos de acceso al medio, y propone una solución ligera que opera hasta la capa 3 del modelo OSI. Para ello, se ha definido un conjunto de tipos de paquetes y mecanismos internos que permiten el descubrimiento de vecinos, el enrutamiento dinámico, la confirmación de entregas y la recuperación ante fallos, todo ello ejecutado mediante *firmware* personalizado.

A lo largo de este capítulo se detalla el modelo de comunicación propuesto, los paquetes definidos para la estructuración del tráfico, los mecanismos que garantizan la operación autónoma de la red, así como la arquitectura general del sistema. Esta incluye la topología adoptada, los componentes de hardware utilizados y los requisitos de software necesarios para su implementación. Este diseño tiene como objetivo ofrecer una capa de comunicación eficiente, adaptable y fácilmente replicable en entornos con hardware limitados.

3.1. Diseño de modelo de comunicación para red *mesh*

El sistema de comunicación diseñado para este trabajo de titulación se fundamenta en una arquitectura de red *mesh* basada en la tecnología LoRa, en la cual cada nodo perteneciente a la red tiene la capacidad de comunicarse con sus vecinos directos y reenviar mensajes hacia otros nodos distantes dentro de la misma red. Esta estructura se caracteriza por mantener una comunicación descentralizada, tolerante a fallos y escalable.

El modelo planteado posee diversas funcionalidades necesarias para el funcionamiento de la red, tales como descubrimiento de nodos vecinos, establecimiento de rutas dinámicas hacia nodos distantes, uso de *Time to Live* (TTL) para controlar la propagación de mensajes en la red, y acuses de recibo (ACK) entre saltos para confirmar entrega de mensajes. Estas capacidades permiten mantener la comunicación entre nodos vecinos y facilitan el enrutamiento de mensajes a nodos a varios saltos de distancia.

Cabe mencionar que el protocolo LoRaWAN no fue considerado como base para este diseño debido a sus limitaciones respecto a su arquitectura rígida, basada en una topología de estrella de estrellas, su dependencia de *gateways* y servidores de red, así como su limitada flexibilidad, lo cual incrementa los costos de implementación e introduce un punto único de falla. El modelo propuesto constituye una alternativa a LoRaWAN, orientada a aplicaciones que requieran descentralización, adaptabilidad y sencillez en la comunicación entre nodos distribuidos.

Basado en estas funcionalidades, el modelo de comunicación puede ser relacionado con el modelo de referencia OSI, a pesar de no emplear protocolos estándar. Como se muestra en la Figura 7, el sistema opera desde la capa 1, correspondiente a la capa física, al utilizar la modulación LoRa, así como los parámetros de frecuencia y potencia de transmisión propios de esta capa. También actúa en la capa 2, que corresponde al enlace de datos, al encargarse de la comunicación punto a punto entre nodos mediante LoRa, gestionando aspectos como la detección de vecinos accesibles. Finalmente, se implementan funciones propias de la capa 3, correspondiente a la capa de red, como el enrutamiento dinámico, el direccionamiento lógico entre nodos y el uso de TTL para limitar el alcance de los mensajes.

Por otro lado, el sistema no opera en las capas superiores del modelo OSI. No gestiona el orden ni la recuperación de los mensajes en la capa 4, correspondiente al

transporte, ni establece sesiones activas entre aplicaciones, como lo hace la capa 5 de sesión. Tampoco realiza transformaciones de datos o formatos de presentación propias de la capa 6, ni utiliza protocolos específicos de aplicación, propios de la capa 7. Esto se debe a que el objetivo del sistema es proporcionar una infraestructura de comunicación base, independiente de una aplicación específica.

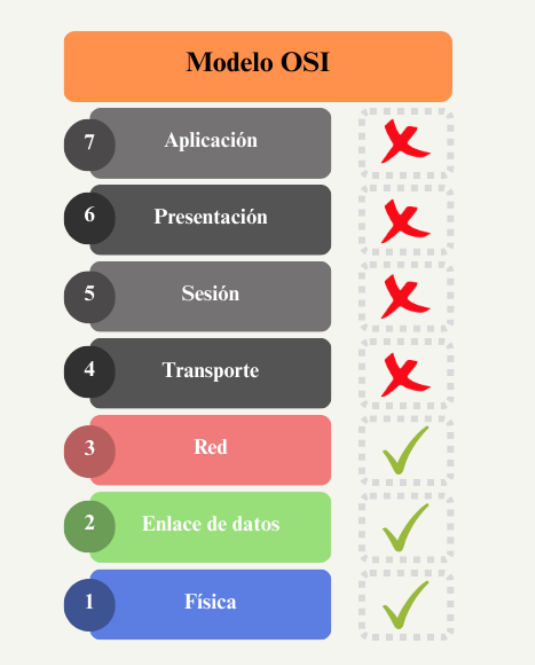

Figura 7 Capas del modelo OSI cubiertas por el sistema de comunicación LoRa mesh. Elaboración propia.

3.1.1. Tipos de paquetes definidos

Todos los paquetes manejados por el sistema de comunicación cumplen un rol importante en el funcionamiento y control de la red. Por ello, sus formatos incluyen los campos necesarios para soportar, además de la transmisión de datos, funciones como el descubrimiento e incorporación de nodos, el mantenimiento de rutas y el control de fiabilidad. Todos los paquetes, además de los campos específicos según su propósito, comparten un encabezado común como se identifica en la Figura 8, el cual está compuesto por el identificador de tipo, el identificador de malla y el identificador de mensaje. Esta cabecera permite discriminar rápidamente cómo procesar cada paquete, aceptar únicamente los pertenecientes a la misma red e identificar de forma única cada transmisión.

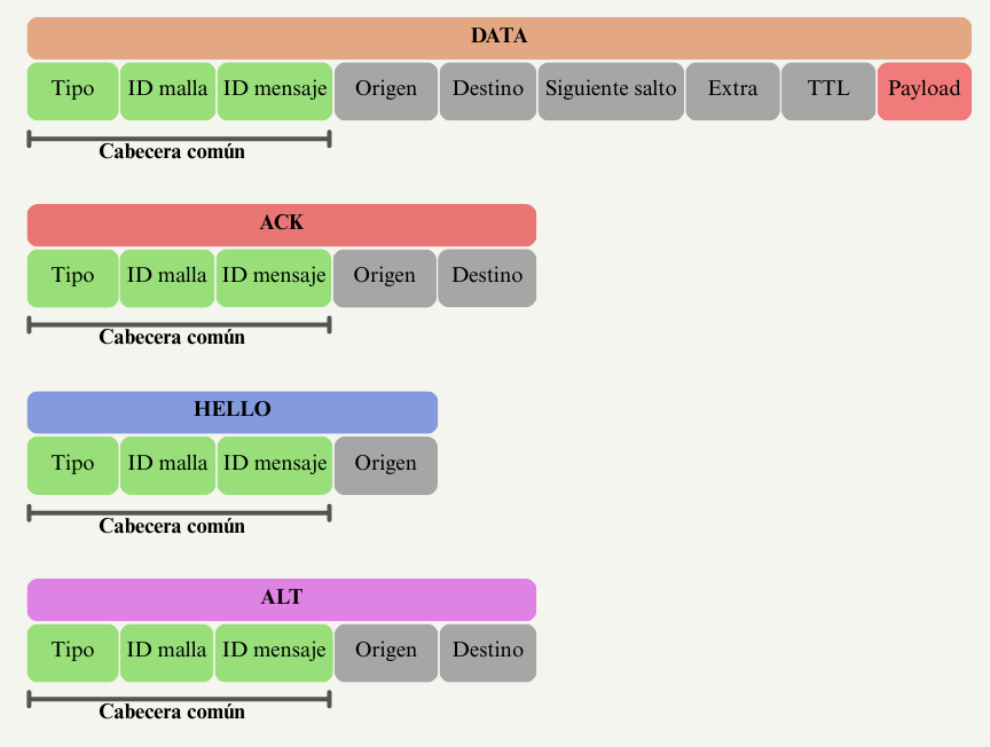

Figura 8 Estructura y campos de los paquetes definidos: DATA, ACK, HELLO y ALT. Elaboración propia.

En este sistema se han definido cuatro tipos principales de paquetes: DATA, ACK, HELLO y ALT. Cada uno cumple con una función particular dentro del proceso de comunicación entre nodos, y su estructura responde a los requerimientos específicos de dicha función. A continuación, se describen en detalle los tipos de paquete y el propósito de cada uno.

- **DATA:** Este tipo de paquete tiene como objetivo principal el transporte de información. Posee la cabecera común, además de campos adicionales como los identificadores del nodo de origen, de destino y del siguiente salto, necesarios para el reenvío de datos entre nodos. También incluye un campo TTL, utilizado para controlar el tiempo de vida del mensaje en la red, y un campo de carga útil o *payload* que contiene la información a transmitir. Finalmente, se incorpora un campo adicional de uso experimental, pensado para futuras extensiones del sistema que requieran incluir información de capas superiores sin modificar la estructura base del paquete. Este enfoque se inspira en el campo *experimental bits* (EXP) de *Multiprotocol Label Switching* (MPLS), originalmente reservado para fines experimentales (Cisco, 2025).

- **ACK:** El paquete de tipo ACK cumple la función de acuse de recibo o confirmación de entrega. Al igual que el paquete DATA, posee la cabecera común. Contiene los identificadores del nodo que emite la confirmación y del nodo que envió el mensaje original. A diferencia del paquete DATA, no incluye un campo de siguiente salto, ya que se trata de una confirmación *hop-by-hop*, es decir, entre nodos vecinos únicamente.

- **HELLO:** Este paquete de control tiene como propósito anunciar la presencia del nodo emisor a sus vecinos. Incluye la cabecera común y un campo adicional que indica el identificador del nodo que emite el mensaje. Su función es permitir el descubrimiento de vecinos accesibles dentro del alcance de transmisión.

- **ALT:** El paquete ALT tiene como función notificar al nodo origen que el camino seleccionado no es válido o se encuentra congestionado, solicitando que se intente una ruta alternativa. Incluye la cabecera común, junto con los identificadores del nodo que detectó el problema y del nodo al que debe llegar la advertencia. De esta forma, permite redirigir el flujo de mensajes y evitar bucles o fallos en la entrega.

3.1.2. Mecanismos de comunicación

Debido a que el sistema de comunicación propuesto opera hasta la capa 3 del modelo OSI (capa de red), debe asumir muchas de las responsabilidades típicamente delegadas a protocolos estándar, como el direccionamiento lógico entre nodos, la selección dinámica de rutas, la gestión del acceso al medio compartido y la entrega confiable de los mensajes. Todo esto debe realizarse de manera eficiente sobre enlaces de baja velocidad, como los ofrecidos por la tecnología LoRa, y bajo condiciones de consumo energético restringido. A diferencia de arquitecturas más complejas como LoRaWAN, este diseño apuesta por un enfoque totalmente distribuido, que elimina la necesidad de servidores centrales o *gateways*, favoreciendo una red malla autónoma, tolerante a fallos y altamente escalable.

Para lograr esto, el sistema implementa un conjunto de mecanismos que cubren desde la asignación automática de identificadores únicos de nodo hasta la gestión del reenvío de mensajes en múltiples saltos, la detección de vecinos disponibles y la recuperación ante fallos de transmisión. Cada uno de estos mecanismos se apoya en distintos tipos de paquetes definidos como HELLO, DATA, ACK y ALT que permiten coordinar las funciones críticas del sistema y permiten coordinar el comportamiento de la red. A continuación, se detallan

los principales mecanismos que permiten sostener la operación descentralizada y confiable de la red *mesh* diseñada.

- **Asignación de identificador de nodo:** Cada microcontrolador deriva su identificador de nodo a partir del número de serie grabado en el *chip* ESP32. Al combinar mediante una operación sencilla porciones alta e inferior, de dicho número se obtiene un valor de 16 *bits* irrepetible dentro de la red. Este procedimiento evita la dependencia de un servidor de registro, garantiza unicidad sin intervención humana y permite que un nodo recién encendido se incorpore de forma *plug-and-play*, reduciendo al mínimo el tiempo de despliegue.
- **Planificador de cola:** El sistema de comunicación gestiona en cada nodo un planificador que agrupa el tráfico saliente y encola mensajes críticos como ALT (rutas alternativas), ACK (confirmaciones), HELLO (anuncios de topología) y DATA (información o datos). Este planificador permite enviar distintos tipos de paquetes según los eventos en la red y evita que un paquete pendiente de envío sea sobrescrito por uno nuevo. El mecanismo se inspira en el funcionamiento del estándar IEEE 802.11e, el cual introduce múltiples colas *First In First Out* (FIFO) denominadas categorías de acceso. En particular, la categoría *Best Effort* opera bajo una política de planificación FIFO básica (S. C. Perez et al., 2013).
- **Transmisión de paquetes:** Cada paquete encolado recibe un tiempo de espera aleatorio antes de transmitirse. El objetivo es romper la sincronización espontánea entre nodos y dispersar los instantes de emisión, reduciendo la probabilidad de colisiones sin recurrir a relojes globales ni coordinación centralizada. Este mecanismo se basa en estudios como el de Shenoy et al. (2015), donde se demuestra que la introducción de retardos cortos aleatorios mejora el rendimiento en redes con múltiples nodos, al reducir significativamente la probabilidad de colisiones.
- **Descarte de paquetes:** Tras la recepción, el nodo descarta de inmediato cualquier paquete que declare un tipo no soportado, supere el tamaño máximo permitido o pertenezca a una malla distinta. Esta filtración temprana protege los recursos locales y evita que tráfico mal formado perturbe la red.
- **Ventana de escucha:** Antes de trasmitir cada nodo abre una ventana de escucha, aplicando los principios de LBT. Si se detecta actividad, se aplaza la transmisión y mediante un *backoff* se espera antes de comenzar a trasmitir. Tras un número finito

de ventanas, el paquete se envía sin importar que el canal esté ocupado para evitar bloqueos prolongados y garantizar que todo nodo tenga una oportunidad de trasmitir.

- **Recepción de paquetes:** El estado por defecto de cada nodo es de escucha continua, es decir mientras el nodo esté desocupado la escucha es continua. En el caso de recibir un paquete se aplica un *backoff* para evitar trasmisiones durante el período de recepción, siguiendo un enfoque basado en lo propuesto por Shenoy et al. (2015). Cabe recalcar que al recibir un paquete este se deserializa en caso de ser el destino y procesado, reenviado o descartado según corresponda.
- **Construcción y depuración de la tabla de vecinos:** Para tener la información de los vecinos válidos a los cuales se pueda enviar los paquetes se emplea una tabla de vecinos que se llena dinámicamente. Cada cierto período de tiempo se encola un paquete HELLO, que se envía en modo de *broadcast* a todo nodo que esté al alcance del emisor, los nodos que lo reciben lo deserializan y procesan, añadiendo al nodo emisor a su tabla de vecinos, incluyendo datos como su fuerza de señal y el *time-stamp* de recepción para tener una métrica de antigüedad del vecino. Mediante un temporizador interno se borra automáticamente los vecinos más antiguos para asegurar que sólo se trabaje con nodos disponibles.
- **Enrutamiento basado en vecinos:** Mecanismo que determina el siguiente salto basado en los vecinos disponibles para el nodo. En el caso que el destino se encuentre en los vecinos se trasmite directamente. Cuando un paquete no puede entregarse directamente, el nodo consulta la tabla de vecinos y calcula una métrica ponderada entre la intensidad de señal (como estimación de cercanía) y el *time-stamp* (como indicador de frescura del enlace). Se selecciona un grupo de nodos candidatos y se elige aleatoriamente entre ellos, de esta forma no se sobre exige a un solo nodo, se prioriza nodos contactados recientemente y que tengan una señal razonable. Este enfoque se alinea con el principio propuesto en el estudio de Karp & Kung (2000), donde las decisiones de reenvío se toman usando únicamente la información local de los vecinos inmediatos. Este tipo de enrutamiento ligero es ideal para redes descentralizadas compuestas por dispositivos con recursos limitados, como los ESP32.
- **Reenvío por saltos:** En primera instancia, si un nodo genera un mensaje, este emplea el mecanismo de enrutamiento basado en vecinos para determinar su primer salto dentro de una cadena de reenvíos. Tras verificar el destino del paquete y con ayuda

del enrutamiento basado en vecinos, cada nodo determina si el mensaje recibido debe encolarse para reenviarse. En este proceso se decrementa el TTL y cada nodo involucrado en la cadena de saltos sólo reencola el paquete si el valor del TTL es positivo. De esta forma se controla y limita la distancia que puede recorrer un paquete, evitando bucles o circulaciones indefinidas.

- **Alerta de ruta alternativa:** Si un nodo detecta un paquete ya procesado anteriormente, es decir duplicado o al verificar los vecinos válidos en la tabla de vecinos, identifica que no existen vecinos válidos diferentes al emisor original, se envía un paquete ALT al origen. El emisor interpreta el paquete como indicio de congestión, nodo inaccesible o nodo distante y reprograma el paquete buscando otra ruta alternativa. Este proceso toma inspiración del enrutamiento AODV, específicamente de los mensajes de error de ruta (RERR) (Perkins et al., 2003).
- **Historial de mensajes duplicados:** Para evitar bucles de reenvío, cada nodo maneja una lista de identificadores de paquetes ya procesados. Si llega un paquete con un identificador presente en el historial, se descarta y notifica al emisor con un paquete de respuesta ALT.
- **Confirmación *hop-by-hop*:** Cada paquete de datos debe ser confirmado por el nodo más próximo que lo recibe, independientemente que sea un nodo intermedio o destino. Sólo se envían ACKs entre saltos, no entre origen y destino, esto para desocupar al nodo lo antes posible y evitar largas esperas por un ACK o saturar toda la cadena de saltos (Scofield et al., 2008). Si la confirmación no llega dentro de un plazo establecido el paquete se vuelve a encolar un número finito de veces para evitar bloqueos y permitir la transmisión de otros paquetes.
- **Reconvergencia tras fallo de ACK:** Si un nodo no responde con un ACK tras varios intentos, el nodo emisor asume que el destino es inaccesible, se lo elimina de la tabla de vecinos y recalcula otra ruta alternativa para enviar el paquete. Este mecanismo permite una auto reparación local de la red sin necesidad de un redescubrimiento global o de control centralizado, lo que lo hace eficiente en redes dinámicas o de bajos recursos.

3.2. Diseño de la arquitectura del sistema

La arquitectura del sistema de comunicación multi-nodo se fundamenta en tres elementos esenciales, el hardware empleado en cada nodo, el software encargado de

gestionar la comunicación entre nodos y la topología que define el relacionamiento entre los dispositivos. Estos tres elementos actúan de forma complementaria para que la red *mesh* sea funcional, descentralizada, flexible y adaptable a distintos escenarios.

Cada nodo de hardware está compuesto por una placa de desarrollo comercial del proveedor Heltec, que integra el microcontrolador ESP32 y el módulo de radiofrecuencia LoRa. El software que soporta el sistema está desarrollado en C++ y se compila mediante Arduino IDE, donde se implementa la lógica de los mecanismos de comunicación previamente detallados, apoyándose en bibliotecas propias de Arduino y del proveedor. Finalmente, el tercer componente es la topología, donde se adopta una *mesh* parcial, ya que representa el escenario ideal para demostrar la funcionalidad de los distintos mecanismos y garantizar la compatibilidad con diversas variantes de topologías malla.

3.2.1. Topología de red *mesh* básica

La topología del sistema de comunicación es una topología *mesh* parcial compuesta por un total de cinco nodos, seleccionada ya que, como prototipo, esta topología tiene la capacidad de representar múltiples escenarios y comportamientos en redes *mesh*. A diferencia de una topología *mesh* completa, en la cual todos los nodos se conectan directamente entre sí con un solo salto, una *mesh* parcial permite evaluar comportamientos más realistas, como rutas indirectas, nodos periféricos y redundancia de caminos.

Como se puede observar en el diagrama de topología representado en la Figura 9, la topología definida incluye cinco nodos etiquetados como A, B, C, D y E. El nodo A representa un nodo periférico, conectado únicamente al nodo B, lo que permite analizar el comportamiento de un nodo con un solo vecino directo. Por su parte, el nodo B ocupa una posición central, siendo el único con tres vecinos directos (A, C y D), y actúa como nodo de interconexión clave en la red. Finalmente, desde B se extienden dos rutas distintas hacia el nodo E, una a través del nodo C y otra mediante el nodo D.

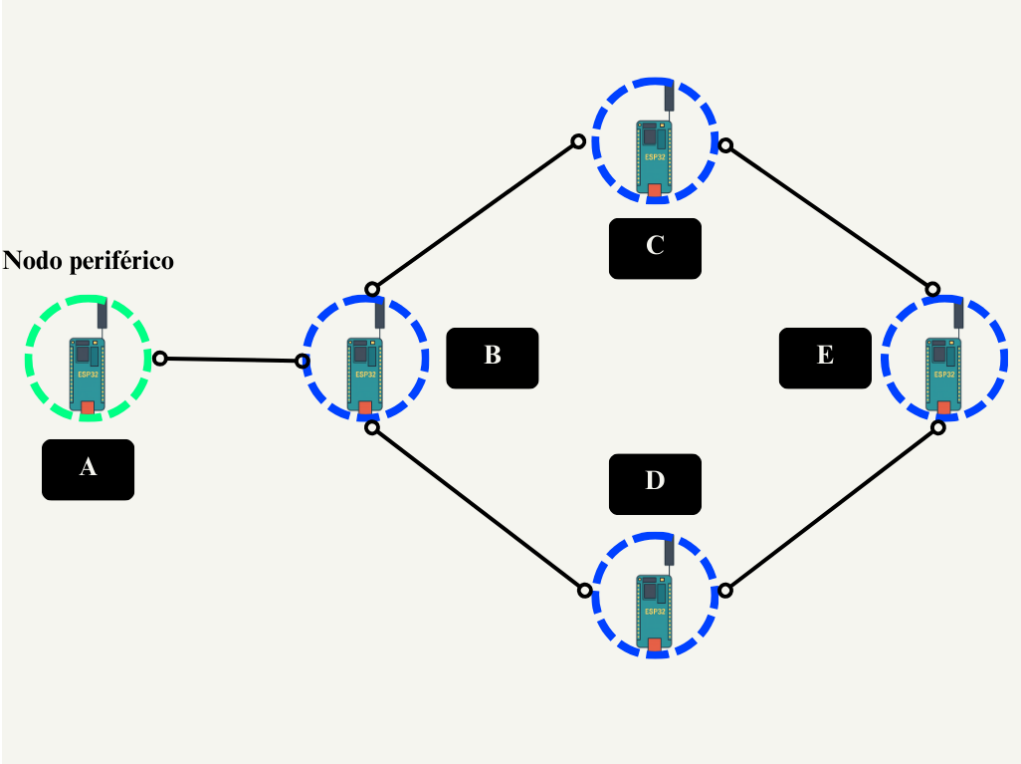

Figura 9 Topología mesh parcial utilizada en la red de microcontroladores ESP32 con comunicación LoRa. Elaboración propia.

