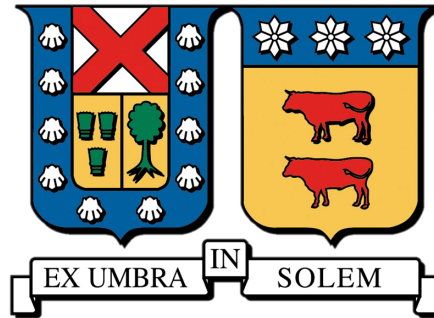


UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAÍSO - CHILE



**DISEÑO DE UNA ARQUITECTURA DE
COMUNICACIÓN INALÁMBRICA EFICIENTE
PARA LA GESTIÓN UNIFICADA DE SENSORES DE
TELEMETRÍA**

CRISTHOFER BRIAN RUBEN ARAYA RECABARREN

**MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO
CIVIL TELEMÁTICO**

PROFESOR GUÍA:

BERIOSKA CONTRERAS VARGAS

PROFESOR CORREFERENTE:

MOHAMED ABDELHAMID

MARZO 2026



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título; ☐ Tesis de Postgrado;

Título del trabajo: Diseño de una arquitectura de comunicación inalámbrica eficiente para la gestión unificada de sensores de telemetría

Nombre del candidato(a): Cristhofer Brian Ruben Araya Recabarren

Carrera / Grado: Ingeniería Civil Telemática

Campus: Casa Central Valparaíso ; **Departamento:** Electrónica

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Berioska Maureen Contreras Vargas, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL

☒ El trabajo **NO contiene información que amerite confidencialidad** y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (embargo) por:

☐ 6 meses; ☐ 12 meses; ☐ 2 años; ☐ 3 años; ☐ 5 años; ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

BERIOSKA
MAUREEN
CONTRERAS
VARGAS

Digitally signed by
BERIOSKA MAUREEN
CONTRERAS VARGAS
Date: 2026.03.04
21:34:17 -03'00'

Fecha: 04/03/2026 ; **Firma:** _____

Estudiante o Candidato(a):

Fecha: 04/03/2026 ; **Firma:** _____

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

Agradecimientos

Agradezco a cada persona que aportó a formar mi persona en este periodo universitario, sin cada uno de ellos siento que no hubiera estado aquí. Agradecer primero a mi familia que siempre me apoyó, a mi tío Víctor quien me dio muy buenos consejos, mis abuelos Eledicto y Marisol que siempre me enseñaron el buen camino y siempre demostraron todo su amor. A mi padre Rubén que fue la persona que se llevó la peor parte de esta etapa pero aun así hizo todo lo posible para darme lo mejor, te quiero viejo. A mi pareja Javiera quien me sostuvo en los peores momentos y la amo con todo mi corazón.

Agradezco a mis amigos del liceo Jano, Fito, Franco, Edu, Jorge y Moldo, en especial a mi mejor amigo Alejandro, quien vivió conmigo muy de cerca toda esta experiencia y me dio muchos consejos.

Por último, pero no menos importante, a mis amigos que hice en la universidad: Martín, Nacho, Julio, Alondra, Felipe, Kuki, Gato, Vega, Enrique y Javier; sin ellos no habría sido lo mismo. Siempre tendré los mejores recuerdos de todos los momentos que pasamos juntos

Resumen

El presente escrito desarrolla e implementa un canal de comunicación inalámbrico bidireccional para la gestión de sensores de bajo consumo de energía. El estudio aborda la ineficiencia de la dependencia de conexiones USB y la necesidad de un software de reprogramación unificado. El objetivo central fue validar la fiabilidad del canal bidireccional, integrando el protocolo Bluetooth de Bajo Consumo Energetico (*BLE*) para la transferencia de datos seriales. El experimento se realizó en un prototipo Arduino Nano 33 BLE utilizando el RTOS Zephyr y el Servicio de Transporte UART Nórdico (*NUS*). Los resultados demuestran la capacidad de transferencia de comandos esenciales para la identificación del firmware y proyectan la viabilidad de la reprogramación total de 64 KB en aproximadamente 35.5 segundos. El análisis confirma que el diseño del puente de transporte *NUS-UART* es un paso crítico hacia la unificación de las herramientas de gestión de sensores.

Keywords: Redes de Sensores Inalámbricos, Sistemas Embebidos, Protocolos de Comunicación.

Abstract

This report documents research into the implementation of a wireless communication channel for telemetry sensor management. The study addresses the inefficiency of reliance on USB connections and the need for unified reprogramming software. The central objective was to validate the reliability of the bidirectional BLE channel for serial data transfer. The experiment was conducted on an Arduino Nano 33 BLE prototype using the Zephyr RTOS and the Nordic UART Transport Service (NUS). The results demonstrate the transfer capability of commands essential for firmware identification and project the feasibility of total reprogramming of 64KB in roughly 35.5 seconds. The analysis confirms that the design of the NUS-UART *bridge* is a critical step toward the unification of sensor management tools.

Keywords: Wireless Sensor Networks, Embedded Systems, Communication Protocols.

Glosario

- **ARM:** Advanced RISC Machines.
- **ATT:** Attribute Protocol.
- **BLE:** Bluetooth Low Energy (Protocolo Bluetooth de Bajo Consumo Energético).
- **FIFO:** First-In, First-Out (Primero en entrar, primero en salir).
- **GATT:** Generic Attribute Profile.
- **GFSK:** Gaussian Frequency Shift Keying.
- **IIoT:** Industrial Internet of Things (Internet Industrial de las Cosas).
- **IoT:** Internet of Things (Internet de las Cosas).
- **ISM:** Industrial, Scientific, and Medical.
- **K-FIFO:** Kernel First-In, First-Out (mecanismo de sincronización de Zephyr).
- **NUS:** Nordic UART Service (Servicio de Transporte UART Nórdico).
- **OTA:** Over-The-Air (Por el aire).
- **RTOS:** Real-Time Operating System (Sistema Operativo en Tiempo Real).
- **UART:** Universal Asynchronous Receiver-Transmitter.
- **UUID:** Universally Unique Identifier.
- **Zephyr OS:** RTOS para dispositivos IoT.

Índice de figuras

2.1. Arquitectura de capas integrada: Flujo de datos desde el Hardware hasta la App mediante Zephyr y NUS.	18
3.1. Diagrama de Componentes e Interacciones del Sistema.	23
3.2. Diagrama de Flujo de Datos entre Componentes	25
3.3. Diagrama de secuencia del ciclo de sincronización obligatoria.	26
4.1. Evidencia de recepción de datos binarios en nRF Connect.	29
4.2. Transcripción del Log de Depuración de Zephyr OS.	30

Índice de cuadros

2.1. Comparativa Técnica de RTOS para el Proyecto	16
3.1. Comparativa de Tasas de Transmisión: UART vs. BLE	27
4.1. Métricas Experimentales de Transmisión (Downlink)	31
5.1. Decodificación de la Trama Binaria Recibida	33
5.2. Proyección de Tiempos de Actualización de Firmware (OTA)	34

Índice general

1. Introducción	4
1.0.1. Planteamiento del problema	5
1.0.2. Objetivo del Proyecto	6
1.0.3. Alcance del Trabajo	6
2. Marco Teórico	8
2.0.1. Tecnología Bluetooth	8
2.0.2. Sistemas Operativos en Tiempo Real (RTOS)	14
2.0.3. Servicio de Transporte UART Nórdico (NUS)	16
2.0.4. Microcontrolador experimental	18
2.0.5. Codificación y Serialización de Datos en Sistemas Embebidos .	20
3. Arquitectura del Canal de Comunicación	23
3.0.1. Arquitectura General y Flujo de Datos	23
3.0.2. Diseño Lógico para la Fiabilidad: Ciclo de Sincronización . . .	25
3.0.3. Optimización y Eficiencia Operativa	26
4. Resultados Experimentales	28
4.0.1. Validación de Transmisión de Estructuras Binarias (Uplink) . .	28
4.0.2. Validación de Recepción y Decodificación de Datos (Downlink)	30
5. Análisis de Resultados	32
5.0.1. Análisis de Integridad de la Serialización Binaria	32