Este diseño de topología permite simular la existencia de rutas alternativas hacia un mismo destino, condición esencial para evaluar mecanismos de reenvío, enrutamiento dinámico y selección de rutas alternativas ante fallos. Cabe recalcar que el hecho que el nodo E no esté conectado directamente al nodo A, permite que el sistema requiera de múltiples saltos y dependencia del enrutamiento para establecer comunicación.

De este modo, la topología propuesta permite recrear escenarios fundamentales para el análisis del sistema. Entre ellos se incluye la comunicación entre nodos vecinos en un solo salto, como ocurre entre A y B; la comunicación *multihop* a través de varios saltos intermedios, como en la ruta A, B, D, E; la existencia de rutas alternativas hacia un mismo destino, como en los caminos B, C, E y B, D, E; y la situación de aislamiento relativo, representada por el nodo periféricos A, limitado a un solo enlace directo.

3.2.2. Especificación de hardware

El *hardware* empleado en cada nodo está basado en placas de desarrollo comerciales del proveedor Heltec (2023), específicamente el modelo *Wireless Stick* V3. Estas placas integran el microcontrolador ESP32-S3FN8, un procesador de doble núcleo con arquitectura Xtensa LX7 de 32 *bits*, que alcanza una frecuencia máxima de funcionamiento de 240 MHz.

El microcontrolador cuenta con 384 KB de memoria ROM, 512 KB de SRAM y 8 MB de memoria *flash*, adecuada para el almacenamiento del *firmware* y otros datos esenciales del sistema.

En cuanto a su capacidad de comunicación, la placa incorpora el módulo SX1262, responsable de las transmisiones mediante tecnología LoRa en la banda de 863 MHz a 928 MHz. Este módulo permite alcanzar una potencia máxima de transmisión de 21 ±1 dBm y una sensibilidad de recepción de hasta -134 dBm, lo que garantiza una comunicación eficiente en entornos con infraestructura limitada. Cabe destacar que las placas basadas en ESP32 se caracterizan por incluir soporte nativo a otras tecnologías de comunicación, y la placa utilizada no es la excepción, ya que incorpora conectividad *Wi-Fi* (802.11 b/g/n) y *Bluetooth* 5.0, lo cual refuerza su aplicabilidad en proyectos IoT.

Al tratarse de una placa de desarrollo multipropósito, ofrece un *pinout* completo. Como se puede ver en la Figura 10, este está distribuido en dos hileras de pines hembra, con un total de 34 pines, entre puertos GPIO y líneas de entrada/salida de voltaje. Dispone de dos pulsadores físicos, uno de ellos destinado al reinicio del sistema o del *firmware*, y otro orientado a la carga del *firmware* o de uso general configurable por el usuario.

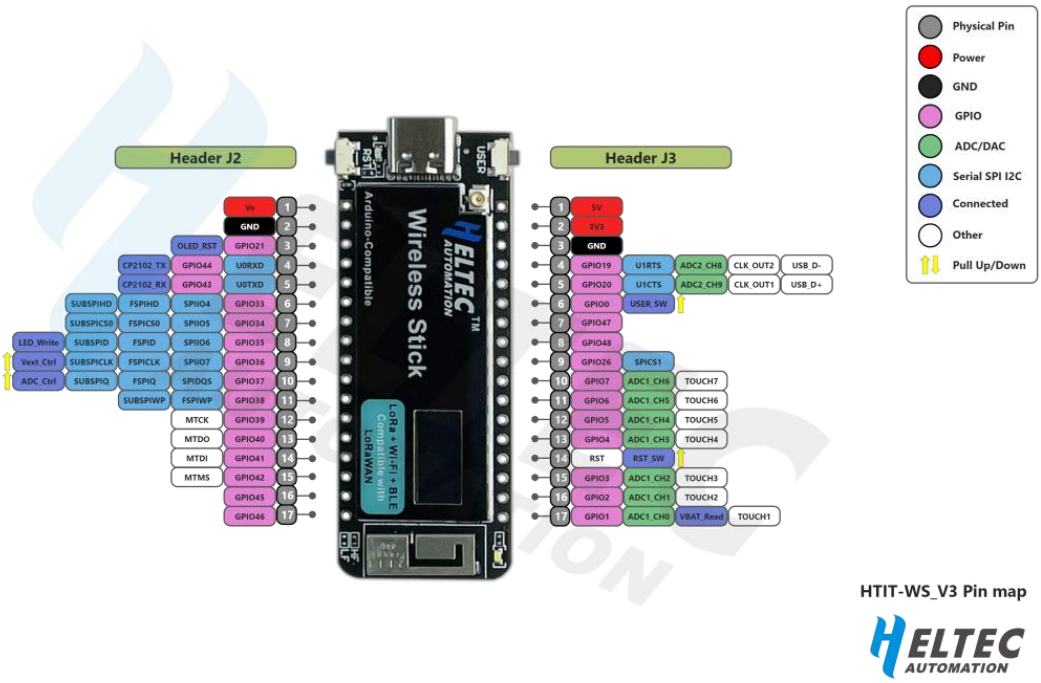

Figura 10 Mapa de Pines GPIO y Funciones del Heltec Wireless Stick V3 (ESP32-S3). Tomado de Zona Industrial (2025).

Cabe mencionar, que al estar orientada a aplicaciones IoT, esta placa incluye elementos adicionales que facilitan su operación y desarrollo embebido. Cuenta con un conector *Universal Serial Bus* (USB) tipo C, que permite tanto la alimentación eléctrica como la carga de firmware y la depuración del sistema. También dispone de un módulo de gestión energética compatible con baterías externas de 3.7 V, con protección contra sobrecargas y conmutación automática entre fuente USB o batería. Asimismo, incorpora una pantalla OLED monocromática de 0.49 pulgadas (64×32 píxeles), útil para mostrar información del sistema, estados operativos o mensajes de depuración durante el desarrollo y pruebas.

Finalmente, el prototipo del sistema de comunicación depende completamente del hardware descrito, ya que cada nodo está basado en esta placa por su alta integración y funcionalidad. La elección del modelo *Wireless Stick* V3 se justifica por incluir, en un solo módulo compacto, todos los elementos esenciales para el funcionamiento del sistema, tales como el microcontrolador ESP32-S3, el *chip* LoRa SX1262, conectividad USB tipo C para programación y alimentación, y una pantalla OLED integrada que facilita la verificación del estado del sistema durante el desarrollo y operación.

3.2.3. Especificación de *software*

El sistema de comunicación multi-nodo se apoya en una arquitectura de software específicamente configurada para programar y gestionar el comportamiento de cada uno de los nodos basados en ESP32. Para el desarrollo y despliegue del firmware, se utiliza el entorno de desarrollo Arduino IDE, en su versión 2.3.2 para Windows de 64 *bits*. Esta plataforma permite escribir, compilar y cargar el código directamente en las placas Heltec.

Para el correcto funcionamiento del software montado en las placas, el sistema depende del repositorio de definición de placas de Heltec para dispositivos ESP32, específicamente en su versión 3.0.2. Con esta configuración se habilita el soporte completo para modelos como el *Wireless Stick* V3, permitiendo configurar parámetros específicos del módulo LoRa SX1262, así como acceder a las funciones integradas del microcontrolador.

Por otro lado, para poder programar las placas y cargar el firmware personalizado, se requiere la instalación de los controladores CP210x USB to UART *Bridge*, en su versión para arquitecturas de 64 *bits*. Estos controladores garantizan una conexión estable mediante

el puerto USB tipo C incorporado en las placas, tanto para la transferencia del *firmware* como para la depuración en tiempo real.

El *firmware* de este sistema depende de diversas bibliotecas, tanto estándar como específicas del entorno ESP32 y del proveedor Heltec, que permiten implementar una lógica completamente personalizada. La biblioteca *Arduino.h* proporciona funciones esenciales de entrada y salida digital, temporización y control de periféricos. Para la manipulación eficiente de memoria y la serialización de estructuras de datos se emplea *string.h*, mientras que *esp_system.h* ofrece acceso a funcionalidades internas del microcontrolador y control directo del *hardware*.

Por otro lado, los métodos necesarios para interactuar con la pantalla OLED integrada se gestionan mediante la biblioteca *Wire.h*, y su visualización se realiza a través de *HT_SSD1306Wire.h*, una librería específica de Heltec diseñada para controlar pantallas monocromáticas de 0.49 pulgadas. Finalmente, la biblioteca *LoRaWAN_APP.h* permite configurar y operar el módulo de radiofrecuencia SX1262, incluyendo la inicialización del *chip*, la gestión de eventos de transmisión y recepción, y la definición de parámetros físicos del canal, como la frecuencia, el ancho de banda, el SF y la potencia de transmisión.

Todas estas bibliotecas son fundamentales para el funcionamiento del firmware, ya que permiten interactuar directamente con el hardware y ejecutar los mecanismos del sistema, tales como asignación de identificador de nodo, planificador de cola, transmisión de paquetes, descarte de paquetes, ventana de escucha, recepción de paquetes, construcción y depuración de la tabla de vecinos, enrutamiento basado en vecinos, reenvío por saltos, alerta de ruta alternativa, historial de mensajes duplicados, confirmación *hop-by-hop* y reconvergencia tras fallo de ACK.

CAPÍTULO IV: Desarrollo del Prototipo

Este capítulo contiene una descripción técnica del desarrollo e implementación del prototipo de sistema de comunicación multi-nodo, cubriendo la configuración inicial del entorno de desarrollo y la implementación de los distintos mecanismos asociados con transmisión, recepción, enrutamiento entre otros. El sistema fue programado en su totalidad en Arduino IDE, escogido por su compatibilidad con librerías específicas para hardware especifico, así como por su facilidad de uso y carga del *firmware*.

El eje central del sistema está ubicado en el archivo principal *LoRaMesh.ino* que contiene el bucle estándar presente en sistemas embebidos. Este archivo gestiona la operación de varios módulos que encapsulan mecanismos específicos del sistema definidos en la etapa de diseño, así como módulos auxiliares como el gestor de la pantalla integrada *oled_manager.h.* Todo esto con ayuda del archivo de configuración global *config.h*.

Dentro de dichos módulos se encuentra *packet_manager.h* que implementa el mecanismo de manejo de paquetes, la definición de las estructuras de los tipos de paquetes, así como la gestión de deserialización y serialización de estos. En este módulo también se gestiona el llenado de los paquetes y el etiquetado de los identificadores de nodo con el mecanismo de asignación de identificador de nodo mediante el número de serie del ESP32.

Por otra parte, *communication_manager.h* gestiona la transmisión, recepción básica y procesamiento general de paquetes, incluyendo el descarte de estos. Este módulo trabaja con *message_scheduler.h*, el cual implementa el planificador de cola, encargado de la programación de envíos, la confirmación *hop-by-hop* y la reconvergencia en caso de fallos de ACK. Además, *message_receiver.h* complementa la recepción con mecanismos como la ventana de escucha y el historial de duplicados, evitando colisiones y retransmisiones innecesarias. Por su parte, *routing_manager.h* se encarga del mantenimiento de la tabla de vecinos y del cálculo dinámico del siguiente salto para asegurar la comunicación entre varios nodos.

De esta manera a lo largo de este capítulo se aborda desde la implementación técnica del prototipo hasta la interacción y dependencia de los módulos con sus respectivos mecanismos para asegurar una comunicación descentralizada en una red *mesh*. Para mayor detalle técnico, puede consultarse el repositorio del proyecto disponible en GitHub (Arregui, 2025).

4.1. Configuración del entorno de desarrollo

Debido a que el sistema de comunicación está basado en su totalidad en el entorno de Arduino IDE, la configuración del entorno de desarrollo de este es sencilla, lo que facilita el despliegue de proyectos embebidos. El acceso a este entorno se consigue mediante la descarga de Arduino IDE de la página oficial como se muestra en la Figura 11. Una vez instalado el entorno está listo para ser personalizado y utilizado.

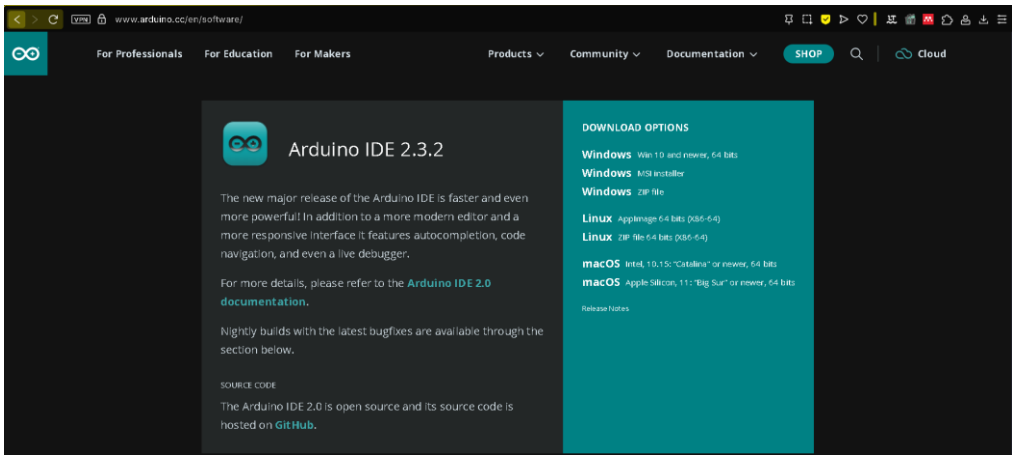

Figura 11 Obtención del entorno Arduino IDE. Elaboración propia.

Cabe recalcar, que para que el entorno esté en condiciones de configurar correctamente la placa *Wireless Stick* de Heltec, es necesaria la instalación previa del controlador CP210x USB to UART *Bridge.* Así como lo menciona Silicon Labs (2021), este controlador es necesario para un correcto funcionamiento adecuado del puerto USB tipo C integrado en la placa, así como para la comunicación serial con el microcontrolador ESP32. Así como se evidencia en la Figura 12, el controlador está disponible para descargar directamente de la página oficial de Silicon Labs.

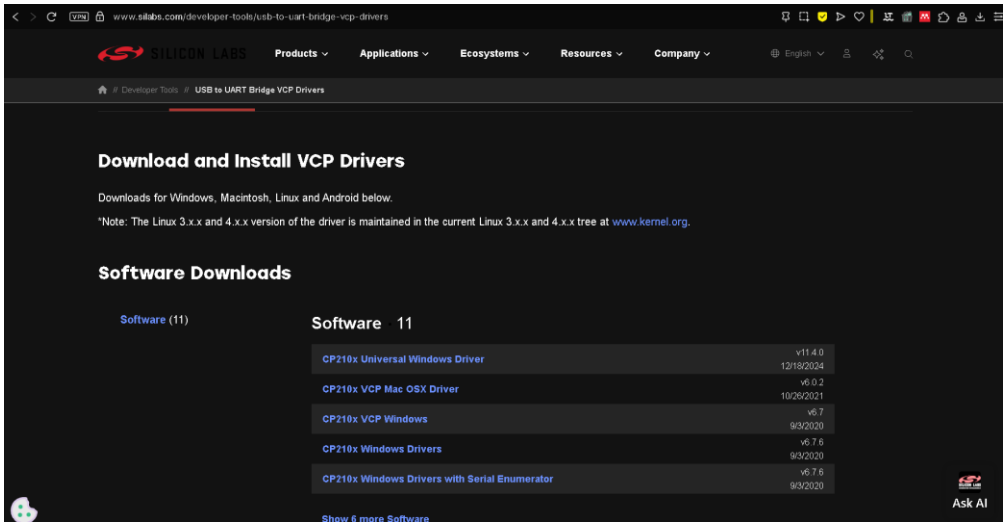

Figura 12 Obtención del controlador CP210x. Elaboración propia.

El entorno de Arduino IDE ya incluye por defecto librerías para la programación de placas de desarrollo como *Arduino.h* y otras librerías estándar del lenguaje C, utilizadas para tareas de manipulación de datos y estructuras. Sin embargo, para poder interactuar correctamente con placas de proveedores externos como Heltec, es necesario instalar tanto el repositorio de definición de placas ESP32 como las bibliotecas específicas para el modelo *Wireless Stick* V3, las cuales permiten gestionar el *chip* SX1262 de LoRa y acceder a las funciones integradas del microcontrolador. Como se visualiza en la Figura 13, el repositorio de placas de desarrollo está disponible en el gestor de placas del IDE y el paquete de librerías para el *Wireless Stick* de Heltec se encuentra en la sección de gestor de librerías.

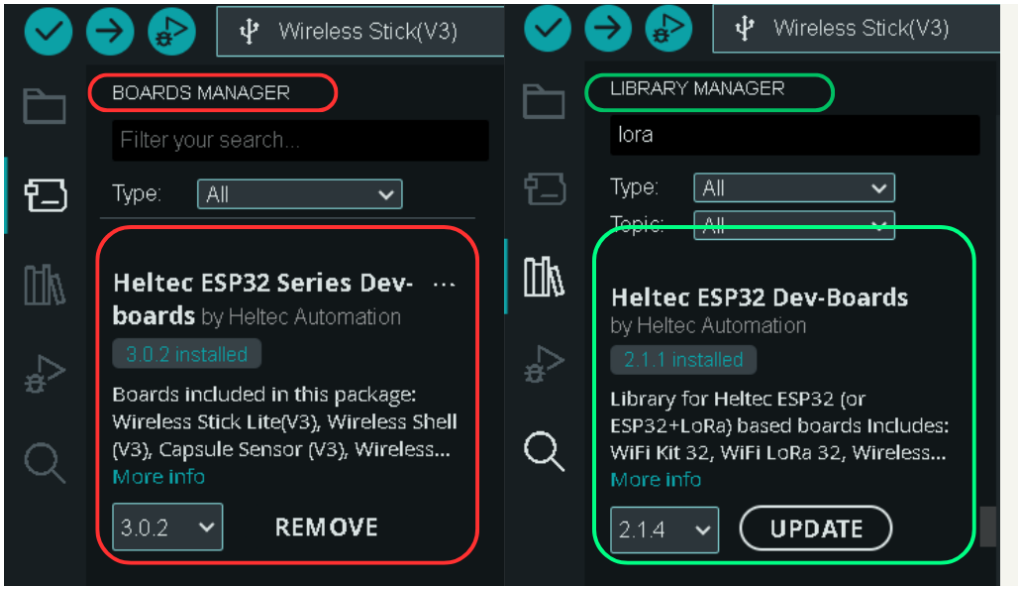

Figura 13 Gestor de placas y Gestor de librerías de Arduino IDE. Elaboración propia.

El entorno de Arduino IDE permite una integración directa con el ecosistema de Heltec, facilitando la programación de las placas *Wireless Stick* V3 gracias a su compatibilidad con repositorios y bibliotecas específicas. Esto permite que cada nodo del sistema de comunicación cuente con una configuración adecuada para habilitar tanto la transmisión mediante LoRa como el control del microcontrolador ESP32. La combinación de estos elementos, junto con las bibliotecas propias del entorno y el controlador CP210x, asegura que el entorno de desarrollo esté completamente preparado para soportar la configuración del sistema de comunicación.

4.2. Implementación del sistema de comunicación

El *firmware* del sistema de comunicación está compuesto por varios módulos que encapsulan los mecanismos de comunicación en distintos archivos fuente, esto facilita su organización, comprensión y una posible ampliación en un futuro. Debido a que se está trabajando con Arduino IDE se necesita del archivo principal *LoRaMesh.ino* ya que el entorno exige la existencia de este archivo con extensión *.ino* para compilar y ejecutar cualquier programa.

El archivo principal tiene la estructura estándar de todo firmware de microcontrolador generado con Arduino, compuesto por la función *setup()* que se encarga de inicializar toda variable y componentes del nodo de comunicación, tal como el módulo de comunicación LoRa y la pantalla OLED. Por otro lado, está la función *loop()* que representa el ciclo infinito típico de los sistemas embebidos. En este bucle se gestionan las entradas de usuario desde el monitor serial y las llamadas a funciones repetitivas como la recepción, limpieza de vecinos, envío automático de mensajes de control, etc.

En cuanto a los módulos especializados del *firmware* el primero de ellos es *packet_manager.h.* Este módulo se encarga de definir y manipular las distintas estructuras de paquetes, además de etiquetado de paquetes y nodos. Por otro lado, el módulo de *communication_manager.h* gestiona la transmisión, recepción básica y descarte de paquetes.

Por otra parte, *message_scheduler.h* incluye la cola de planificación para organizar el envío de paquetes y la gestión de reenvíos, mientras que *message_receiver.h* engloba los mecanismos asociados con la recepción como la ventana de escucha y la gestión de duplicados. El módulo *routing_manager.h* mantiene la tabla de vecinos actualizada y ejecuta

el algoritmo de enrutamiento basado en métricas locales como *Received Signal Strength Indicator* (RSSI) y antigüedad del enlace.

También, existen módulos orientados a manipulación del *hardware* de la placa como *lora_manager.h* que configura el *chip* SX1262 para transmisión LoRa y *oled_manager.h* que controla la pantalla OLED integrada. Finalmente, el archivo *config.h* centraliza todos los parámetros de configuración del sistema.

4.2.1. Manejo de paquetes

El mecanismo de manejo de paquetes y sus funciones asociadas con la gestión de los distintos tipos de paquetes están centralizadas en el módulo de *packet_manager.h*, donde también se definen las estructuras de los paquetes DATA, ACK, HELLO y ALT. Cada uno de estos paquetes incluye un conjunto de campos dependiendo de la función de cada paquete, además, como se puede visualizar en la Figura 14 cada paquete tiene un tamaño distinto y sólo comparten la cabecera común que incluye el identificador de tipo (*messageType*), el identificador de malla (*meshID*) y el identificador de mensaje (*messageID*).

El contenido de las variables *messageType* y *meshID* se obtiene de las configuraciones globales del archivo *conf.h.* Por su parte, el campo *messageID* se obtiene mediante una función en *packet_manager.h* llamada *getMessageID()*. La generación del identificador de mensaje se hace mediante la combinación del identificador de nodo, el tipo de mensaje y 8 *bits* aleatorios, lo que asegura la unicidad facilitando la identificación del origen del mensaje.

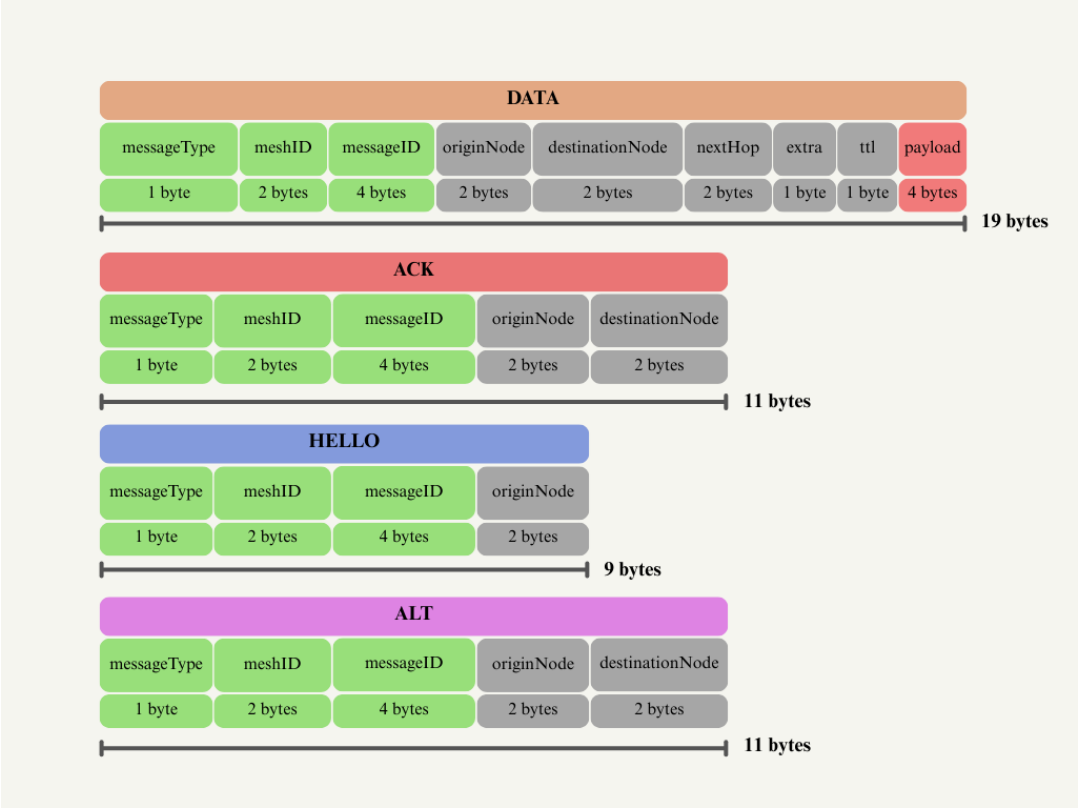

Figura 14 Definición de campos de paquetes y su tamaño en bytes. Elaboración propia.

Por otro lado, este mecanismo implementa funciones para el llenado de las estructuras de los paquetes para prepararlos para su transmisión, garantizando que contengan todos los campos necesarios para el sistema de comunicación. Para el llenado del paquete DATA se recibe como parámetros el nodo de destino (*destinationNode*), nodo de siguiente salto (*nextHop*), campo adicional (*extra*), TTL y el *payload*. En el caso del llenado de los paquetes ACK y ALT se recibe como parámetro solamente el nodo de destino (*destinationNode*) y el identificador de mensaje a confirmar o alertar (*messageID*). Por sí solos los paquetes ACK y ALT no tienen un identificador propio heredan el del mensaje DATA que debe confirmar o alertar de la búsqueda de ruta alterna. Cabe mencionar que el llenado del paquete HELLO no recibe ningún parámetro solamente usa la cabecera común y su identificador de nodo.

Finalmente, para facilitar la transmisión y recepción, los paquetes se serializan (*serializePacket()*) en secuencias de *bytes* y se reconstruyen con funciones de deserialización (*deserializePacket()*). Ambas operaciones se apoyan de la sentencia *switch* que clasifica según el valor del campo *messageType*, definido como constante en el archivo *config.h*. De

esta forma las funciones de serialización y deserialización tratan de distinta forma a cada tipo de paquete y los procesan en función del tamaño total del paquete.

El mecanismo de manejo de paquetes abarca tanto el llenado como la serialización y deserialización de los distintos tipos de mensajes definidos en el sistema. Este proceso es esencial para el correcto funcionamiento de otros mecanismos como la transmisión y recepción de datos, ya que proporciona la estructura base sobre la cual se construyen e interpretan todos los paquetes generados entre nodos. En la Figura 15 se puede visualizar el comportamiento general de este mecanismo.

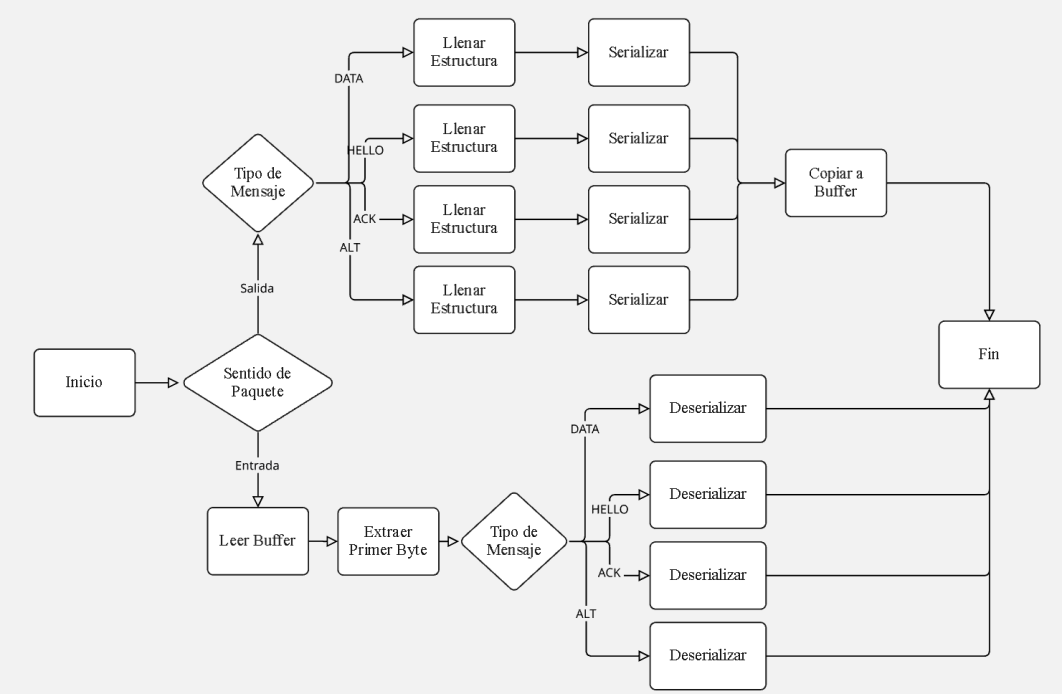

Figura 15 Diagrama de actividades del mecanismo de Manejo de paquetes. Elaboración propia.

4.2.2. Asignación de identificador de nodo

El mecanismo de asignación de identificador de nodo, fundamental para la comunicación en múltiples saltos, se implementa mediante la función *getNodeID()*, ubicada dentro del módulo de *paquet_manager.h*. La ubicación de esta función en dicho modulo ocurre debido a que el identificador de nodo es requerido directamente al momento del llenado de los paquetes, ya que el campo de *originNode* debe establecerse de forma automática. Además, el identificador de nodo es parte del campo de la cabecera común, específicamente el campo *messageID*, lo que lo convierte en un elemento fundamental del identificador para garantizar la unicidad de cada mensaje.

La función *getNodeID()* obtiene un valor único basado en el número de serie físico del *chip* ESP32, empleando internamente la función *getEfuseMac()*. Este número se compacta en un identificador de 16 *bits* mediante una operación XOR entre la mitad superior e inferior del valor original. Este enfoque permite que cada nodo gestione su propia identificación de forma autónoma, sin requerir configuración manual ni externa, y asegurando que el identificador permanezca constante incluso tras reinicios o pérdidas de energía.

Como se puede ver en la Figura 16, el proceso es breve y vincula directamente el identificador de nodo con el hardware físico, proporcionando así una base sólida para la identificación persistente de cada dispositivo dentro de la malla.

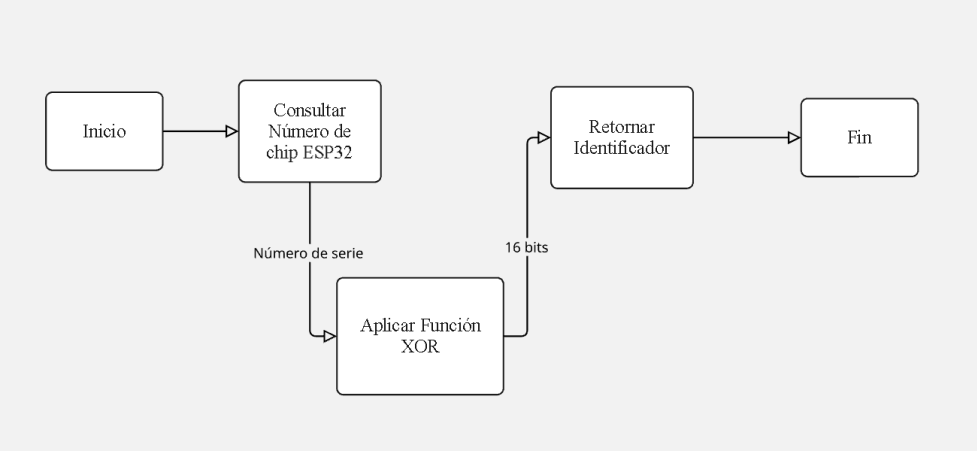

Figura 16 Diagrama de actividades del mecanismo de Asignación de identificador de nodo. Elaboración propia.

4.2.3. Planificador de cola

El mecanismo de planificación de mensajes o encolamiento de paquetes está centralizado en el módulo de *message_schdualer.h*, donde se implementa una cola circular para gestionar el envío tanto de paquetes de datos como de control. Esta cola está compuesta por un arreglo de estructuras *ScheduledItem*, donde cada elemento almacena el paquete pendiente, el tiempo de programación de envío, el tipo de paquete y si está en uso el elemento del arreglo.

El propósito del planificador es evitar colisiones en la red y permitir que el sistema gestione el envío de varios paquetes en un período de tiempo. Para lograr esto se determina un tiempo de espera aleatorio cuyo rango está definido en *config.h* (*INITIAL_WAIT_LOWER* y *INITIAL_WAIT_UPPER*). De esta forma cada paquete planificado será enviado con

tiempos de espera aleatorios lo que reduce la probabilidad de que dos o más nodos transmitan exactamente en el mismo momento.

El planificador también es el mecanismo que asiste a la lógica del reenvío. Tras encolar un paquete de datos también se guarda aquellos paquetes que están a la espera de un ACK en el arreglo *pendingAcks*. Cuando un paquete no recibe su ACK correspondiente dentro de un intervalo (*ACK_TIMEOUT*), el planificador reprograma su envío insertándolo de nuevo en la cola, de esta forma se puede reintentar el envío de un paquete un número finito de veces.

Finalmente, cuando el *chip* LoRa queda libre, es decir no está recibiendo ni trasmitiendo, el planificador mediante un bucle escanea elementos cuyo tiempo de programación haya vencido y lo trasmite con ayuda del módulo de *communication_manager.h* y sus funciones asociadas, liberando un *slot* en la cola y permitiendo la planificación de nuevos paquetes. El funcionamiento completo del mecanismo de encolamiento está detallado en la Figura 17.

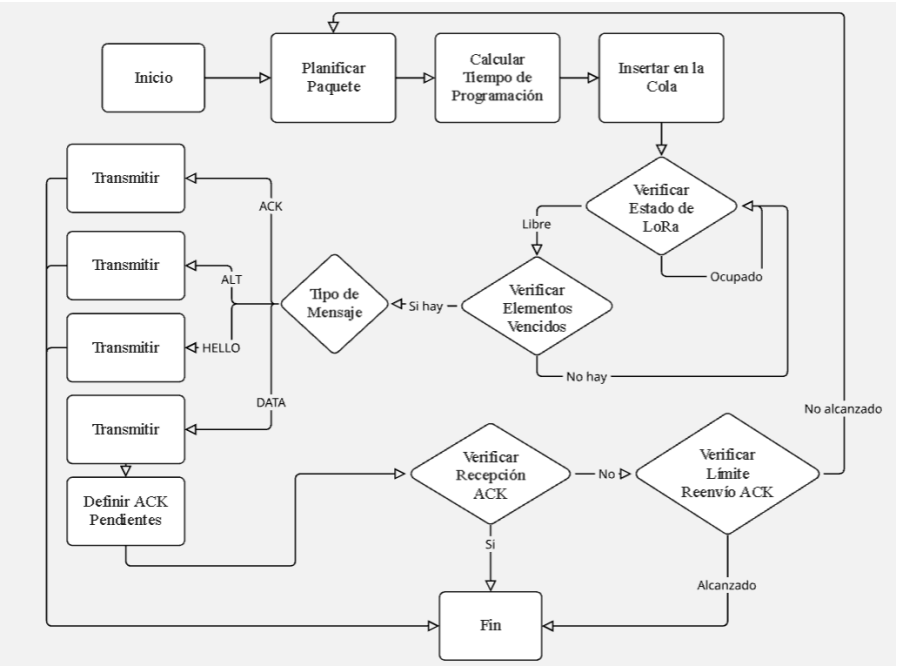

Figura 17 Diagrama de actividades del mecanismo de Planificador de cola. Elaboración propia.

4.2.4. Transmisión de paquetes

El mecanismo de transmisión de paquetes se implementa en el módulo *communication_manager.h*, el cual interactúa directamente con el *chip* LoRa mediante el

objeto *loraAntena*, definido en el módulo *lora_manager.h*. Su función es utilizar el paquete previamente preparado, serializarlo y transmitirlo utilizando la interfaz física de LoRa.

Para esto, el mecanismo de transmisión define la función sobrecargada *handleTransmission()*, que permite enviar los distintos paquetes disponibles. Cada función recibe el paquete del tipo correspondiente, lo serializa usando la función *serializePacket()*, definida en *packet_manager.h*, y lo envía con el método *send()* del objeto *loraAntena*. Inmediatamente, se marca la bandera de disponibilidad como falsa para indicar que el nodo se encuentra ocupado.

Es importante mencionar que los eventos relacionados con la transmisión se controlan mediante funciones asociadas a las interrupciones de LoRa. La función *OnTxDone()* marca la transmisión como completada y establece la bandera de disponibilidad como verdadera, liberando el canal. Por otra parte, *OnTxTimeout()* indica si ha ocurrido un fallo por tiempo de espera y *OnRxDone()* habilita nuevamente la recepción. Estas funciones se asocian al hardware y son llamadas en la configuración inicial del sistema, específicamente en la sección *setup()* propia de los archivos principales de Arduino.

Finalmente, es importante señalar que la transmisión no ocurre de forma inmediata; esta está ligada al planificador de cola y al archivo principal *LoRaMesh.ino*, ya que desde este se planifican los mensajes deseados por el usuario y, periódicamente, los paquetes de control HELLO.

El funcionamiento general del mecanismo de transmisión de paquetes se encuentra representado en la Figura 18, donde se visualiza que el flujo inicia con la verificación del tipo de paquete (DATA, ACK, HELLO o ALT), seguido por su serialización, transmisión y marcación del canal como ocupado. Finalmente, dependiendo del resultado del evento de transmisión, se actualiza el estado del canal y se concluye el proceso marcando nuevamente a LoRa como disponible.

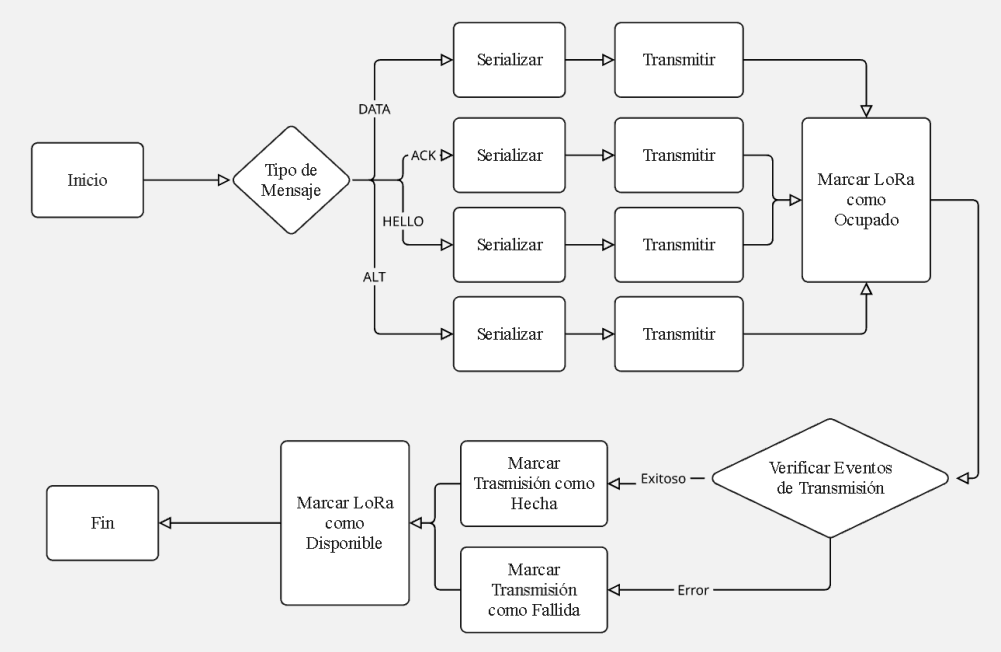

Figura 18 Diagrama de actividades del mecanismo de Transmisión de paquetes. Elaboración propia.

4.2.5. Recepción de paquetes

La lógica del mecanismo de recepción de paquetes se implementa principalmente en el módulo de *communication_manager.h*, y su ejecución se inicia mediante la función *handleReception()*, la cual activa el *chip* LoRa para comenzar a escuchar mensajes. Cuando una transmisión de paquete es recibida, se activa la función asociada con eventos de hardware *OnRxDone()*, esta automáticamente copia los datos en el arreglo *receivedBuffer*, registrando también el RSSI y el tamaño del *buffer* recibido.

Tras una interrupción por recepción, la función *processPayload()*, se encarga de obtener el tipo de mensaje desde el primer *byte* del *buffer* (*messageType*), lo deserializa usando *deserializePacket()*, y determina su tratamiento según su tipo. En el caso de los mensajes DATA, además de verificar duplicados y TTL, se reetiqueta el paquete si debe reenviarse y se programa un ACK con *scheduleAckMessage()*. Para ACK, se elimina su *messageID* de la lista de pendientes y se marca como recibido. Los paquetes HELLO y ALT actualizan la tabla de vecinos o buscan rutas alternas según el caso.

Complementando este mecanismo, el módulo *message_receiver.h* implementa la función *processReceivedMessage()*, que comprueba si la bandera de *receptionDone* está activa. Si es así, llama a *processPayload()*, restablece la bandera a *false* para habilitar futuras recepciones y evitar que el proceso sea bloqueante. El funcionamiento general de este

mecanismo se resume en la Figura 19, donde se muestra cómo se inicia la recepción, se almacena el paquete, por seguridad se verifica si el mecanismo de recepción está activo, se evalúa su tipo, se procesa conforme a las reglas del sistema y se habilita nuevamente la recepción.