5.0.2.	Análisis de Fiabilidad del Canal Transparente	33
5.0.3.	Proyección de Viabilidad para Reprogramación	33
5.0.4.	Consolidación de Hallazgos Técnicos	34
6.	Conclusion y Trabajo Futuro	35
6.0.1.	Trabajo Futuro	36
	Referencias	37

Capítulo 1

Introducción

Una empresa de telemetría especializada requiere optimizar la gestión de sus sensores desplegados en entornos de agricultura de precisión y vigilancia ambiental. Históricamente, cada nueva generación de sensores ha requerido el desarrollo de un *software* específico que es reprogramado manualmente en cada sensor para mantener su actualización. Este enfoque de desarrollo *ad-hoc* genera una arquitectura técnicamente no sostenible debido a la siguiente problemática que afecta el ciclo de vida del desarrollo de software:

- Ineficiencia por Fragmentación: El mantenimiento de múltiples bases de código independientes incrementa los recursos de ingeniería e impide la reutilización de módulos lógicos probados. Esto obliga al equipo de ingeniería a duplicar esfuerzos de validación para cada variante del dispositivo, donde la corrección de un error en el protocolo de comunicación debe replicarse manualmente en múltiples herramientas distintas, aumentando el tiempo de despliegue y la probabilidad de introducir inconsistencias lógicas entre versiones de productos que deberían comportarse de manera similar.

1.0.1. Planteamiento del problema

El proceso actual de reprogramación de sensores (*burning/flash*) presenta vulnerabilidades técnicas críticas, como, por ejemplo; el bloqueo del sensor o transmisión de datos erróneos.

Fiabilidad del Sistema

Existe una correlación directa entre la fragmentación del software descrita anteriormente y la fiabilidad del sistema. El problema central de fiabilidad radica en que el proceso actual depende totalmente de la decisión manual del asesor técnico para seleccionar la versión de software adecuada entre múltiples opciones incompatibles.

Al no existir un mecanismo de negociación automático que identifique el modelo del sensor al momento de la conexión, el operador debe confiar en su criterio personal. Esta dependencia del factor humano, exacerbada por la diversidad de herramientas fragmentadas, induce a errores operativos frecuentes (Arzo et al., 2021). Las consecuencias técnicas de una selección incorrecta incluyen:

- *Bloqueo del Dispositivo*: El sensor queda en un estado inoperable al cargar instrucciones incompatibles con el modelo físico.
- *Datos Erróneos*: La configuración de parámetros erróneos invalida la calibración de los instrumentos, comprometiendo la veracidad de las mediciones ambientales recolectadas.

Limitaciones Físicas de la Interfaz

Desde la perspectiva de la eficiencia operativa, el sistema está limitado por la exigencia de una conexión física USB. En entornos de despliegue real, esta dependencia técnica presenta desventajas críticas (Ahmed et al., 2006):

- *Vulnerabilidad de Hardware*: La necesidad de exponer puertos físicos para la conexión compromete la integridad física del dispositivo frente a manipulaciones

externas (tampering).

- **Acceso Restringido:** La reprogramación se vuelve técnicamente inviable en sensores ubicados en posiciones de difícil acceso físico, impidiendo actualizaciones críticas de seguridad o corrección de errores.

1.0.2. Objetivo del Proyecto

Demostrar la fiabilidad y eficiencia de un canal de comunicación bidireccional basado en Bluetooth Low Energy (*BLE*) y el protocolo *NUS* para la reprogramación unificada de la red de sensores, validando la transferencia de los comandos esenciales para la identificación.

1.0.3. Alcance del Trabajo

La investigación se delimita al diseño e implementación de la capa de transporte sobre el prototipo Arduino Nano 33 BLE con Zephyr OS. La placa seleccionada tiene un propósito experimental, siendo propicia para este estudio, no así para escenarios productivos. El enfoque principal es la validación del canal de comunicación bidireccional, que es el componente habilitador para la reprogramación unificada. El proyecto demuestra:

1. El establecimiento de un enlace *BLE-NUS* fiable.
2. La funcionalidad del *bridge NUS-UART* para transferir datos de forma transparente.
3. La viabilidad técnica de la carga de datos (proyección de rendimiento) necesaria para la reprogramación completa.