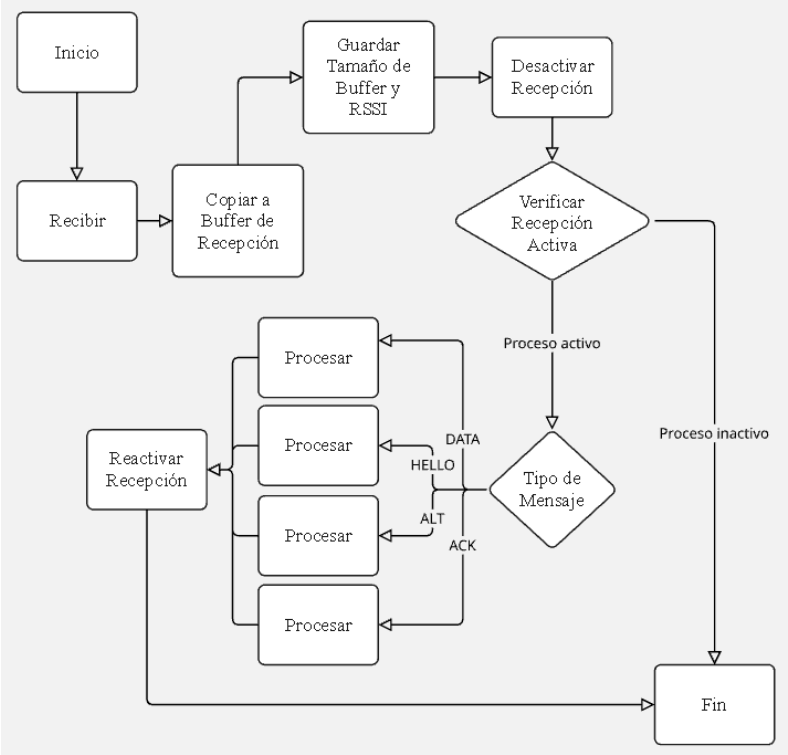

Figura 19 Diagrama de actividades del mecanismo de Recepción de paquetes. Elaboración propia.

4.2.6. Descarte de paquetes

En cuanto al mecanismo de descarte de paquetes, este se implementa dentro del módulo de *communication_manager.h* y está directamente vinculado con el mecanismo de recepción. Cada que un nodo recibe un paquete el mecanismo de descarte funciona como una especie de filtro, realizando una serie de verificaciones para determinar si el paquete debe ser descartado o procesado. Esta validación se realiza mediante funciones como *dropPacket()*, *dropAckPacket()*, *dropHelloPacket()* y *dropAltPacket()*, según el tipo de mensaje recibido.

Existen varios criterios para no descartar tales como la verificación si el campo de *meshID* coincide con el identificador de la red según el archivo *config.h*, si el TTL aún no ha expirado, si el nodo receptor es el destino (*destinationNode*) o si es el próximo salto (*nextHop*) del mensaje. Específicamente, la validación conjunta de los campos *nextHop* y

destinationNode garantiza que el mensaje siga la ruta establecida y no sea procesado por un nodo que no le corresponde, previniendo así que termine su trayecto de forma errónea por una mala propagación o interferencia.

Es relevante mencionar que la única validación común entre todos los tipos de paquete es la validación del identificador de malla. Al filtrar por el valor del campo *meshID*, se garantiza que cada nodo sólo procese los mensajes dirigidos a su propia red, evitando interferencias entre diferentes mallas que operen bajo el mismo sistema de comunicación.

El funcionamiento general del mecanismo puede visualizarse en la Figura 20 donde se muestra que el proceso inicia con un paquete recibido, continúa evaluando el tipo de mensaje, donde dependiendo del tipo y de los campos que tenga este se aplican los distintos parámetros de descarte. Estas verificaciones aseguran que sólo se procesen los mensajes relevantes, se desechen paquetes incorrectos, evitando reenvíos innecesarios, conservando recursos, y manteniendo la integridad del sistema de comunicación.

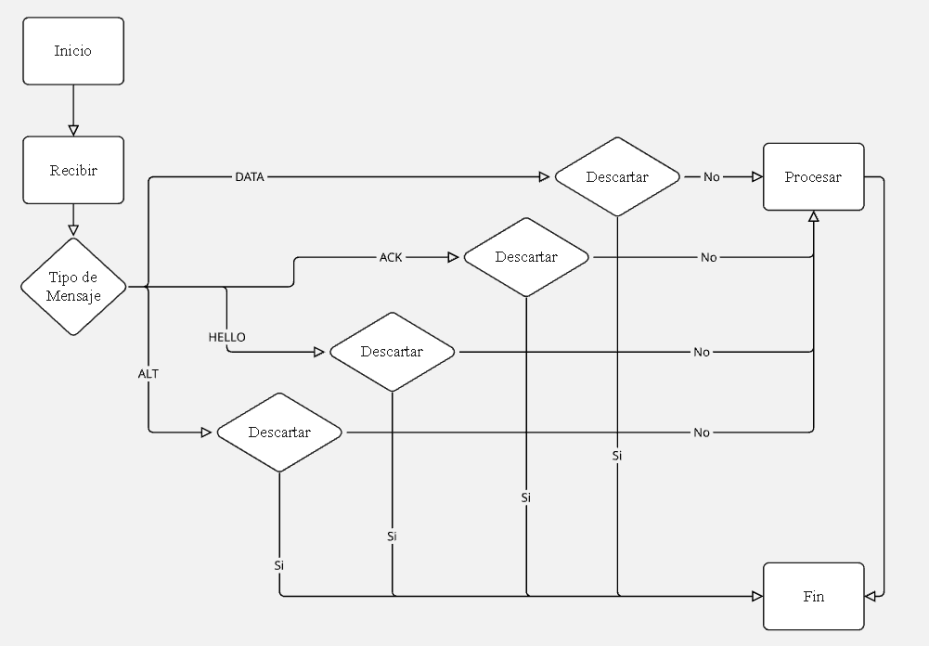

Figura 20 Diagrama de actividades del mecanismo de Descarte de paquetes. Elaboración propia.

4.2.7. Ventana de escucha

La ventana de escucha es un mecanismo preventivo basado en LBT, diseñado para prevenir colisiones justo antes de cada transmisión. Aunque su lógica está implementada en el archivo *message_receiver.h* ya que este mecanismo es un proceso de recepción, su

ejecución ocurre en *message_scheduler.h*, específicamente antes de que se transmita cualquier mensaje programado en la cola.

Este mecanismo, implementado en la función *windowCollisionPrevention()*, activa la recepción durante una pequeña ventana temporal definida por *LISTEN_WINDOW_MS* en *config.h*. Durante este intervalo, el sistema mantiene la variable *receptionDone* en estado *false* e inicia un bucle de recepción continua. Si durante este período de tiempo se recibe un paquete se prioriza la ejecución de la función *processReceivedMessage()* y se para la transmisión. Este proceso de intento de transmisión se repite un número finito de veces definido en *MAX_WINDOW_RETRIES*, encontrado en el archivo *config.h*. En el caso que los intentos se agoten se fuerza la transmisión, de esta manera se evita que el mecanismo sea bloqueante.

Este enfoque es una versión simplificada y ligera de LBT, similar a CSMA donde, así como se puede observar en la Figura 21, el proceso comienza cuando hay un mensaje planificado para ser trasmitido, se escucha el canal y se toma la decisión de reenviar o reintentar después de un período de tiempo.

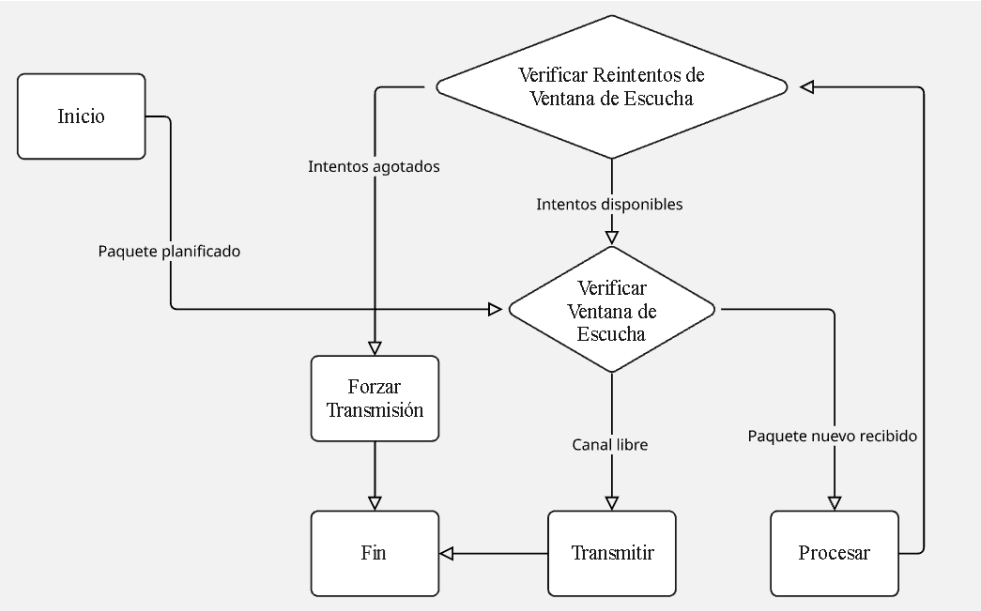

Figura 21 Diagrama de actividades del mecanismo de Ventana de escucha. Elaboración propia.

4.2.8. Confirmación *hop-by-hop*

El mecanismo de confirmación *hop-by-hop* es aquel que permite verificar que los paquetes tipo DATA hayan llegado correctamente a su destino. Esta lógica está

implementada en el módulo *message_scheduler.h*, mediante el uso de la estructura *PendingAck* y el arreglo *pendingAcks*, que actúa como una tabla de paquetes pendientes de confirmación.

Después de cada transmisión de datos exitosa, el contenido del paquete se almacena en el arreglo *pendingAcks* mediante la función *addPendingAck()*, junto con una marca de tiempo y un contador de reintentos. Esto permite llevar una referencia para calcular el tiempo trascurrido mientras se espera la confirmación del paquete mediante un ACK, enviado por el nodo receptor inmediato o vecino receptor. En el caso que si llegue un ACK el contenido es eliminado del arreglo de *pendingAcks* para evitar esperar por una confirmación ya recibida.

La función *updateMessageScheduler()* se encarga de escanear periódicamente el arreglo de pendientes, evaluando si alguno ha excedido el tiempo máximo de espera definido por *ACK_TIMEOUT*. En el caso de que sobrepase el tiempo se asume que el paquete no llegó y se reprograma nuevamente en el planificador de cola, reiniciando el temporizador e incrementando el contador de reintentos. Si el paquete sobrepasa el contador de reintentos, es decir no recibió nunca un ACK, se considera que el nodo no es accesible. El funcionamiento de todo el mecanismo se representa en la Figura 22.

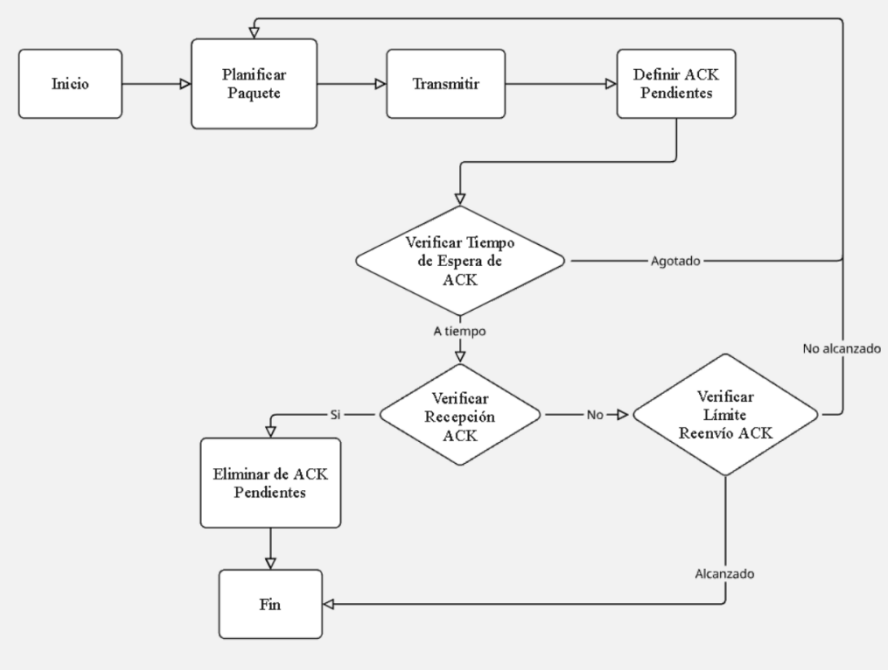

Figura 22 Diagrama de actividades del mecanismo de Confirmación hop-by-hop. Elaboración propia.

Es importante mencionar que este enfoque permite que cada nodo verifique individualmente la recepción correcta de sus paquetes por el nodo siguiente, sin necesidad de que el destino final confirme todo el trayecto, lo cual permite mantener simplicidad en entornos microcontroladores y además permite desocupar cuanto antes a cada nodo. Además, el sistema complementa al planificador de cola, ya que los mensajes reprogramados por fallos en el ACK son reinsertados al flujo normal de transmisión.

4.2.9. Historial de mensajes duplicados

En el módulo *message_receiver.h* se encuentra implementado el mecanismo de historial de mensajes duplicados, este emplea un arreglo circular (*messageIDHistory*) que almacena los identificadores de los mensajes recibidos recientemente. Cada que se recibe un nuevo paquete, una función de verificación de duplicados *(checkDuplicates())* se encarga de comparar el identificador del paquete recibido con el historial de mensajes y en función del valor booleano retornado se toma una decisión para evitar bucles o que el paquete fluya por una ruta ya recorrida.

Para asegurar que sólo se registren como duplicados los paquetes realmente transmitidos y confirmados, el sistema emplea la función *addMessageIDAfterAck()*. Esta se invoca únicamente cuando se recibe un ACK válido, garantizando así que únicamente los mensajes exitosamente entregados sean añadidos al historial. Internamente, esta función utiliza *addMessageID()* para insertar el *messageID* correspondiente en el arreglo circular.

Esta implementación impide que se marquen como duplicados aquellos paquetes que están siendo reenviados como parte del mecanismo de confirmación *hop-by-hop*, pero que aún no han recibido respuesta. De esta forma, se evitan descartes erróneos y se mantiene la lógica de reintentos funcionando correctamente. Es importante destacar que el llamado a estas funciones se gestiona desde el módulo *communication_manager.h*, que coordina tanto la recepción como el procesamiento de mensajes.

El funcionamiento de este mecanismo auxiliar se representa en la Figura 23 donde se representa cómo el mecanismo trabaja en dos fases tanto en la fase de transmisión, donde un mensaje correctamente enviado y procesado es registrado en el historial de duplicados, y en la fase de recepción donde se verifica si el mensaje recibido ya ha sido procesado.

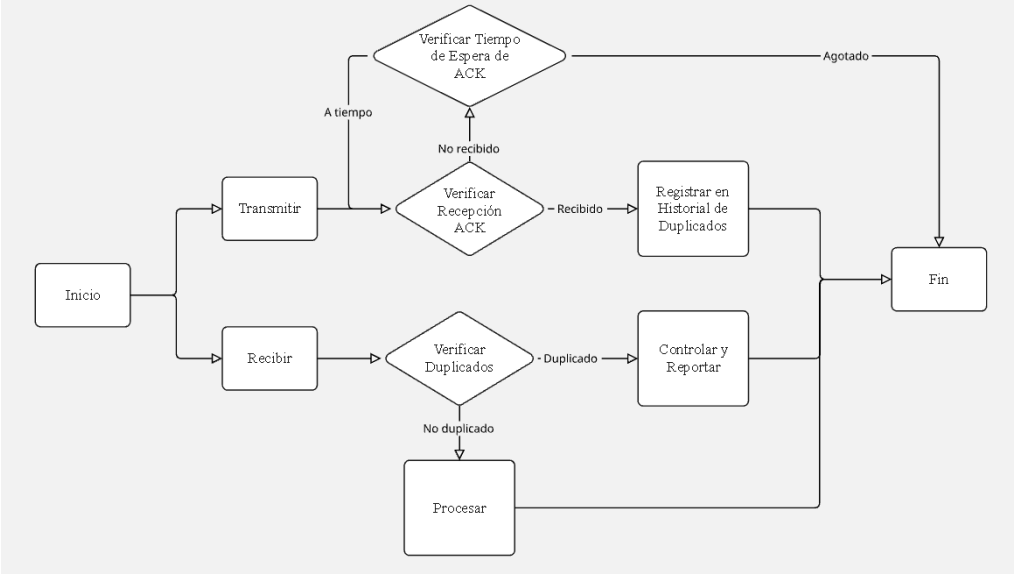

Figura 23 Diagrama de actividades del mecanismo de Historial de mensajes duplicados. Elaboración propia.

4.2.10. Construcción y depuración de la tabla de vecinos

El mecanismo de construcción y depuración de la tabla de vecinos se implementa en el módulo *routing_manager.h*, donde también se define dicha tabla como un arreglo de estructuras de tipo *NeighborInfo*. Esta estructura, representada en la Figura 24, incluye el identificador del nodo (*neighborId*), la intensidad de la señal recibida (rssi) y el instante de tiempo (*lastHeard*) en el que se recibió el último paquete tipo HELLO.

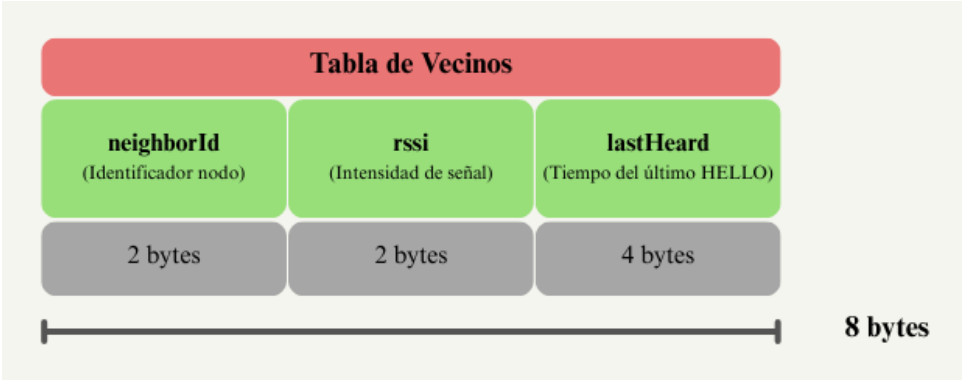

Figura 24 Estructura y campos de la tabla de vecinos. Elaboración propia.

En cuanto al mantenimiento de esta tabla, este depende de la función *loop()* del archivo principal, donde se invoca la función *checkAutoHello()* para programar periódicamente el envío automático de paquetes HELLO y mantener la tabla actualizada con los vecinos activos. Cabe recalcar que, cuando un nodo recibe un paquete HELLO, este se procesa en el módulo *communication_manager.h* y, si es válido, se actualiza la tabla

llamando a la función *addOrUpdateNeighbor()*, también ubicada en *routing_manager.h*. Dicha función permite añadir un nuevo vecino o actualizar la información de uno ya registrado.

Finalmente, la depuración de la tabla se realiza mediante la función *cleanupNeighbors()*, también invocada desde el *loop()* del archivo principal. Esta función revisa el tiempo transcurrido desde la última señal HELLO recibida por cada vecino, y elimina aquellos nodos que superen el umbral de expiración (*NEIGHBOR_EXPIRATION_TIME*), evitando así que se mantenga información desactualizada o inválida.

En la Figura 25 se representa cómo el proceso de planificación de envíos de paquete HELLO y la verificación de antigüedad nunca culmina, esto al estar involucrados en el bucle principal (*loop()*). Por otro lado, el único proceso que no se ejecuta de forma cíclica es el proceso de añadir o actualizar la tabla de vecinos tras la recepción de un paquete HELLO.

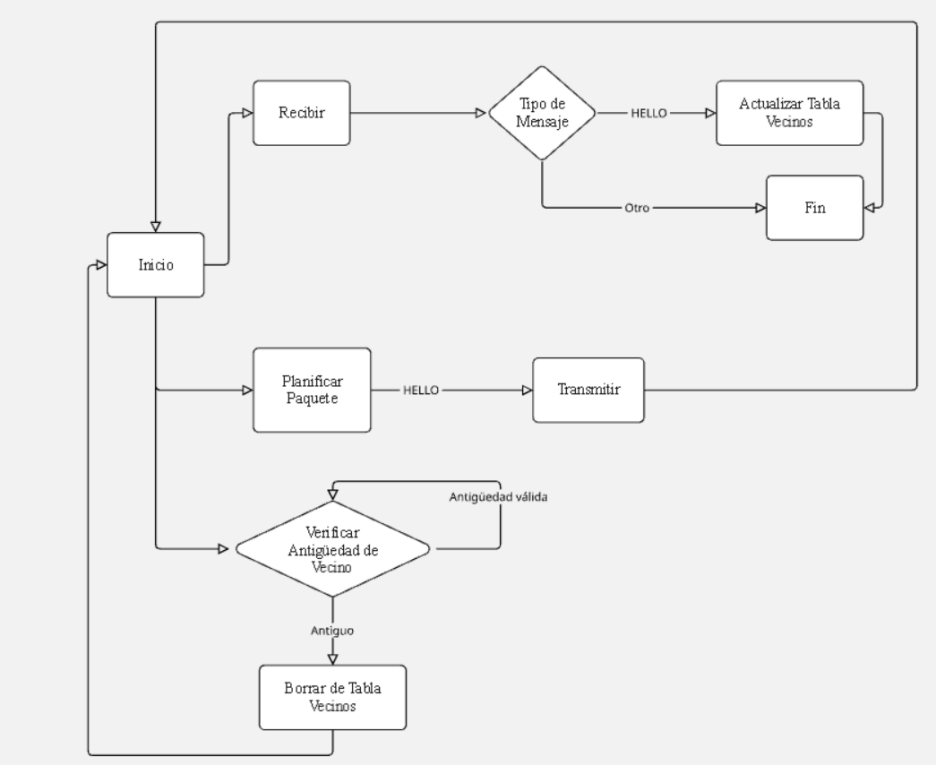

Figura 25 Diagrama de actividades del mecanismo de Construcción y depuración de la tabla de vecinos. Elaboración propia.

4.2.11. Enrutamiento basado en vecinos

El módulo de *routing_manager.h* al englobar todas las funciones orientadas al enrutamiento también contiene la implementación del mecanismo de enrutamiento basado en vecinos, el cual depende directamente del mecanismo de construcción y depuración de la tabla de vecinos ya que emplea la información de este mismo como insumo para tomar decisiones dinámicas sobre el siguiente salto *(nextHop)*.

La función principal es *getNextHop()*, que evalúa todos los vecinos disponibles, si el destino final está entre los vecinos retorna directamente a dicho nodo como siguiente salto y si no está, le asigna a cada vecino válido un puntaje mediante la función *getNeighborScore()*. La verificación previa del destino entre los vecinos permite evitar todo el proceso de ponderación y selección, ahorrando recursos del microcontrolador, esto puede observarse en la Figura 26, que muestra cómo se realiza primero la búsqueda directa del destino antes de aplicar el cálculo de candidatos.

El cálculo de la puntuación se realiza restando el RSSI al tiempo transcurrido desde el último mensaje HELLO. Dado que el RSSI es un valor negativo, donde un valor más cercano a cero indica mejor señal, al restarle el valor de tiempo, el resultado es positivo. En este caso cuanto más cercano sea el valor a cero hay una mejor puntuación ya que esto significa la combinación de una buena señal y una recepción reciente de un HELLO.

Tras el cálculo de la puntuación, la lógica de la función *getNextHop()* selecciona aleatoriamente uno del grupo de mejores candidatos para actuar como siguiente destino. El número de candidatos para escoger está definido en *conf.h* (*ROUTING_MAX_CANDIDATES*).

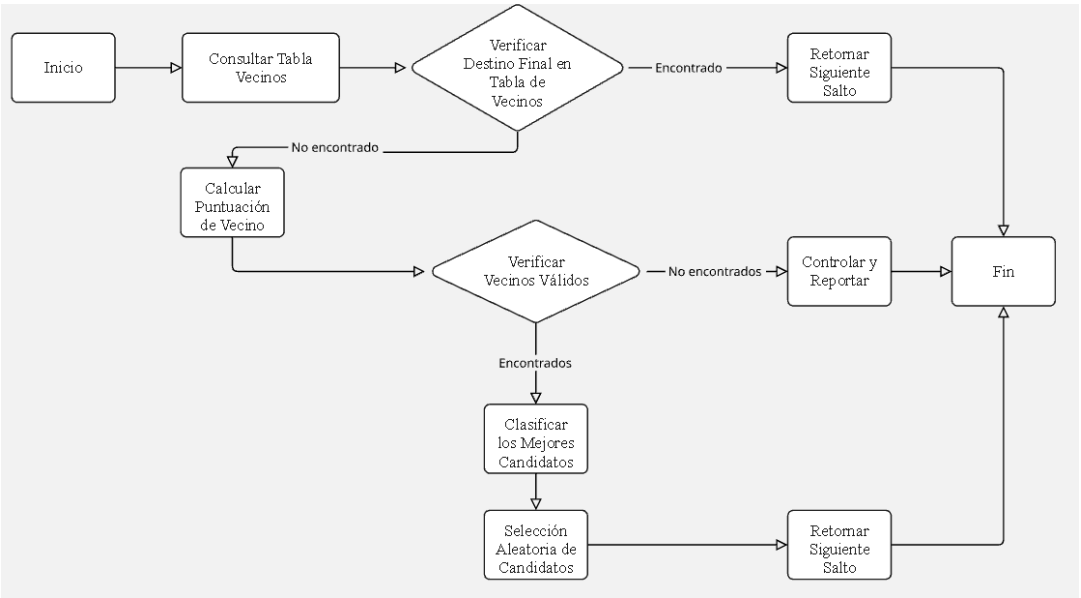

Figura 26 Diagrama de actividades del mecanismo de Enrutamiento basado en vecinos. Elaboración propia.

Cabe destacar que, la decisión de enrutamiento se aplica en el momento de planificar un paquete para ser transmitido, ya sea un mensaje nuevo o un reenvío. Esta lógica se encuentra integrada en el planificador de cola definido en *message_scheduler.h*, específicamente en la función *enqueueDataMessage()*. En esta función, antes de encolar el paquete, se realiza una llamada a *getNextHop()* para asignar el destino inmediato o siguiente salto de ese paquete.

4.2.12. Reenvío por saltos

El reenvío por saltos es un mecanismo clave para el funcionamiento del sistema de comunicación. Pueden existir dos ligeras variaciones de comportamiento, en el caso de que el nodo sea quien origina el mensaje, llenando y etiquetando el paquete para enviarlo al siguiente salto; o cuando un nodo intermedio recibe un paquete, lo procesa, lo reetiqueta y lo vuelve a enviar hacia un siguiente salto. Este mecanismo depende del enrutamiento basado en vecinos y de los mecanismos de control, como el descarte de paquetes y el historial de mensajes duplicados, para evitar reenviar paquetes inválidos. La lógica de transmisión de paquetes que involucren reenvío por varios saltos se encuentra implementada principalmente en el módulo *communication_manager.h* y se relaciona con el planificador de cola en *message_scheduler.h*.

Cuando un nodo origina el paquete, este sigue el proceso estándar de manejo de paquetes, lo llena, lo serializa y lo transmite. Durante el llenado, se etiqueta el campo

correspondiente al siguiente salto utilizando la función de enrutamiento *nextHop()* y el nodo se identifica a sí mismo como origen.

Cuando un nodo recibe un paquete de datos y tras ser evaluado como válido se procesa y al no ser el nodo destino se comporta como repetidor, reetiquetando el paquete asignándose como nuevo nodo de origen para poder recibir el ACK. También, se calcula un nuevo salto con la función de enrutamiento *nextHop()* excluyendo al nodo emisor original para evitar bucles y se resta el TTL.

El paquete reetiquetado recibe el mismo tratamiento que cualquier paquete en trasmisión, es decir el mecanismo de planificador de cola se encarga de él. Para dicho mecanismo es irrelevante si se trata de un paquete generado por el mismo nodo o un paquete reetiquetado de otro nodo. El proceso general de este mecanismo se resume en la Figura 27, donde se ve los dos posibles flujos de comportamiento, tanto en el caso que se recibe como en el caso que se llena inicialmente el paquete debido a que es el nodo emisor original.

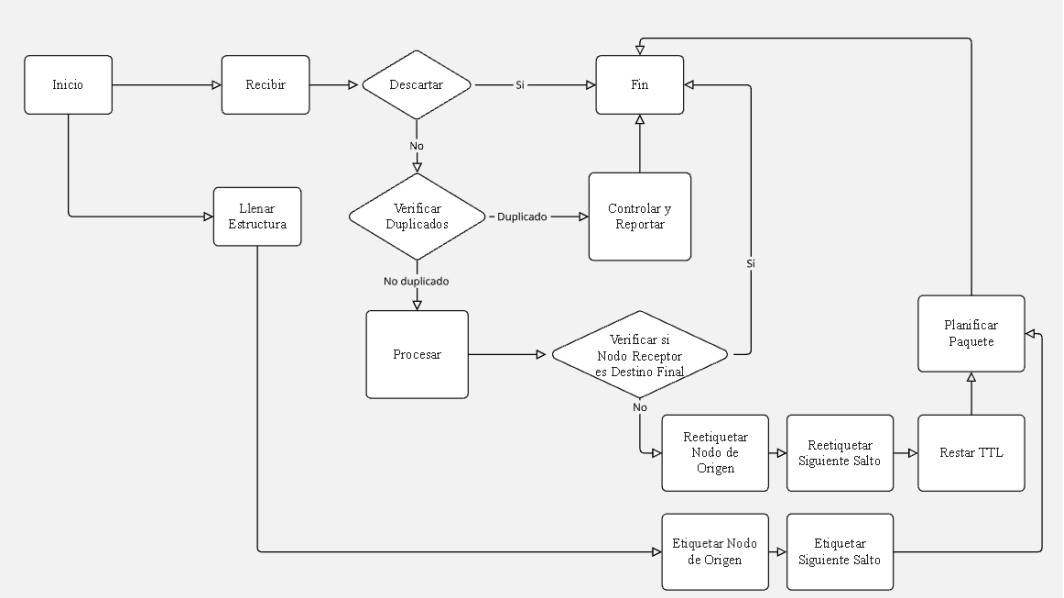

Figura 27 Diagrama de actividades del mecanismo de Reenvío por saltos. Elaboración propia.

4.2.13. Reconvergencia tras fallo de ACK

El mecanismo de reconvergencia inicia cuando un paquete DATA no ha recibido su correspondiente ACK tras el número máximo (*MAX_RETRIES*) de intentos configurados en *config.h*. Este comportamiento está implementado en el módulo *message_schedualer.h*,

dentro de la función de *updateMessageSchedualer()*, esto debido a que requiere un reencolamiento, pero en otra ruta.

Debido a que, si falla tras el número finito de intentos, se requiere la búsqueda de otra ruta mediante la función *reEnqueueAlternateRoute()*, la cual involucra el proceso para recalcular una nueva ruta hacia el destino, excluyendo al nodo que falló en confirmar la recepción. También, se elimina el nodo fallido de la tabla de vecinos usando la función *removeNeigbor()* ya que se asume que el nodo fallido es inaccesible. Para mantener la comunicación, se llama a la función *getNextHop()* pasando el destino original (*destinationNode*) y el nodo fallido (*excludeNeighbor*) como parámetro, lo que permite encontrar una nueva ruta de reenvío que no dependa del camino que falló.

En el caso que si se encuentre una ruta válida el paquete es reenviado hacia el nuevo salto (*newHop*) tras ser reencolado en el planificador de cola. En el caso de que no exista una ruta válida ya sea porque no hay vecinos disponibles o porque el único vecino disponible es el nodo que falló se asume que el nodo emisor está aislado y se descarta el paquete, reportando el error.

Este mecanismo actúa como un proceso de autocuración de la red, permitiendo que el sistema intente de forma autónoma otras rutas disponibles. El funcionamiento general se describe en la Figura 28, donde se visualiza que tras no recibir un ACK dentro del límite de reenvío (*MAX_RETRIES*), el sistema verifica las rutas válidas e intenta planificar el paquete para un próximo envío en la nueva ruta.

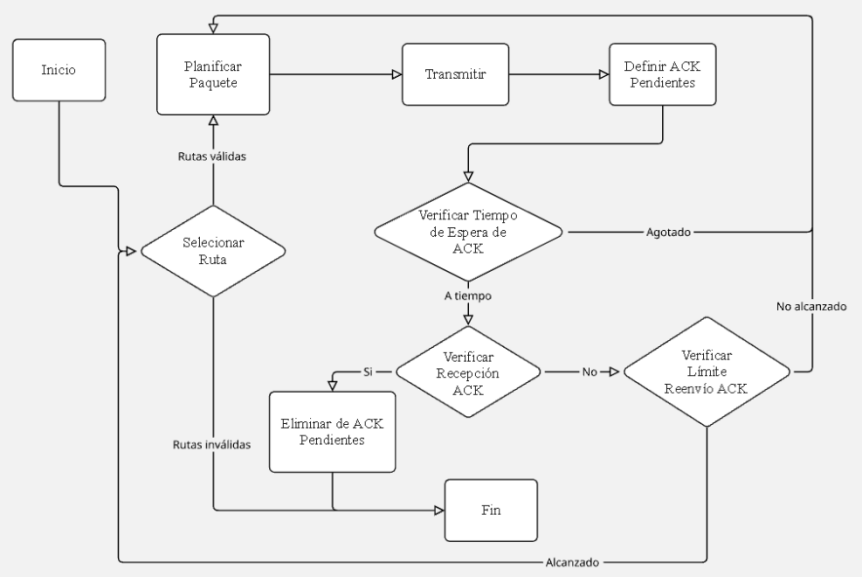

Figura 28 Diagrama de actividades del mecanismo de Reconvergencia tras fallo de ACK. Elaboración propia.

4.2.14. Alerta de ruta alternativa

El sistema implementa un mecanismo de alerta de ruta alternativa mediante paquetes ALT, este tiene el propósito de notificar al nodo emisor que el receptor ya ha procesado alguna vez el paquete recibido y que el emisor debe seleccionar otra ruta para evitar bucles. Este mecanismo se activa con ayuda del mecanismo de historial de mensajes duplicados que notifica cuando un paquete ya ha sido procesado. La planificación del paquete asociado a este mecanismo se encuentra implementado en *message_schedualer.h* pero se la llama en el módulo *communication_manager.h*.

Cuando el mecanismo de historial de mensajes duplicados identifica un paquete ya procesado se llama a la función *scheduleAltMessage()*, que encola un paquete ALT dirigido al nodo que originó el paquete original. Este mensaje ALT es procesado como cualquier otro en el planificador de cola y sigue su curso hasta llegar al nodo origen.

Una vez recibido el paquete ALT, el nodo que originalmente transmitió el paquete lo busca en el arreglo de ACKs pendientes (*pendingAcks*) o elimina para no esperar un ACK del nodo que envío el ALT y llama a la función *reEnqueueAlternateRoute()*. Esta función genera una nueva ruta excluyendo al nodo que envió el ALT, lo que permite reenviar el paquete mediante un nuevo salto, un proceso idéntico al mecanismo de reconvergencia tras fallo de ACK. El funcionamiento descrito puede ser visualizado en la Figura 29.

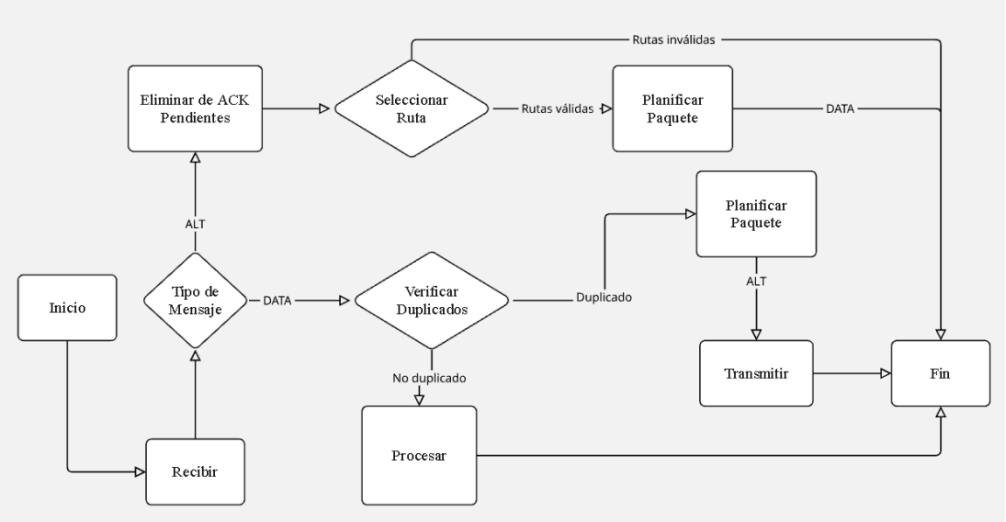

Figura 29 Diagrama de actividades del mecanismo de Alerta de ruta alternativa. Elaboración propia.

CAPÍTULO V: Pruebas y Análisis de Resultados

Este capítulo presenta la validación experimental de la implementación del prototipo de sistema de comunicación multi-nodo, evaluando su comportamiento bajo distintos escenarios de operación. Las pruebas se realizaron sobre una topología *mesh* parcial compuesta por microcontroladores ESP32 y módulos LoRa SX1262, la cual involucra un total de tres saltos para la comunicación de extremo a extremo.

Dichas pruebas se plantean en base a tres escenarios cuyo objetivo es analizar el comportamiento del sistema en condiciones ideales, ante la falla de un nodo intermedio y en un entorno con mayor separación física entre nodos. En cada escenario se evalúan métricas clave como la Tasa de Entrega de Paquetes (PDR), la Tasa de Recuperación ante Fallos (PRR) y el número total de retransmisiones necesarias para asegurar la comunicación *end-to-end*.

A lo largo de este capítulo se describe el entorno de prueba, la metodología de evaluación y recolección de datos, así como el planteamiento de los escenarios definidos. Posteriormente, se presenta la documentación de los resultados experimentales y un análisis detallado que permite interpretar el comportamiento de la red en términos de efectividad, resiliencia y viabilidad operativa. Este análisis constituye la base para valorar el desempeño del sistema y sustentar una posible extensión futura del prototipo.

5.1. Descripción del entorno de pruebas

Las pruebas del prototipo de sistema de comunicación se plantean en un entorno controlado, utilizando microcontroladores ESP32 con módulos LoRa SX1262. La topología está compuesta por un total de cinco nodos, formando una *mesh* parcial, así como se muestra en la Figura 9 y 30, con la finalidad de que se requieran tres saltos para comunicar los extremos. Es importante mencionar que para mantener la consistencia de la topología se emplea una restricción lógica de visibilidad entre nodos implementada por software, así evitando enlaces no planificados y forzando una comunicación de extremo a extremo mediante tres saltos obligatoriamente.

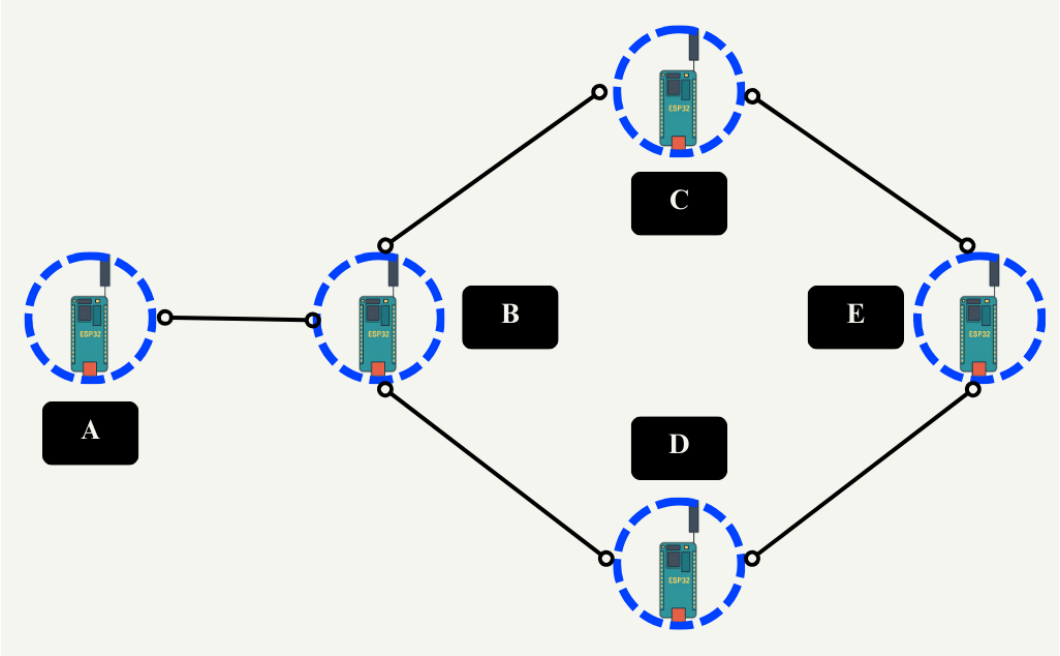

Figura 30 Topología mesh parcial para pruebas. Elaboración propia.

Las pruebas se establecen con dos variaciones del entorno físico. En primer lugar, se plantea un entorno físico limitado en cuanto a distancia y espacio, con el fin de facilitar el montaje del prototipo y reducir la complejidad logística, lo cual permite el análisis y la evaluación del comportamiento base y de los distintos mecanismos del prototipo de sistema de comunicación.

En la segunda variación del entorno físico se plantea la misma topología, pero con una mayor separación entre nodos. Esto permite un acercamiento a un entorno de funcionamiento más real y la validación de la capacidad del prototipo para mantener la

comunicación a mayor distancia, aprovechando una de las características clave de la tecnología LoRa.

Los parámetros de configuración de todo nodo usado en el entorno de pruebas se pueden apreciar en la Tabla 1. Dichos parámetros se encuentran en el archivo *config.h* del código fuente.

Tabla 1 *Parámetros de configuración.*

Parámetro	Valor
Frecuencia de operación	915 MHz
Ancho de banda	125 kHz
SF	SF7
Potencia de transmisión	5 dBm
TTL inicial	5 saltos

5.2. Metodología

La metodología definida para el prototipo involucra un enfoque experimental y controlado, orientada en evaluar el comportamiento del sistema de comunicación multi-nodo basado en LoRa bajo distintos escenarios de funcionamiento en una misma topología, ya sea en condiciones de funcionamiento normales o en condiciones de falla de un nodo intermedio.

Durante las pruebas se establece la medición de ciertas métricas que van a permitir evaluar tanto la comunicación entre nodos como los mecanismos involucrados. Es importante destacar que la selección de estas métricas no sólo responde a la necesidad de verificar el correcto funcionamiento del sistema de comunicación, sino que se basa en su capacidad para reflejar el desempeño específico de los mecanismos diseñados e implementados. Las métricas están enfocadas en eventos clave del proceso de comunicación, como la entrega exitosa de paquetes, las retransmisiones o la recuperación ante fallos, lo cual permite identificar de forma indirecta la efectividad o posibles deficiencias en mecanismos.

Cabe destacar que no se incluyeron métricas temporales como latencia, *jitter* o tiempo de retransmisión, ya que el sistema está diseñado para operar de forma descentralizada sin requerir sincronización entre nodos tal como lo establece Pietrzak et al. (2024). Medirlas con precisión implicaría mecanismos adicionales que aumentarían innecesariamente la complejidad del prototipo. Las pruebas del prototipo se enfocan en

evaluar la efectividad y robustez del sistema, no su rendimiento temporal ni su aplicación en entornos de tiempo real. Las métricas seleccionadas a evaluar son:

- **Tasa de entrega de paquetes (PDR):** Métrica adaptada del trabajo sobre AODV de Umamaheswari et al. (2021). Es la relación porcentual entre la cantidad de paquetes recibidos y paquetes enviados, utilizada como medida para evaluar efectividad del sistema.

$$PDR = \frac{Paquetes\ recibidos}{Paquetes\ enviados} x100$$

- **Tasa de recuperación ante fallo (PRR):** Métrica adaptada del estudio de métodos de restauración en sistemas digitales de conexión cruzada elaborado por Doverspike & Wilson (1994). Es la relación porcentual de paquetes que logran entregarse exitosamente luego de que ocurre una falla en un nodo intermedio y que se los enruta por un enlace alternativo. Métrica empleada para la evaluación de reconvergencia.

$$PRR = \frac{Paquetes\ recuperados\ tras\ falla}{Paquetes\ esperados\ tras\ la\ falla} x100$$

- **Número de retransmisiones totales:** Es la cantidad total de reintentos que realiza el sistema de comunicación para reenviar un mismo paquete hasta recibir un ACK en cada enlace del trayecto *end-to-end*. Esta métrica permite evaluar el sistema en términos de pérdidas temporales de conexión, considerando la totalidad de los enlaces de la ruta y no sólo el comportamiento *hop-by-hop*.