El presente escrito de orientación experimental se organiza en seis capítulos principales. El Capítulo 1, Introducción, describe la problemática de la fragmentación tecnológica y define los objetivos de la solución inalámbrica. El Capítulo 2, Marco Teóri-

co, profundiza en los fundamentos de BLE, la arquitectura GATT y el sistema operativo Zephyr. El Capítulo 3, Arquitectura del Canal de Comunicación, detalla el diseño de la solución, enfocándose en la sincronización de colas y el flujo de datos. Los Capítulos 4 y 5, Resultados y Análisis de Resultados, presentan la evidencia experimental de la transmisión binaria y evalúan el desempeño en términos de eficiencia y fiabilidad. Finalmente, el Capítulo 6 presenta las Conclusiones generales y las líneas de trabajo futuro para la maduración del prototipo.

Capítulo 2

Marco Teórico

2.0.1. Tecnología Bluetooth

El estándar Bluetooth es una tecnología de comunicación inalámbrica diseñada para el intercambio de datos en cortas distancias. Para comprender su importancia en la ingeniería actual, es necesario analizar su evolución normativa entre organismos de estandarización.

Originalmente, la especificación fue adoptada por el Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) bajo el estándar IEEE 802.15.1. Sin embargo, este estándar dejó de mantenerse activamente por dicha organización, pasando la responsabilidad total de su desarrollo y licenciamiento al Bluetooth Special Interest Group (SIG).

En la actualidad, las especificaciones del Bluetooth SIG se consideran un estándar de facto. Un estándar de facto es aquel que, sin tener necesariamente un respaldo legal o normativo de una entidad gubernamental, ha logrado una aceptación y uso tan generalizados en la industria que se convierte en la norma práctica obligatoria (Koulouras et al., 2025).

Selección Tecnológica: Bluetooth BR/EDR vs. BLE

Dentro de esta tecnología, existen dos variantes incompatibles entre sí: Basic Rate/Enhanced Data Rate (BR/EDR), conocido como Bluetooth Clásico, y Low Energy

(BLE).

La diferencia fundamental radica en su enfoque de conexión. El Bluetooth BR/EDR está diseñado para mantener una conexión continua y de alto ancho de banda, ideal para transmisión de audio o transferencia de archivos pesados. Su desventaja es el alto consumo energético y tiempos de conexión lentos (Kajikawa et al., 2016).

Por el contrario, BLE está optimizado para transmisiones esporádicas de pequeñas cantidades de datos. Permite que el dispositivo permanezca en modo de suspensión la mayor parte del tiempo, despertando solo por milisegundos.

Para este proyecto, se seleccionó BLE por sobre BR/EDR. El argumento técnico es que la gestión de un sensor no requiere un flujo de datos constante, sino el intercambio de datos de configuración y tramas de firmware puntuales. BLE permite establecer conexiones mucho más rápidas y maximiza la autonomía de la batería del sensor, alineándose con los requisitos de eficiencia energética de la telemetría IoT.

Gestión de Estados en el Controlador de Enlace

La gestión de la comunicación en BLE es responsabilidad de la Capa de Enlace (Link Layer), la cual reside en el Controlador del dispositivo. En el caso del Arduino Nano 33 BLE utilizado en este proyecto, este controlador funciona con un microchip central llamado nRF52840 (Gholami & Heydarian Caydan, 2023).

El controlador opera bajo una máquina de estados finitos estricta (Wiecha et al., 2016). A continuación, se describen los cinco estados fundamentales y cómo se aplican específicamente en el funcionamiento del prototipo desarrollado:

1. *Standby (Reposo)*: Es el estado por defecto. El controlador no transmite ni recibe paquetes. En el proyecto, el sensor entra en este estado para ahorrar energía cuando no se requiere comunicación activa.
2. *Advertising (Anuncio)*: El dispositivo transmite paquetes de datos en los canales de publicidad (37, 38, 39) para indicar su presencia. En el prototipo, el sensor

asume el rol de anunciante, difundiendo su nombre y el UUID del servicio NUS para ser descubierto por la aplicación móvil.