Cabe recalcar que, en cuanto al proceso de medición de las métricas, se define un total de cinco repeticiones para cada escenario, con el objetivo de observar el comportamiento del sistema. Esta decisión se basa en el enfoque propuesto por Montgomery (2013), quien utiliza un diseño experimental de cinco repeticiones para una evaluación industrial. Esta decisión permite obtener resultados más confiables al evaluar métricas minimizando el impacto de factores externos no controlados que podrían sesgar la medición.

En cada repetición se envían dos paquetes por cada nodo extremo, es decir, dos mensajes desde el nodo A hacia el nodo E y dos mensajes desde el nodo E hacia el nodo A, sumando cuatro transmisiones por repetición. Este enfoque experimental permite visualizar el comportamiento general del sistema, así como las posibles variaciones de las métricas.

5.3. Escenarios de pruebas

Las pruebas se llevan a cabo mediante tres escenarios que permiten la evaluación del funcionamiento del sistema y sus mecanismos de comunicación bajo distintas condiciones. Los escenarios comparten la misma topología, pero difieren en la distancia física entre los nodos y en la presencia de fallos en uno de los nodos intermedios. Esto permite analizar la comunicación tanto en condiciones ideales como en condiciones de mayor separación física, así como en situaciones especiales donde se pone a prueba la reconvergencia y la resiliencia del sistema.

5.3.1. Escenario 1: Comunicación bidireccional sin fallos

El primer escenario planteado tiene un enfoque en una comunicación en condiciones ideales, donde todos los nodos de la red están funcionando adecuadamente. En dicho escenario se establece una comunicación bidireccional entre los dos extremos de la red, involucrando transmisiones de tres saltos.

El flujo de paquetes ocurre en ambas direcciones simultáneamente. Así como se puede ver en la Figura 31, la comunicación se lleva a cabo en el sentido del nodo A hacia el nodo E y del nodo E al nodo A.

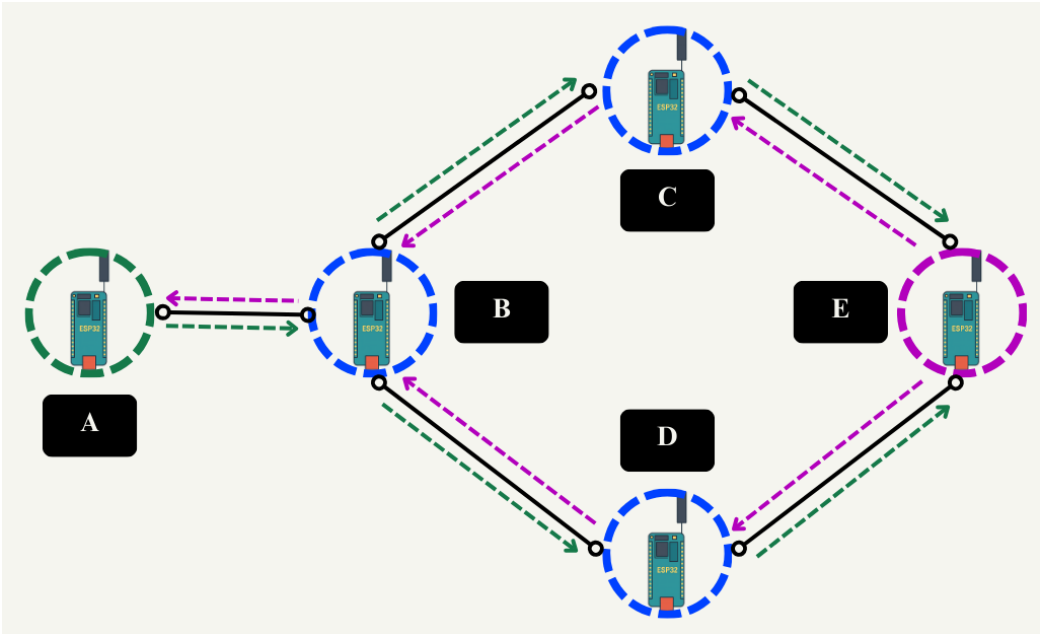

Figura 31 Flujo de paquetes Escenario 1. Elaboración propia.

La implementación física de este escenario puede observarse en la Figura 32, la cual muestra la disposición real de los nodos durante la prueba, replicando la topología presentada en el diagrama.

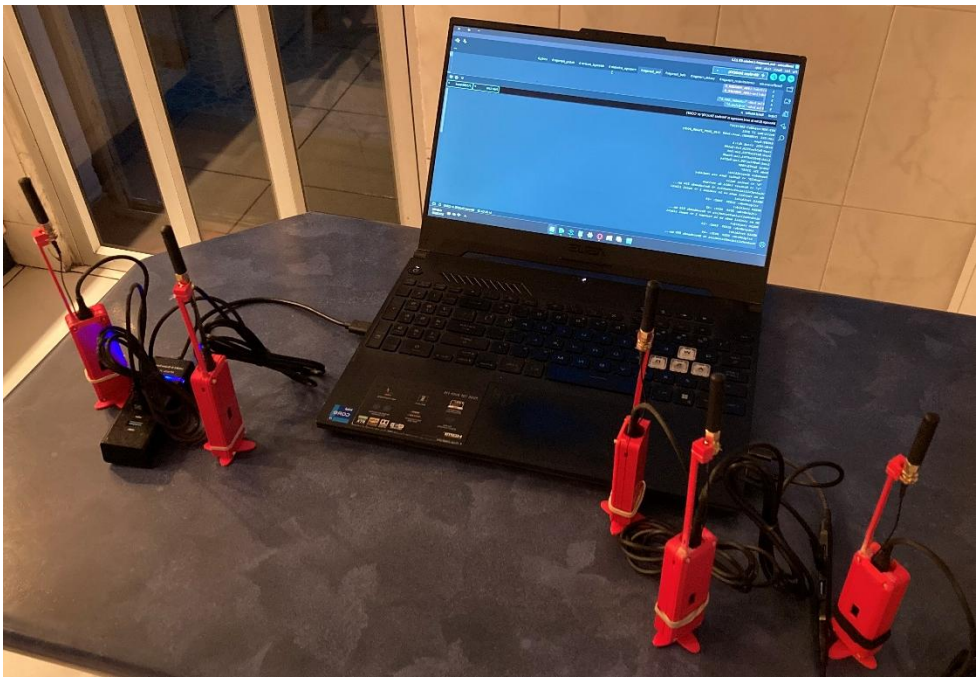

Figura 32 Fotografía del escenario de prueba. Elaboración propia.

Este escenario busca validar los mecanismos base del sistema, medir el PDR y evaluar el número de retransmisiones asociadas con los ACKs durante la transmisión de múltiples paquetes en la red.

5.3.2. Escenario 2: Comunicación bidireccional con fallo de nodo intermedio

El enfoque del segundo escenario planteado está orientado a la comunicación cuando un nodo intermedio falla. Aquí se simula la falla del nodo mediante un apagado intencional del microcontrolador para forzar el uso de la ruta alternativa disponible. Igualmente, se establece una comunicación bidireccional entre los dos extremos de la red, involucrando transmisiones de tres saltos.

En cuanto al flujo de paquetes, este sigue siendo bidireccional, de manera similar al Escenario 1, y comparte la misma implementación física. En la Figura 33, se visualiza como la comunicación ocurre desde el nodo A hacia el nodo E y desde el nodo E al nodo A, pero el nodo D es el que presenta una desconexión.

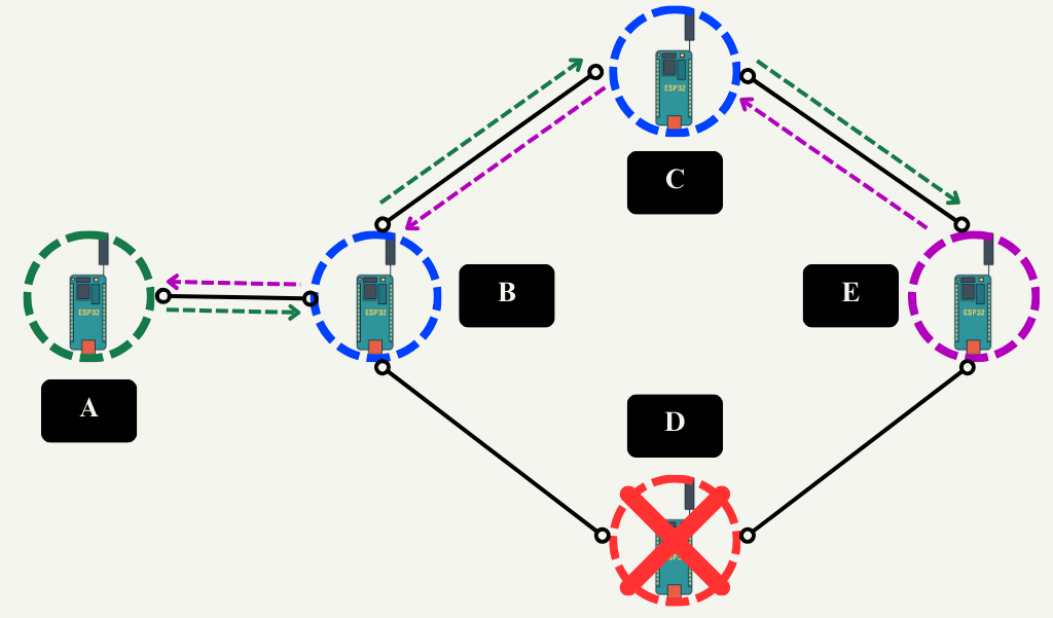

Figura 33 Flujo de paquetes Escenario 2. Elaboración propia.

Este escenario busca evaluar la capacidad del sistema de comunicación para recuperarse ante la falla de un nodo, medir el PRR y el PDR, evaluar el número de retransmisiones asociadas con los ACKs durante la transmisión y analizar el impacto de la falla de un nodo intermedio en la comunicación.

5.3.3. Escenario 3: Comunicación bidireccional sin fallos en entorno físico extendido

El tercer escenario planteado tiene como objetivo evaluar el desempeño del sistema de comunicación bajo una condición de mayor separación física entre los nodos. La topología de red utilizada es la misma que en los escenarios anteriores, pero distribuida en un entorno más amplio, como se observa en la Figura 34. Cabe mencionar que las distancias representadas en el gráfico son aproximadas.

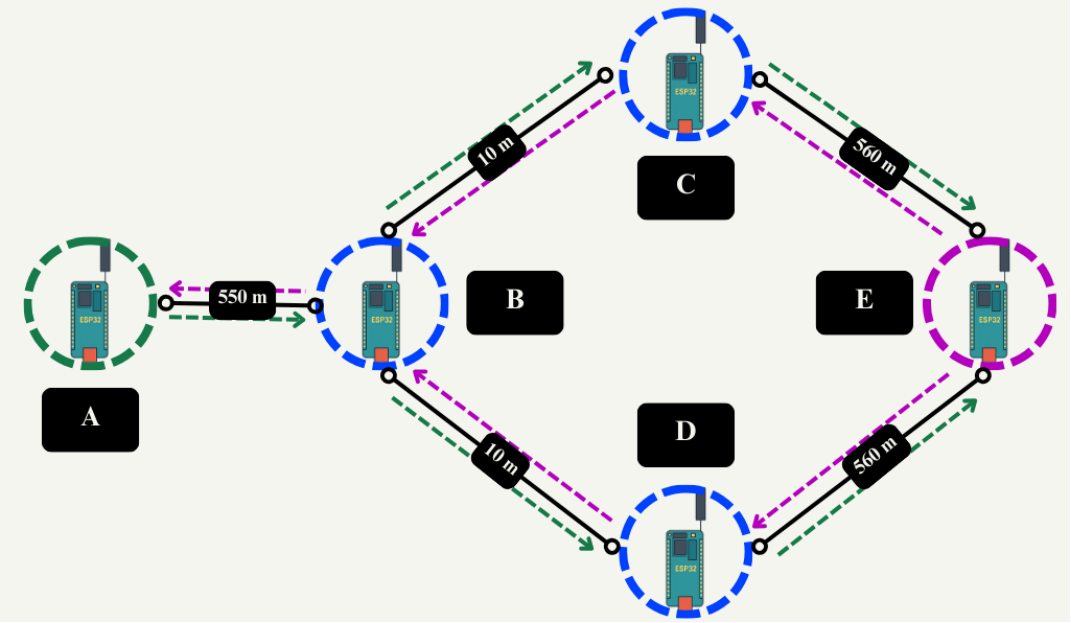

Figura 34 Flujo de paquetes Escenario 3. Elaboración propia.

En la Figura 35 se muestra la distribución de los nodos en el entorno físico real como referencia visual. Cabe recalcar que las distancias no son exactas y que, aunque algunos nodos físicamente están cercanos, el sistema restringe por software los vecinos detectables, por lo que no todos los nodos pueden comunicarse directamente entre ellos. Esto permite forzar la comunicación en la misma topología planteada.

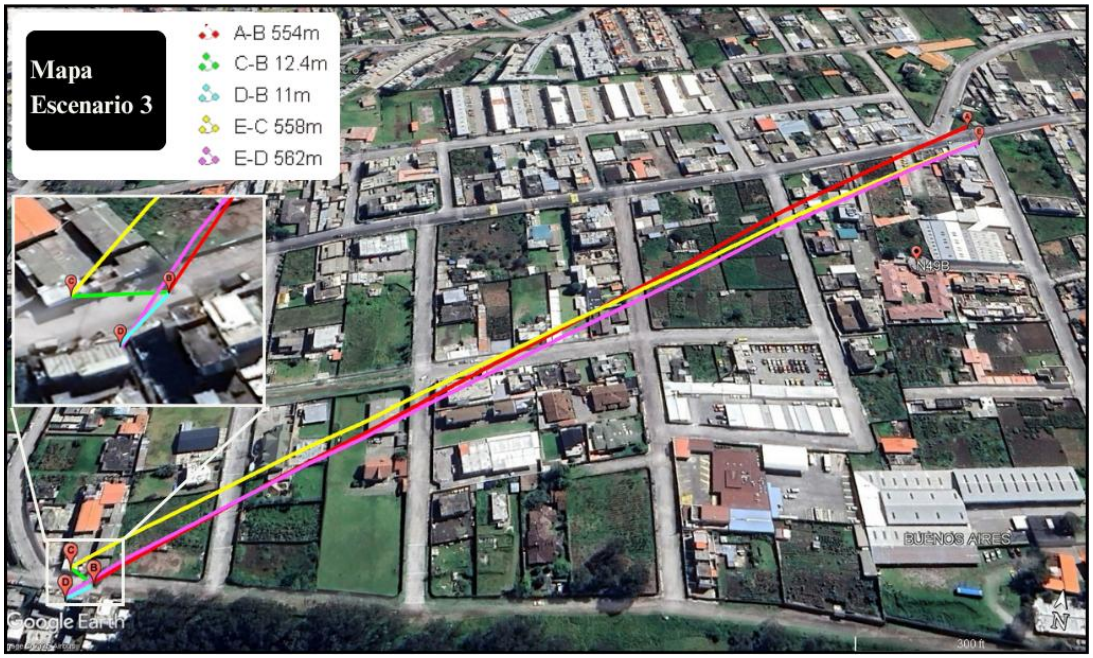

Figura 35 Distribución física referencial de nodos. Elaboración propia.

Este escenario reproduce las condiciones del Escenario 1, manteniendo una comunicación bidireccional entre los nodos extremos de la red a través de tres saltos, pero bajo un entorno físico realista con distancias aumentadas. El flujo de paquetes también ocurre en ambas direcciones simultáneamente, desde el nodo A hacia el nodo E y viceversa.

Al igual que en el primer escenario, se busca validar los mecanismos base del sistema y evaluar métricas como el PDR, el número de retransmisiones asociadas con los ACKs y el comportamiento general de la red. No obstante, este escenario incorpora condiciones espaciales más exigentes, donde no se tiene control sobre factores externos como posibles interferencias electromagnéticas propias de un entorno urbano denso y la acumulación de obstáculos estructurales urbanos, lo cual permite observar el rendimiento del sistema en condiciones operativas más cercanas a un despliegue real.

5.4. Resultados obtenidos

Los resultados obtenidos se documentaron según los tres escenarios definidos. En cada caso se registraron los datos por repetición, incluyendo el estado de recepción de los paquetes y el número de retransmisiones requeridas para completar la entrega. Únicamente en el escenario que involucró la falla de un nodo intermedio se incluyó la documentación relacionada con la recuperación de paquetes tras la reconvergencia.

A partir de los datos registrados se calcularon las métricas de PDR, PRR y número total de retransmisiones. Mientras que la PDR y las retransmisiones fueron evaluadas en los tres escenarios, el PRR fue calculado exclusivamente para el escenario con fallo, al ser la única situación en la que se presentó un proceso de reconvergencia. Además, a partir de los datos se calcularon los promedios de las métricas por escenario.

5.4.1. Escenario 1: Comunicación bidireccional sin fallos

Las pruebas del Escenario 1 consistieron en establecer una comunicación bidireccional entre los nodos extremos de la red *mesh* (A y E), bajo condiciones ideales y sin fallos. Las pruebas se realizaron en un entorno de pruebas físico limitado y restringiendo por software a los vecinos detectables de cada nodo, para así evitar una modificación inesperada de la topología o la formación de una *mesh* completa.

Se realizaron cinco repeticiones del proceso de prueba, enviando dos paquetes por cada sentido de la comunicación. Durante la ejecución se registraron datos individuales por paquete como el identificador de paquete, estado de recepción y el número total de

retransmisiones requeridas para la entrega *end-to-end.* También se registró la dirección de envío, con el fin de llevar un control de los paquetes que fluyen desde cada origen.

Los resultados registrados corresponden a condiciones controladas, planificadas para evaluar la estabilidad y la comunicación sin variables externas inesperadas. Este conjunto de registros permitió documentar de forma ordenada el comportamiento del sistema, sirviendo como base para los cálculos posteriores. Estos resultados de las pruebas sobre este escenario están plasmados en la Tabla 2.

Tabla 2 *Resultados de las transmisiones en Escenario 1.*

Repetición	Dirección	ID Paquete	Enviado	Recibido	Retransmisiones
1	A → E	25318573	X	X	0
		25318503	X	X	3
	E → A	22175720	X	X	2
		22175552	X	X	1
2	A → E	25318568	X	X	3
		25318552	X	X	0
	E → A	22175669	X	X	4
		22175502	X	X	1
3	A → E	25318438	X	X	4
		25318475	X	-	3
	E → A	22175524	X	X	0
		22175619	X	X	3
4	A → E	25318524	X	X	0
		25318427	X	X	2
	E → A	22175556	X	X	5
		22175614	X	X	3
5	A → E	25318628	X	X	1
		25318437	X	X	3
	E → A	22175658	X	X	0
		22175653	X	X	4

(Nota: "X" indica acción realizada, "-" indica paquete perdido)

También, en función de los resultados obtenidos, se calcularon el PDR y el total de retransmisiones realizadas durante toda la comunicación, ambas métricas registradas por cada intento. Para ello, se determinó el valor de PDR individual por repetición y se contabilizó el número total de retransmisiones requeridas en cada una. Finalmente, se calcularon los valores promedio de PDR y retransmisiones correspondientes a las cinco repeticiones del experimento, como se muestra en la Tabla 3.

Tabla 3 *Promedio de PDR y retransmisiones en Escenario 1.*

Repetición	PDR (%)	Retransmisiones
1	100	6
2	100	8
3	75	10
4	100	10
5	100	8
Promedio	95.00	8.00

Como resultado de las pruebas realizadas en el Escenario 1, se obtuvo PDR promedio del 95% y un promedio de ocho retransmisiones por repetición para lograr la entrega completa de los paquetes. Sólo en una repetición hubo perdidas llegando a tener un PDR del 75%, mientras que el resto de las repeticiones registran un PDR del 100%, es decir todos los paquetes enviados se recibieron exitosamente.

5.4.2. Escenario 2: Comunicación bidireccional con fallo de nodo intermedio

Las pruebas del Escenario 2 consistieron en establecer una comunicación bidireccional entre los nodos extremos de la red *mesh* (A y E), simulando la falla de un nodo intermedio. Para ello, se desconectó intencionalmente uno de los nodos encargados del enrutamiento, forzando así la selección de un nodo alterno como ruta alternativa tras fallo.

Esta situación permitió enfocarse en la evaluación del sistema frente a eventos inesperados como lo es la caída de una ruta activa, simulando condiciones que podrían presentarse en aplicaciones reales. Se realizaron cinco repeticiones del proceso de prueba, enviando dos paquetes por cada sentido de la comunicación en cada intento.

Durante la ejecución se registraron datos individuales por paquete como el identificador de paquete, estado de recepción, si fue recuperado tras reconvergencia, y el número total de retransmisiones requeridas para la entrega *end-to-end*. Es importante mencionar que sólo se consideraron paquetes que fueron recuperados tras la falla del enlace que tomaron, se estableció como no aplicables a aquellos paquetes que desde un inicio fueron enrutados por la ruta que no tenía una falla planificada. Estos resultados se presentan en la Tabla 4.

Tabla 4 *Resultados de las transmisiones en Escenario 2.*

Repetición	Dirección	ID Paquete	Enviado	Recibido	Recuperado tras reconvergencia	Retransmisiones
1	A → E	25318575	X	X	No aplica	1
		25318589	X	X	No aplica	0
	E → A	22175724	X	X	X	4
		22175572	X	X	X	7
2	A → E	25318531	X	X	X	4
		25318647	X	X	X	7
	E → A	22175529	X	X	X	4
		22175738	X	-	-	3
3	A → E	25318596	X	X	X	4
		25318577	X	X	X	6
	E → A	22175611	X	X	No aplica	0
		22175677	X	X	X	7
4	A → E	25318412	X	-	-	0
		25318525	X	X	X	7
	E → A	22175512	X	X	No aplica	1
		22175617	X	X	X	8
5	A → E	25318580	X	X	No aplica	0
		25318404	X	X	No aplica	2
	E → A	22175546	X	X	X	4
		22175695	X	X	X	4

(Nota: "X" indica acción realizada; "-" indica paquete perdido. "No aplica" en la columna "Recuperado tras reconvergencia" indica que el paquete fue correctamente enrutado por la ruta alternativa desde el inicio.)

También, en función de los resultados obtenidos, se calcularon el PDR, el total de retransmisiones realizadas durante toda la comunicación, y el PRR. Estas métricas fueron registradas por cada intento. Finalmente, se calcularon los valores promedio correspondientes a las cinco repeticiones del experimento, como se muestra en la Tabla 5.

Tabla 5 *Promedio de PDR, retransmisiones y PRR en Escenario 2.*

Repetición	PDR (%)	Retransmisiones	PRR (%)
1	100	12	100
2	75	18	75
3	100	17	100
4	75	16	66.67
5	100	10	100
Promedio	90.00	15.00	88.33

Como resultados del Escenario 2, se obtuvo un PDR promedio del 90%, con un promedio de quince retransmisiones para la entrega *end-to-end* por repetición y un PRR promedio del 88.33%, tomando en cuenta sólo aquellos paquetes que fueron recuperados tras la falla, no aquellos recibidos pero que se enrutaron desde un inicio al nodo alterno.

5.4.3. Escenario 3: Comunicación bidireccional sin fallos en entorno físico extendido

Las pruebas del Escenario 3 fueron muy similares a la comunicación bidireccional del Escenario 1 al consistir en la comunicación entre los nodos extremos A y E pero esta vez fue realizada en un entorno físico con mayor separación entre nodos. Igualmente, se realizaron cinco repeticiones del experimento, enviando dos paquetes por cada sentido de la comunicación en cada intento.

Durante la ejecución se recopilaron datos individuales por paquete, tales como el identificador, estado de recepción y la cantidad de retransmisiones requeridas para completar la entrega. Estos resultados están registrados en la Tabla 6.

Tabla 6 *Resultados de las transmisiones en Escenario 3.*

Repetición	Dirección	ID de Paquete	Enviado	Recibido	Retransmisiones
1	A → E	25318637	X	X	3
		25318491	X	X	2
	E → A	22175722	X	X	2
		22175497	X	X	1
2	A → E	25318475	X	X	3
		25318422	X	-	0
	E → A	22175640	X	-	3
		22175586	X	X	7
3	A → E	25318519	X	X	3
		25318604	X	X	8
	E → A	22175540	X	X	2
		22175440	X	-	3
4	A → E	25318633	X	X	4
		25318424	X	X	2
	E → A	22175654	X	X	2
		22175578	X	X	4
5	A → E	25318525	X	-	6
		25318580	X	X	2
	E → A	22175640	X	-	3
		22175691	X	X	0

(Nota: "X" indica acción realizada; "-" indica paquete perdido.)

De igual manera que los escenarios anteriores se calcularon las métricas de PDR y número total de retransmisiones durante toda la comunicación. Finalmente, se calculó el promedio con las métricas registradas en cada repetición como se puede visualizar en la Tabla 7.

***Tabla** 7 Promedio de PDR y retransmisiones en Escenario 3.*

Repetición	PDR (%)	Retransmisiones
1	100	8
2	50	13
3	75	16
4	100	12
5	50	11
Promedio	75.00	12.00

Como resultado de las pruebas realizadas en el Escenario 3, se obtuvo un PDR promedio del 75% y un promedio de doce retransmisiones por repetición para lograr la entrega completa de los paquetes.

5.5. Análisis de resultados

Tras la ejecución de las pruebas en cada uno de los escenarios planteados, se recopilaron conjuntos de datos que, analizados en conjunto, permiten identificar el comportamiento general del sistema frente a diversas condiciones. Estas condiciones incluyeron entornos controlados, contextos urbanos y situaciones de falla intencionada, lo que permitió observar cómo varían métricas como el PDR, el número total de retransmisiones y el PRR.

El análisis de estas métricas proporciona una visión integral sobre aspectos clave como la fiabilidad del sistema, su capacidad de auto reparación y su efectividad de operación. Cabe destacar que se agruparon las métricas en función de su propósito, con el fin de identificar el desempeño y comportamiento base del sistema en cuanto a transmisión y recepción efectiva, reconvergencia tras fallos y la función de las retransmisiones en la continuidad de la comunicación en la red *mesh*.

5.5.1. Análisis de Tasa de entrega de paquetes (PDR)

En primer lugar, al evaluar el PDR en cada uno de los escenarios, se puede observar un comportamiento bastante estable a pesar de las variaciones de condiciones en los distintos entornos de prueba. Esto incluye tanto la presencia de fallas en nodos intermedios como el

efecto de factores externos no controlables, como la densidad urbanística y las interferencias electromagnéticas propias del entorno urbano del tercer escenario.

En la Figura 36 se visualiza con claridad que el escenario con menor pérdida fue el Escenario 1, lo cual es esperable ya que la topología se mantuvo intacta y las pruebas se realizaron en un entorno controlado. Este escenario alcanzó un PDR promedio del 95 %, lo cual refleja una alta efectividad en condiciones ideales.

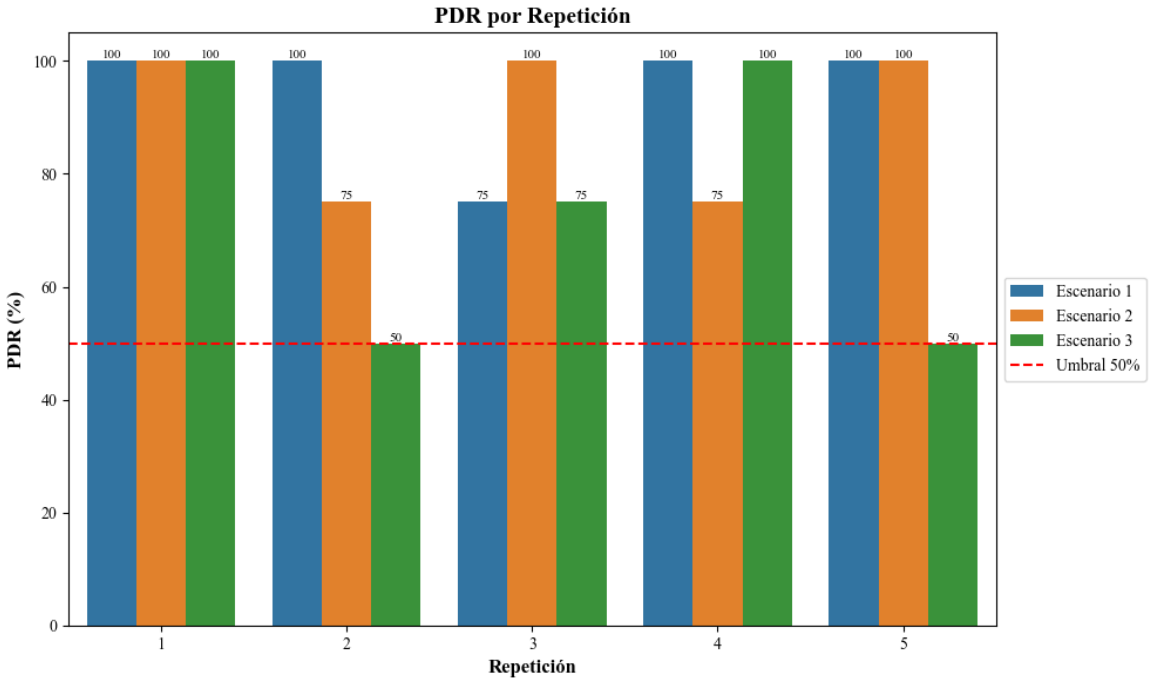

Figura 36 Comparativa del PDR por Escenarios. Elaboración propia.

En cuanto al Escenario 2, a pesar de haber sufrido de una falla intencionada en un nodo intermedio, se mantuvo un comportamiento bastante estable respecto a la recepción de paquetes y muy cercano al comportamiento del primer escenario, logrando un PDR promedio del 90%. Esto demuestra que se mantuvo la efectividad de la comunicación apoyándose del mecanismo de enrutamiento y el mecanismo de reconvergencia, permitiendo la auto reparación de la red tras una falla.

Por otro lado, en el Escenario 3, donde se involucraba un entorno con mayor separación física entre nodos y varios factores externos fuera de control experimental, se experimentó la mayor pérdida de paquetes de los tres escenarios. En este caso la entrega exitosa de la totalidad de los paquetes sólo se logró en dos de las cinco repeticiones. Dichas pérdidas se pueden relacionar con lo mencionado en el marco teórico y conceptual, donde

se establece que una de las grandes limitaciones de la tecnología LoRa es justamente su susceptibilidad y pérdidas de señal en entornos urbanos con estructuras gruesas, estructuras metálicas e interferencia generada por los distintos dispositivos varios en las ciudades.

Es oportuno destacar que a pesar de los factores externos que propiciaron la pérdida de paquetes en ninguna repetición se experimentó una pérdida total de paquetes y el PDR nunca descendió del 50%. El PDR promedio en este escenario fue del 75%, lo cual puede considerarse aceptable dadas las condiciones operativas más variables y exigentes. Este resultado guarda cierta similitud con el estudio de Rademacher et al. (2021), en el que a pesar de llevarse a cabo en un entorno urbano con nodos móviles y a distancias mayores, se reportó que alrededor del 72% de los paquetes fueron recibidos al menos por dos *gateways*.

En dicho estudio el porcentaje fue considerado positivo al tener en cuenta los efectos negativos de las transmisiones en entornos urbanos, como la difracción, las reflexiones múltiples y la pérdida causada por edificios y estructuras urbanas. En este sentido, los resultados registrados en el Escenario 3 al tratarse de pruebas en un entorno urbano, era esperable una mayor dificultad de la propagación de la señal y una reducción del PDR. Aun así, el promedio de 75% del PDR demuestra un desempeño aceptable y que confirma la efectividad del prototipo de sistema de comunicación en condiciones adversas.

5.5.2. Análisis Número de retransmisiones totales

En cuanto al análisis de las retransmisiones totales, se observa un comportamiento más variable entre los distintos escenarios de pruebas. Especialmente el Escenario 1 destaca entre los demás, presentando la menor cantidad de retransmisiones, con un promedio de apenas ocho retransmisiones necesarias para entregar cuatro paquetes en un entorno de comunicación bidireccional. Este valor es aceptable y refleja un funcionamiento adecuado del sistema.

Cabe recalcar que el sistema está diseñado para ejecutar retransmisiones automáticas en caso de no recibir un ACK en un tiempo determinado, lo que podría ocurrir por pérdida de paquetes o demora en la entrega, por lo que la presencia de retransmisiones es normal y esperable. Estos reintentos aseguran la fiabilidad de la comunicación y la integridad de los datos transmitidos.

Como se observa en la Figura 37, el Escenario 2 muestra un alto número de retransmisiones, con una repetición de hasta dieciocho retransmisiones. Aunque este

escenario presenta un promedio de quince retransmisiones y este número podría parecer un indicativo de congestión o sobrecarga, en realidad responde al comportamiento esperado del sistema. Muchos de estos reintentos corresponden a intentos de comunicación con el nodo caído. Mientras se agotan los reintentos hacia el nodo fallido, el resto de la red se mantiene libre, procesando únicamente paquetes de control como los mensajes HELLO, esto ocurre ya que los intentos ocurren a nivel de vecinos, no *end-to-end*, esto asegurando que sólo un enlace tenga el flujo de retransmisiones y no toda la red.

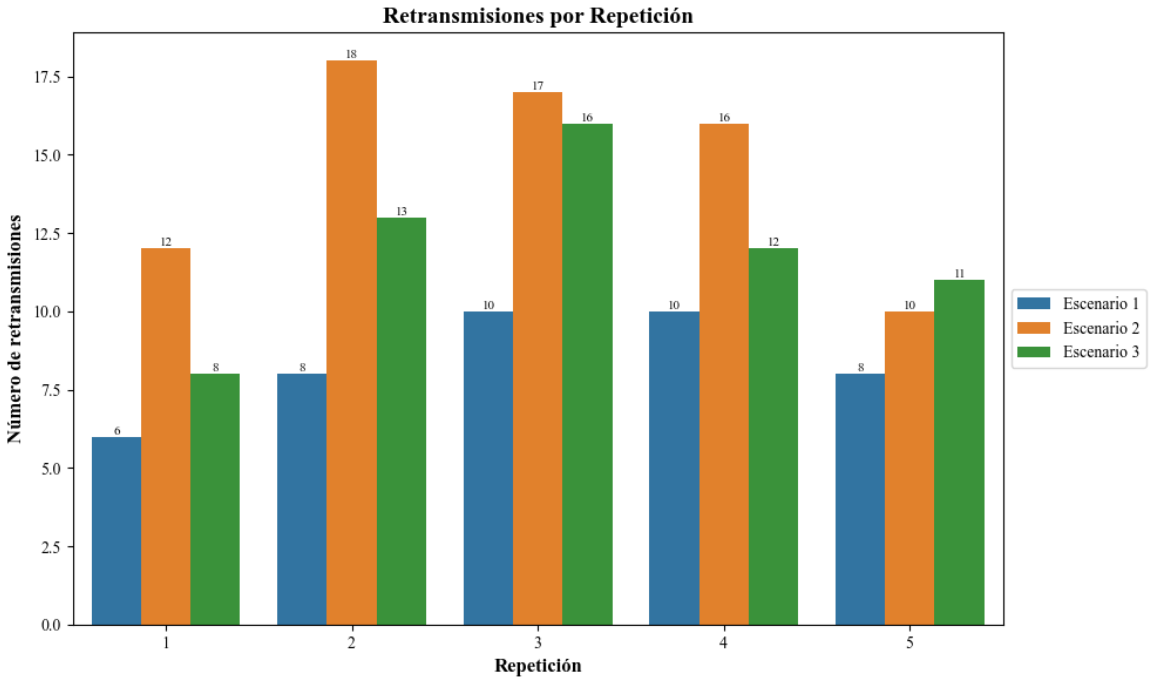

Figura 37 Comparativa de Retransmisiones por Escenarios. Elaboración propia.

En el Escenario 3, también se observa un número considerable de retransmisiones, teniendo un promedio de doce retransmisiones, aunque con una causa distinta. Aquí, los reintentos no se deben a la falla de un nodo, sino a las pérdidas de paquetes influenciadas por factores externos del entorno urbano como ruido, interferencia electromagnética y obstáculos físicos. Estas condiciones dificultan la propagación de la señal, y el sistema responde con retransmisiones automáticas en su esfuerzo por asegurar la entrega de los paquetes. Esto se reafirma en el estudio de Guo et al. (2021) donde se establece que en ciudades la señal LoRa se atenúa por los obstáculos, además se menciona que mayor ruido o interferencia se asocia con mayor cantidad de errores y retransmisiones.

Es importante destacar que un número de retransmisiones elevado por sí sólo no implica un mal funcionamiento del sistema. Por el contrario, es aceptable ya que demuestra

que el sistema está orientado a asegurar la entrega efectiva de datos. Sólo sería motivo de preocupación si este número elevado estuviera acompañado de un PDR muy bajo, lo cual no ocurre en ninguno de los escenarios evaluados.

5.5.3. Análisis Tasa de recuperación ante fallo (PRR)

El análisis del PRR relacionado con fallos en la topología, se centró exclusivamente en el Escenario 2, ya que este fue el único escenario que se planificó con una falla intencionada en un nodo intermedio, de tal forma que se fuerce la selección de una ruta alternativa. Por esta razón este es el único caso en que pudo evaluarse la capacidad del sistema para reconverger y encontrar una ruta cuando la seleccionada se haya vuelto inválida.

Durante esta prueba, se enviaron un total de veinte paquetes tomando en cuenta las cinco repeticiones, sin embargo, sólo catorce de ellos pudieron ser evaluados en términos de recuperación ya que los otros seis fueron enrutados desde un inicio por la ruta alternativa que no dependía del nodo fallido, por lo que no requirieron de la activación del mecanismo de reconvergencia.

Dicho comportamiento, aunque no fue el objetivo central del análisis permitió validar de forma indirecta la efectividad del mecanismo de enrutamiento basado en vecinos. Dicho mecanismo selecciona varios candidatos posibles para el siguiente salto y elige aleatoriamente entre ellos, lo que ayuda a evitar la sobrecarga en un único nodo.

Por otro lado, el PRR promedio alcanzado en la prueba fue de 88,33%, lo cual demuestra un comportamiento estable del sistema de comunicación ante fallos o cambios en la topología y demuestra la capacidad de este para auto repararse mediante la reelección de rutas alternativas para los paquetes. Este porcentaje de recuperación se puede considerar aceptable en redes IoT de propósito general, como aquellas orientadas a la recolección de datos, donde se toleran ciertas pérdidas sin comprometer la funcionalidad del sistema.

Como se muestra en la Figura 38, en tres de las cinco repeticiones se logró una recuperación total, alcanzando un PRR del 100%. El valor más bajo se registró en la cuarta repetición, con un PRR del 66,67%. Este valor se debe a que uno de los paquetes se enrutó desde el inicio por la ruta disponible, lo cual lo descartó del cálculo de la recuperación y otro paquete no logró recuperarse tras la falla, pese a la reconvergencia y los intentos de retransmisión asociados al mecanismo. Además, en ningún caso el PRR descendió por

debajo del 65%, lo que indica que incluso en situaciones adversas, el sistema mantiene un umbral mínimo de recuperación de paquetes.

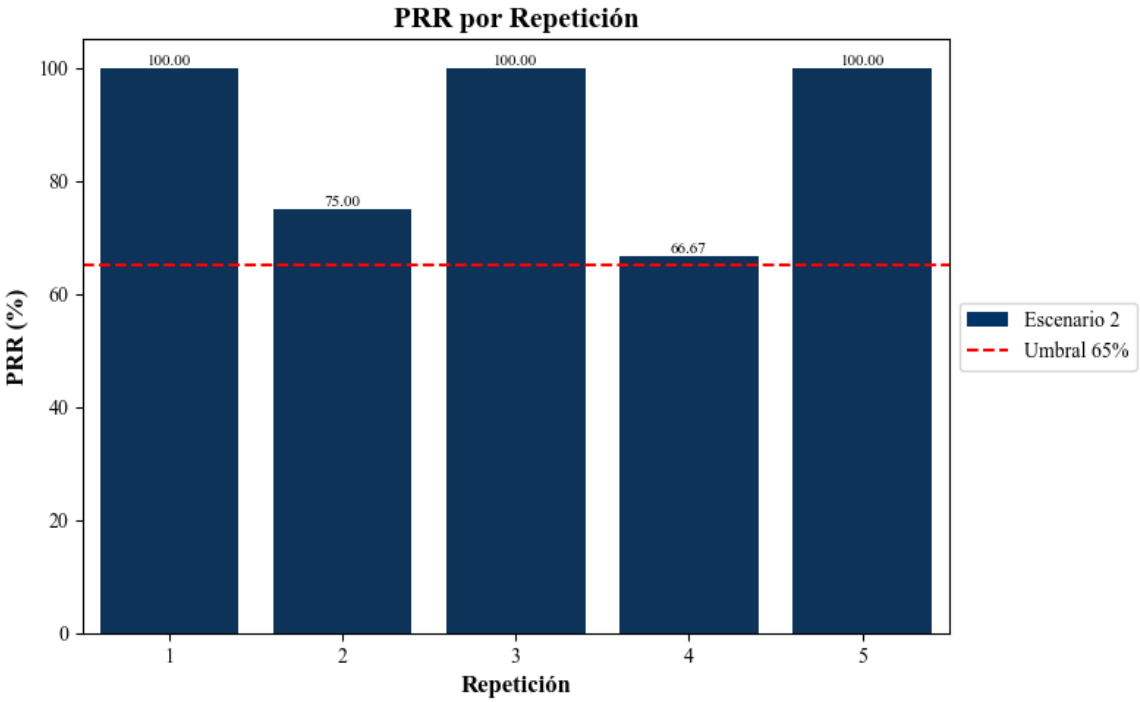

Figura 38 Resultados del PRR en Escenario. Elaboración propia.

Estos resultados demuestran un correcto funcionamiento del proceso de reconvergencia, donde si un nodo emisor no recibe ACKs después de varios reintentos, se inicia la búsqueda de un nuevo vecino disponible para continuar la transmisión. Es importante mencionar que toda reconvergencia está necesariamente acompañada de un número considerable de reintentos, ya que el mecanismo depende de agotar estos intentos antes de seleccionar una nueva ruta.

En general, esta prueba sobre el Escenario 2 permite demostrar que el sistema de comunicación cuenta con una estrategia de tolerancia a fallos, orientada a garantizar a la entrega incluso ante la presencia de fallas físicas o cambios repentinos en la topología de la red *mesh.*

CAPÍTULO VI: Conclusiones y Recomendaciones

Este capítulo se centra en resumir los principales resultados obtenidos a lo largo del desarrollo del prototipo de sistema de comunicación multi-nodo y los hallazgos generales. Se abordan las conclusiones que compilan las contribuciones más significativas derivadas del diseño y desarrollo del prototipo de sistema, basado en microcontroladores ESP32 y módulos LoRa para el despliegue de una red *mesh* descentralizada. Se destacan tanto hallazgos teóricos como prácticos, abordando la pertinencia del enfoque escogido, validando el diseño de los mecanismos de comunicación, su implementación y su viabilidad.