3. *Scanning (Escaneo)*: El dispositivo escucha los canales de publicidad buscando anunciantes. Este estado es gestionado por la aplicación móvil, que busca activamente el sensor en el entorno.
4. *Initiating (Iniciación)*: Tras descubrir al sensor objetivo, el smartphone entra en este estado para enviar una solicitud de conexión.
5. *Connection (Conexión)*: Una vez establecida la conexión, el sensor pasa a operar como Periférico y el móvil como Central. En este estado, el controlador del chip nRF52840 gestiona el salto de frecuencia y los intervalos de conexión para intercambiar los datos de manera sincronizada.

Capas Físicas y Esquema de Direccionamiento

Una vez definidos los estados operativos, es necesario detallar la estructura física y lógica que permite la identificación unívoca de los dispositivos en la red.

Capa Física (PHY) y Canales: La capa física opera en la banda ISM de 2.4 GHz utilizando modulación GFSK. El espectro se divide en 40 canales de 2 MHz cada uno. Para garantizar la robustez en entornos industriales ruidosos, BLE separa estos canales en dos funciones: 3 canales (37, 38, 39) se dedican exclusivamente a la publicidad (*Advertising*), evitando interferencias con las redes Wi-Fi comunes, mientras que los 37 canales restantes se utilizan para la transmisión de datos mediante un algoritmo de salto de frecuencia adaptativo.

Direccionamiento de Dispositivos (Addressing): Uno de los aspectos más críticos de la capa de enlace es cómo se identifican los dispositivos. A diferencia de un “paquete estándar”, la identidad de un nodo BLE se define mediante una Dirección de Dispositivo de 48 bits.

Según el estándar RFC 7668, que adapta IPv6 sobre redes BLE, y la especificación

Core, existen cuatro tipos de direcciones clasificadas en dos categorías principales: Públicas y Aleatorias (Montenegro et al., 2015).

- *Dirección Pública (Public Device Address)*: Es una dirección fija y globalmente única, similar a una dirección MAC IEEE 802. EUI-48. Se compone de un identificador de compañía (OUI) y un identificador de dispositivo. Se utiliza cuando la identidad del dispositivo no requiere privacidad.
- *Dirección Aleatoria (Random Device Address)*: No requieren registro ante el IEEE y se subdividen para manejar la privacidad y seguridad:
 - *Aleatoria Estática*: Se genera al arrancar el dispositivo y se mantiene fija durante el ciclo de energía. Puede cambiar al reiniciarse, pero no durante una conexión.
 - *Privada Resoluble (Resolvable Private Address - RPA)*: Es fundamental para la privacidad en dispositivos móviles. La dirección cambia periódicamente (rotación), evitando el rastreo por terceros. Sin embargo, un dispositivo de confianza (que posee la clave *Identity Resolving Key* o IRK) puede “resolver” esta dirección aleatoria para identificar al dispositivo real.
 - *Privada No Resoluble*: También cambia periódicamente, pero no puede ser resuelta por nadie. Se utiliza raramente, solo en escenarios donde la reconexión con dispositivos previos no es necesaria.

En el contexto de este proyecto, el prototipo Arduino Nano 33 BLE utiliza por defecto una dirección *Aleatoria Estática* generada por el stack de Zephyr, lo cual simplifica el desarrollo al mantener una identidad estable durante las pruebas de laboratorio sin la complejidad de gestión de claves IRK.

Protocolos Lógicos y Estructura de Datos (GATT/GAP)

Una vez establecidos los mecanismos físicos de transmisión y el esquema de direccionamiento detallados en la sección anterior, es necesario definir cómo se gestiona,

procesa y asegura la información que fluye a través de ese enlace. Aquí entra en juego la capa superior de la arquitectura BLE, conocida como Host.