Adicionalmente, se concluye el análisis del comportamiento del sistema durante la fase de pruebas experimentales, evaluando su efectividad y adaptabilidad en distintas condiciones. Las observaciones destacadas en este proceso proveen una base sólida para evaluar el desempeño del sistema.

También se abordan recomendaciones y sugerencias para expandir y continuar con el desarrollo del sistema de comunicación, mejorando sus capacidades. Se incluyen consideraciones de escalabilidad, operabilidad, integración de nuevas funcionalidades y configuración óptima de parámetros basada en pruebas. Estas recomendaciones están dirigidas tanto a futuras investigaciones como a implementaciones prácticas, partiendo del diseño modular y flexible que caracteriza al prototipo.

A lo largo de este capítulo se busca consolidar los hallazgos obtenidos durante todo el proceso de investigación, diseño, desarrollo y pruebas; reafirmar el enfoque que motivó la propuesta; y ofrecer una base sólida para la evolución del prototipo, contribuyendo al ecosistema IoT.

6.1. Conclusiones

A lo largo del presente trabajo de titulación se abordó la fundamentación teórica, el diseño, desarrollo y validación de un prototipo de sistema de comunicación multi-nodo basado en microcontroladores ESP32 con la tecnología LoRa, orientado a conformar redes *mesh* ligeras, descentralizadas y efectivas. El planteamiento y desarrollo del prototipo permitió construir una solución funcional y técnicamente fundamentada, pudiendo ser aplicado en diversos escenarios IoT debido a su enfoque general.

En primer lugar, se logró plantear un marco teórico y conceptual sólido que abordó de forma integral los aspectos clave del sistema propuesto, incluyendo la modulación LoRa, el protocolo LoRaWAN, las características de los microcontroladores ESP32 y la estructura de redes *mesh*, lo cual permitió identificar beneficios y limitaciones de las tecnologías involucradas. Precisamente, las restricciones de LoRaWAN como la dependencia de *gateways*, su escaso soporte para diversas topologías y limitada adaptabilidad ante fallos motivaron la propuesta de un nuevo sistema de comunicación. Además, aplicar principios teóricos sobre el funcionamiento de LoRa en largas distancias y su susceptibilidad al ruido en entornos urbanos fue clave para anticipar comportamientos y adaptar el diseño del sistema para mantener su funcionamiento confiable.

Por otro lado, el diseño cuidadoso de los mecanismos de comunicación y su interacción derivaron en un sistema orientado a la efectividad, centrado en procurar la entrega de paquetes transmitidos y la corrección de fallas de los nodos. Esta meta de confiabilidad fue abordada desde la etapa de diseño, mediante mecanismos de confirmación *hop-by-hop*, reconvergencia tras fallo de ACK y retransmisiones controladas. Así mismo, se procuró siempre que el sistema fuera viable para microcontroladores, los cuales presentan restricciones en procesamiento, memoria, almacenamiento y conectividad. Por ello, se utilizaron mecanismos simples que, al operar en conjunto, permiten una transmisión y recepción correctas sin sobresaturar los nodos, priorizando confirmación *hop-by-hop* y enrutamiento por vecinos, evitando coordinación o sincronización entre nodos, que serían difíciles de manejar para los microcontroladores.

El desarrollo del sistema tuvo una estrecha dependencia del diseño ya que la definición clara de los mismos permitió distribuir la lógica del sistema en módulos independientes, lo que condujo a una implementación organizada, comprensible y fácilmente ampliable. La separación en archivos como *packet_manager.h*,

communication_manager.h, *message_scheduler.h* y *routing_manager.h* no sólo permitió una mejor implementación de la transmisión, recepción y enrutamiento, sino que también sentó las bases para su mantenimiento o extensión futura.

Por otra parte, durante las pruebas y análisis se evidenció que el prototipo de sistema de comunicación tiene un comportamiento general aceptable, alineado con las expectativas teóricas. En ningún escenario evaluado la tasa de recepción de paquetes (PDR) descendió por debajo del 75%, incluso cuando se realizaron pruebas a mayor distancia en un entorno urbano. Esto demuestra que efectivamente el sistema maneja bien la transmisión y recepción de paquetes a pesar de la variación de condiciones del entorno.

También, el análisis de la tasa de recuperación ante fallo (PRR) y del número total de retransmisiones mostró que el sistema procura asegurar la entrega de paquetes mediante rutas alternativas o reintentos controlados. Esto demuestra que el prototipo es efectivo en su propósito de permitir la comunicación y la auto reparación mediante la reconvergencia, manteniendo la conectividad aun así en condiciones varias y siendo desplegado sobre *hardware* básico como lo son los microcontroladores.

Finalmente, tras la ejecución del trabajo, no solo se pudo establecer los aspectos teóricos clave involucrados, sino también diseñar una arquitectura de red *mesh* adaptada para las capacidades de los microcontroladores ESP32, desarrollar un prototipo funcional y validar este mismo en distintos escenarios experimentales con diversas condiciones. A lo largo de este proceso, se demostró que es posible diseñar e implementar un prototipo de sistema de comunicación ligero y descentralizado, sin sacrificar la efectividad para redes *mesh* LoRa en microcontroladores ESP32, superando las limitaciones de las soluciones tradicionales y abriendo camino para futuros desarrollos en cuanto a redes en entornos IoT.

6.2. Recomendaciones

Además de los resultados obtenidos, existen ciertos aspectos que podrían ser considerados para futuras etapas de extensión del sistema y validación de este. Dado que este prototipo fue planteado como un desarrollo base funcional, aún quedan áreas de mejora para fortalecer su comportamiento y efectividad, tales como la priorización de los paquetes de control y mantenimiento de red, la optimización de los parámetros de configuración, la implementación de mecanismos básicos de QoS y el uso del campo experimental presente en los paquetes de datos.

Una de las sugerencias está orientada a la extensión de las pruebas experimentales hacia entornos rurales, donde exista menor presencia de interferencias y obstáculos, lo que permitiría evaluar con mayor claridad el alcance real del sistema y su estabilidad en dichas condiciones. También, sería conveniente complementar la evaluación en zonas urbanas, pero con acceso a ubicaciones elevadas como terrazas o techos que permitan una mejor visibilidad entre nodos y reducir las pérdidas por bloqueos físicos. Es importante mencionar que este tipo de pruebas requieren mayor cantidad de recursos, logística y coordinación con terceros que asistan en la toma de datos y operación de los nodos.

Otra recomendación importante está orientada al incremento de la densidad de nodos dentro de la red *mesh*, esto con el fin de probar el comportamiento del sistema en cuanto a escalabilidad y auto reparación en topologías más complejas. Cabe recalcar que esto implicaría cierta inversión de recursos para obtener el *hardware* necesario para las evaluaciones.

También, se recomienda llevar a cabo pruebas repetidas en diferentes condiciones, enfocadas a evaluar el impacto de ajustes específicos en los parámetros dentro del archivo de configuración, como los tiempos de espera, potencia de transmisión o los límites de reintento. Este proceso permitiría identificar combinaciones óptimas que mantengan la efectividad observada mientras se comienza a evaluar criterios como latencia, rendimiento temporal o la estabilidad en cuanto al tiempo.

Adicionalmente, se recomienda planificar pruebas aplicadas en escenarios prácticos como la recolección de datos ambientales, el monitoreo climático o la agricultura inteligente, aprovechando que el prototipo desarrollado tiene una arquitectura distribuida que no depende de infraestructura centralizada. Esto lo hace especialmente adecuado para sistemas IoT en áreas extensas, donde los nodos pueden operar de forma autónoma incluso ante fallas parciales. Evaluar el comportamiento del sistema en estos contextos permitiría validar su aplicabilidad en entornos reales y generar evidencia para futuras adaptaciones y mejoras.

Finalmente, es importante mencionar que este prototipo está diseñado de forma modular y flexible lo que lo convierte en un punto de partida para futuras mejoras. A medida que se perfeccionen sus mecanismos, podrían integrarse nuevas funcionalidades o interfaces de monitoreo, e incluso adaptarse a casos de uso específicos dentro del ecosistema IoT.

Tomando en cuenta esto, el sistema no debe entenderse como un producto final sino como una propuesta de solución susceptible a mejoras continuas.

Basado en los resultados obtenidos y las capacidades observadas del sistema, se considera factible impulsar un estudio más amplio que permita validar su aplicabilidad en condiciones más reales, e incluso derivar en la elaboración de una publicación de carácter científico. Esta podría aportar al desarrollo de nuevas investigaciones sobre redes *mesh* basadas en LoRa presentando una alternativa al modelo tradicional de LoRaWAN, con un enfoque útil para ecosistemas IoT descentralizados.

BIBLIOGRAFÍA

Adelantado, F., Vilajosana, X., Tuset-Peiro, P., Martinez, B., Melia-Segui, J., & Watteyne, T. (2017). Understanding the Limits of LoRaWAN. *IEEE Communications Magazine*, *55*(9), 34-40. https://doi.org/10.1109/MCOM.2017.1600613

Arregui, D. (2025). *Danar2714/LoRaMeshProject* (1). Danilo Arregui. https://github.com/Danar2714/LoRaMeshProject/tree/main

Atzori, L., Iera, A., & Morabito, G. (2010). The Internet of Things: A survey. *Computer Networks*, *54*(15), 2787-2805. https://doi.org/10.1016/J.COMNET.2010.05.010

Augustin, A., Yi, J., Clausen, T., & Townsley, W. M. (2016). A Study of LoRa: Long Range & Low Power Networks for the Internet of Things. *Sensors 2016, Vol. 16, Page 1466*, *16*(9), 1466. https://doi.org/10.3390/S16091466

AWS. (2025). *¿Qué es LoRaWAN? - AWS IoT Wireless*. https://docs.aws.amazon.com/es_es/iot-wireless/latest/developerguide/what-is-lorawan.html

Ayoub Kamal, M., Alam, M. M., Sajak, A. A. B., & Mohd Su'ud, M. (2023). Requirements, Deployments, and Challenges of LoRa Technology: A Survey. *Computational Intelligence and Neuroscience*, *2023*(1). https://doi.org/10.1155/2023/5183062

Baltuille, P. (2023). *LoRaWAN y su aportación a las tecnologías IIoT | INCIBE-CERT | INCIBE*. https://www.incibe.es/incibe-cert/blog/lorawan-y-su-aportacion-las-tecnologias-iiot

Cisco. (2025). *MPLS QoS: Classifying and Marking EXP Classifying and Marking MPLS EXP*.

Clausen, T., Jacquet, P., Adjih, C., Laouiti, A., Minet, P., Mühlethaler, P., Qayyum, A., & Viennot, L. (2003). *Optimized Link State Routing Protocol (OLSR)*. https://doi.org/10.34894/VQ1DJA

Dahlqvist, F., Patel, M., Rajko, A., & Shulman, J. (2019). *Growing opportunities in the Internet of Things Maturing underlying technologies will make Internet of Things technologies easier to implement and help companies and investors seize new opportunities*.

https://www.mckinsey.com/~/media/McKinsey/Industries/Private%20Equity%20and%20Principal%20Investors/Our%20Insights/Growing%20opportunities%20%20in%20the%20Internet%20of%20Things/Growing-opportunities-in-the-Internet-of-Things-v5.pdf

Doverspike, R., & Wilson, B. (1994). Comparison of capacity efficiency of DCS network restoration routing techniques. *Journal of Network and Systems Management*, *2*(2), 95-123. https://doi.org/10.1007/BF02139308/METRICS

Easy Automation. (2023). *¿Como funciona LoRaWAN? - Easy Automation*. https://easyautomation.cl/lorawan/

Ejmaa, A. M. E., Subramaniam, S., Zukarnain, Z. A., & Hanapi, Z. M. (2016). A scalable neighbor-based routing protocol for mobile ad hoc networks. *International Journal of Distributed Sensor Networks*, *12*(9). https://doi.org/10.1177/1550147716665501

Ephremides, A. (2024). *The Birth and Growth of Wireless Ad Hoc Networks | IEEE Communications Society*. https://www.comsoc.org/about/news/birth-and-growth-wireless-ad-hoc-networks

Espina, J., Falck, T., Panousopoulou, A., Schmitt, L., Mülhens, O., & Yang, G.-Z. (2014). Network Topologies, Communication Protocols, and Standards. *Body Sensor Networks*, 189-236. https://doi.org/10.1007/978-1-4471-6374-9_5

Espressif Systems. (2025a). *ESP Modules | Espressif Systems*. https://www.espressif.com/en/products/modules

Espressif Systems. (2025b). *ESP32 Wi-Fi & Bluetooth SoC | Espressif Systems*. https://www.espressif.com/en/products/socs/esp32

Greening, C. (2023). *ESP32-S3: Which Pins Should I Use? | atomic14*. https://www.atomic14.com/2023/11/21/esp32-s3-pins

Guo, Q., Yang, F., & Wei, J. (2021). Experimental Evaluation of the Packet Reception Performance of LoRa. *Sensors 2021, Vol. 21, Page 1071*, *21*(4), 1071. https://doi.org/10.3390/S21041071

Held, G., Paolucci, M., Sacile, R., Nicastro, F. M., Middleton, B., Lambert, A. J. D., Gupta, S. M., Golden, P., Dedieu, H., Jacobsen, K., Leo, R., Keyes, J., Peltier, T. R., Peltier,

J., Blackley, J. A., & Blumberg, D. F. (2005). *Wireless Mesh Networks*. https://doi.org/10.1201/9781420031263

Heltec. (2023). *Wireless Stick(V3), ESP32S3 + SX1262 LoRa Node, Meshtastic and LoRaWAN Compatible – Heltec Automation*. https://heltec.org/project/wireless-stick-v3/#DocsResource

Heltec Automation. (2019). *GitHub - HelTecAutomation/Heltec_ESP32: Arduino library for Heltec ESP32 (or ESP32+LoRa) based boards*. https://github.com/HelTecAutomation/Heltec_ESP32

Hercog, D., Lerher, T., Truntič, M., & Težak, O. (2023). Design and Implementation of ESP32-Based IoT Devices. *Sensors 2023, Vol. 23, Page 6739*, *23*(15), 6739. https://doi.org/10.3390/S23156739

Huang, Y. (2020). *What is the Technology Behind LoRa Frequency - MOKOSmart*. https://www.mokosmart.com/lora-frequency/

Islam, M., Jamil, H. M. M., Pranto, S. A., Das, R. K., Amin, A., & Khan, A. (2024). Future Industrial Applications: Exploring LPWAN-Driven IoT Protocols. *Sensors*, *24*(8). https://doi.org/10.3390/S24082509

Karp, B., & Kung, H. T. (2000). *GPSR: Greedy Perimeter Stateless Routing for Wireless Networks £*. https://www.eecs.harvard.edu/~htk/publication/2000-mobi-karp-kung.pdf

López-Matencio, P., Vales-Alonso, J., & Alcaraz, J. J. (2020). LBTM: Listen-before-Talk Protocol for Multiclass UHF RFID Networks. *Sensors 2020, Vol. 20, Page 2313*, *20*(8), 2313. https://doi.org/10.3390/S20082313

LoRa Alliance. (2025a). *About LoRa Alliance® - LoRa Alliance®*. https://lora-alliance.org/about-lora-alliance/

LoRa Alliance. (2025b). *LoRaWAN® Specification v1.1*. https://resources.lora-alliance.org/technical-specifications/lorawan-specification-v1-1

Magrin, D., Member, S., Capuzzo, M., Zanella, A., Member, S., & Zorzi, M. (2019). *A Complete LoRaWAN Model for Single-Gateway Scenarios*. https://www.arxiv.org/pdf/1912.01285v1

Montgomery, D. C. (2013). Design and Analysis of Experiments Eighth Edition. En A. Melhorn (Ed.), *Design and Analysis of Experiments Eighth Edition* (8.ª ed., pp. 66-68). Courier Westford. www.wiley.com/go/permissions.

Murillo Oviedo, J. P., Villamar Anzuategui, R. S., & Paredes Cuyo, J. B. (2024). Cost-effective, customizable ESP32-based controller for agricultural industrial process automation. *Centrosur*, *1*(23), 1-30. https://openurl.ebsco.com/contentitem/asn:182811560?sid=ebsco:plink:crawler&id=ebsco:asn:182811560&crl=c

Perez, R. (1998). Noise Representations in Transponders and Multiple Access. *Wireless Communications Design Handbook*, *1*(C), 202-221. https://doi.org/10.1016/S1874-6101(99)80019-2

Perez, S. C., Facchini, H. A., Bisaro, L. A., & Campos, J. (2013). Tuning mechanism for IEEE 802.11e EDCA optimization. *IEEE Latin America Transactions*, *11*(4), 1134-1142. https://doi.org/10.1109/TLA.2013.6601760

Perkins, C., Belding-Royer, E., & Das, S. (2003). *Ad hoc On-Demand Distance Vector (AODV) Routing*. https://www.rfc-editor.org/rfc/pdfrfc/rfc3561.txt.pdf

Pietrzak, P., Perek, P., & Makowski, D. (2024). Latency Evaluation in the Image Acquisition System Based on MTCA.4 Architecture for Plasma Diagnostics. *Journal of Fusion Energy*, *43*(1). https://doi.org/10.1007/S10894-024-00411-0

Qiu, T., Chen, N., Li, K., Qiao, D., & Fu, Z. (2017). Heterogeneous ad hoc networks: Architectures, advances and challenges. *Ad Hoc Networks*, *55*, 143-152. https://doi.org/10.1016/J.ADHOC.2016.11.001

Rademacher, M., Linka, H., Horstmann, T., & Henze, M. (2021). Path Loss in Urban LoRa Networks: A Large-Scale Measurement Study. *IEEE Vehicular Technology Conference*, *2021-September*. https://doi.org/10.1109/VTC2021-Fall52928.2021.9625531

Rahman, A., Olesinski, W., & Gburzynski, P. (2005). Controlled flooding in wireless ad-hoc networks. *2004 International Workshop on Wireless Ad-Hoc Networks*, 73-78. https://doi.org/10.1109/IWWAN.2004.1525544

Samodya, J. (2023). *ESP32 Industrial Controllers for Innovative Integration - NORVI Industrial Arduino*. https://norvi.lk/esp32-based-industrial-controllers-for-innovations/

Schirn, A. (2023). *IEEE 802.11-2020: Collision Avoidance in Wireless Networks - ANSI Blog*. https://blog.ansi.org/ieee-802-11-collision-avoidance-wireless-networks/

Scofield, D., Wang, L., & Zappala, D. (2008). *HxH: A Hop-by-Hop Transport Protocol for Multi-Hop Wireless Networks*. https://zappala.byu.edu/pubs/hxh-wicon-2008.pdf?utm_source=chatgpt.com

Semtech. (2024a). *LoRa® and LoRaWAN®*. https://www.semtech.com/uploads/technology/LoRa/lora-and-lorawan.pdf

Semtech. (2024b). *LoRaWAN® Device Classes*. https://www.semtech.com/uploads/technology/LoRa/lorawan-device-classes.pdf

Shenoy, N., Hamilton, J., Kwasinski, A., & Xiong, K. (2015). An improved IEEE 802.11 CSMA/CA medium access mechanism through the introduction of random short delays. *2015 13th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks, WiOpt 2015*, 331-338. https://doi.org/10.1109/WIOPT.2015.7151090

Sichitiu, M. (2005). *WIRELESS MESH NETWORKS: OPPORTUNITIES AND CHALLENGES*. https://www.researchgate.net/profile/Mihail-Sichitiu/publication/228525550_Wireless_mesh_networks_opportunities_and_challenges/links/0fcfd50f41892962ee000000/Wireless-mesh-networks-opportunities-and-challenges.pdf

Silicon Labs. (2021). *CP210x USB to UART Bridge VCP Drivers - Silicon Labs*. https://www.silabs.com/developer-tools/usb-to-uart-bridge-vcp-drivers?tab=downloads

Slats, L. (2020). *A Brief History of LoRa®: Three Inventors Share Their Personal Story at The Things Conference*. https://blog.semtech.com/a-brief-history-of-lora-three-inventors-share-their-personal-story-at-the-things-conference

Torres, A. P. A., Silva, C. B. Da, & Filho, H. T. (2021). An Experimental Study on the Use of LoRa Technology in Vehicle Communication. *IEEE Access*, *9*, 26633-26640. https://doi.org/10.1109/ACCESS.2021.3057602

Umamaheswari, S., Antony, W. S., & Joe, A. (2021). Detection and Correction of Node Failures in Wireless Sensor Networks. *2021 7th International Conference on Advanced Computing and Communication Systems, ICACCS 2021*, 1479-1483. https://doi.org/10.1109/ICACCS51430.2021.9441846

Waharte, S., Boutaba, R., Iraqi, Y., & Ishibashi, B. (2006). Routing protocols in wireless mesh networks: Challenges and design considerations. *Multimedia Tools and Applications*, *29*(3), 285-303. https://doi.org/10.1007/S11042-006-0012-8/METRICS

Wu, Z., Qiu, K., & Zhang, J. (2020). A Smart Microcontroller Architecture for the Internet of Things. *Sensors 2020, Vol. 20, Page 1821*, *20*(7), 1821. https://doi.org/10.3390/S20071821

Xiao, W., El Rachkidy, N., & Guitton, A. (2024). Improving collision resolution of superposed LoRa signals using a Slot-Free Decoding Scheme. *Ad Hoc Networks*, *157*, 103442. https://doi.org/10.1016/J.ADHOC.2024.103442

Zona Industrial. (2025). *Heltec Wireless Stick V3 - Dispositivo Compacto de Alto Rendimiento en LoRa*. https://www.zonaindustrial.cl/shop/heltec-wireless-stick-v3-lora-57776