De acuerdo con la arquitectura descrita por Townsend et al. (2014), esta capa implementa la lógica de alto nivel y abstrae la complejidad del hardware. El Host se comunica con el Controlador inferior a través de una interfaz estandarizada llamada HCI (Host Controller Interface). Esta separación es fundamental, ya que permite que el sistema operativo (Zephyr OS) envíe instrucciones lógicas para controlar el transceptor de radiofrecuencia (el hardware físico) sin necesidad de que el programador manipule directamente las señales eléctricas o la sincronización de la radio.

A continuación, se describen los protocolos fundamentales que operan en esta capa para dar estructura a los datos:

Protocolo de Control y Adaptación de Enlace Lógico (L2CAP): El L2CAP actúa como un multiplexor de protocolos y gestor de fragmentación. Su función principal es tomar los paquetes de datos provenientes de las capas superiores (como ATT o SM) y dividirlos en trozos más pequeños que puedan ser transportados por la Capa de Enlace, la cual tiene un tamaño de carga útil limitado (MTU). En el proceso inverso, reensambla los fragmentos recibidos. Esto permite que el sistema maneje tramas de datos largas, como las requeridas para la transferencia de archivos de firmware, de manera transparente para la aplicación.

Gestor de Seguridad (SM - Security Manager): Este protocolo define los métodos para el emparejamiento (Pairing) y la distribución de claves. El proceso de Pairing permite que dos dispositivos establezcan un canal encriptado intercambiando capacidades de seguridad. Si se desea guardar esta relación para futuras conexiones (Bonding), el SM gestiona el intercambio y almacenamiento de las claves de identidad (IRK) y de cifrado (LTK), protegiendo la integridad de los datos de reprogramación frente a ataques de terceros.

Protocolo de Atributos (ATT): El núcleo del intercambio de datos en BLE reside en el protocolo ATT. Este define la unidad básica de información llamada Atributo. Un atributo en memoria se compone de tres elementos críticos:

- *Handle (Manejador)*: Es una dirección de 16 bits (ej. 0x002A) que identifica de forma única al atributo en la memoria del servidor. El cliente utiliza este handle para leer o escribir el valor.
- *UUID (Identificador Único)*: Define el tipo de información que contiene el atributo.
- *Value (Valor)*: Es el dato real almacenado (ej. un byte de información).

Perfil de Atributos Genérico (GATT): Sobre la base técnica del ATT, el perfil GATT establece una estructura jerárquica para organizar la información que el sensor expone al mundo. Esta estructura es fundamental para entender los resultados obtenidos en este proyecto, ya que organiza los datos en dos niveles:

1. *Servicio (Service)*: Es un contenedor lógico utilizado para agrupar datos relacionados con una función específica. Se puede entender como una “carpeta” en un sistema de archivos. En este proyecto, el servicio principal es el NUS (Nordic UART Service), el cual agrupa todas las funciones de comunicación serial.
2. *Característica (Characteristic)*: Es el punto de dato real dentro de un servicio. Se puede entender como un “archivo” dentro de la carpeta. Una característica contiene el valor del dato (ej. el buffer de recepción) y descriptores que definen cómo se puede interactuar con él (Propiedades). Las propiedades más relevantes para este diseño son:
 - *Write (Escritura)*: Permite al móvil enviar datos al sensor.
 - *Notify (Notificación)*: Permite al sensor enviar datos al móvil asincrónamente, sin esperar una petición.

Perfil de Acceso Genérico (GAP) y su Asociación con el Enlace: Mientras que GATT maneja los datos, el perfil GAP (Generic Access Profile) gestiona la topología de la red y los modos de operación. GAP define los roles que asume el dispositivo y,

fundamentalmente, dicta cómo se comporta el Controlador en la Capa de Enlace (Link Layer).

La relación directa entre los roles GAP y los estados de la Capa de Enlace se detalla a continuación:

- *Rol Periférico (Peripheral Role)*: Es el dispositivo que anuncia su presencia y acepta conexiones. Generalmente es el nodo de recursos limitados (el sensor).
 - *Estado de Enlace asociado*: Inicialmente opera en estado Advertising. Al establecerse la conexión, pasa a ser el Esclavo (Slave) en el estado Connection.
- *Rol Central (Central Role)*: Es el dispositivo que busca a otros e inicia la conexión. Generalmente es un smartphone o gateway.
 - *Estado de Enlace asociado*: Inicialmente opera en estado Scanning. Al encontrar al periférico deseado, pasa al estado Initiating y finalmente se convierte en el Maestro (Master) en el estado Connection.

Esta distinción es vital para el proyecto: el prototipo Arduino Nano 33 BLE está configurado en el stack de Zephyr bajo el rol GAP Periférico, lo que obliga a su controlador de radio a iniciar en modo de Anuncio (Advertising) para ser visible por la aplicación de gestión.

2.0.2. Sistemas Operativos en Tiempo Real (RTOS)

Un Sistema Operativo en Tiempo Real (RTOS) es una plataforma de software diseñada para gestionar los recursos de hardware y la ejecución de aplicaciones con una restricción crítica: el determinismo temporal. En un RTOS, el objetivo principal no es maximizar el rendimiento promedio (como en Windows o Linux), sino garantizar que las tareas críticas se completen dentro de un plazo de tiempo predecible y estricto (Silva et al., 2019).

El componente fundamental que diferencia a un RTOS es el *Planificador* (Scheduler). Este componente implementa algoritmos de multitarea preventiva (Preemptive Multitasking), lo que permite al sistema pausar una tarea de baja prioridad para atender inmediatamente un evento urgente (como una interrupción de radiofrecuencia), garantizando así la sincronización requerida por protocolos de comunicación complejos como BLE.

Panorama de Soluciones y Selección Tecnológica

En el ecosistema actual de sistemas embebidos para IoT, existen varias soluciones dominantes. Para fundamentar la elección tecnológica de este proyecto, se analiza Zephyr OS frente a dos de sus competidores más relevantes identificados en la literatura (Sabri et al., 2017): FreeRTOS y Mbed OS.

1. *FreeRTOS*: Es el RTOS más utilizado en la industria debido a su ligereza. Es un microkernel minimalista que proporciona las funciones básicas de planificación. Sin embargo, carece de un stack de drivers unificado, lo que obliga al desarrollador a integrar bibliotecas de terceros para funciones como Bluetooth o sistemas de archivos.
2. *Mbed OS*: Desarrollado por ARM, es un RTOS diseñado específicamente para microcontroladores de 32 bits. Es el sistema operativo por defecto en el ecosistema Arduino profesional (incluyendo la placa Nano 33 BLE). Ofrece una gran abstracción y facilidad de uso, pero su arquitectura está fuertemente acoplada al ecosistema ARM y puede resultar pesada en consumo de memoria para aplicaciones muy optimizadas.

La ventaja decisiva de Zephyr radica en su gestión del protocolo Bluetooth. Al poseer un stack BLE nativo integrado en el kernel (a diferencia de FreeRTOS) y un sistema de configuración modular que permite desactivar servicios innecesarios para ahorrar memoria (ventaja sobre Mbed OS), Zephyr ofrece el equilibrio óptimo entre

Cuadro 2.1: Comparativa Técnica de RTOS para el Proyecto

<i>Criterio</i>	<i>Zephyr OS</i>	<i>FreeRTOS</i>	<i>Mbed OS</i>
Stack BLE	Nativo, Open Source y Certificado	Requiere librería externa	Integrado (Cordio Stack)
Modelo de Drivers	Desacoplado (Devicetree)	Específico del Vendor	APIs C++ de alto nivel
Configurabilidad	Granular (Kconfig)	Manual (Header files)	Perfiles predefinidos
Licencia	Apache 2.0 (Permisiva)	MIT	Apache 2.0

control de bajo nivel y funcionalidades de red avanzadas necesarias para la reprogramación inalámbrica (The Linux Foundation, 2024).

2.0.3. Servicio de Transporte UART Nórdico (NUS)

Definición y Propósito del Servicio

El perfil de puerto serie (SPP) del Bluetooth clásico no existe en la especificación estándar de BLE. Para suplir esta carencia, Nordic Semiconductor desarrolló el Nordic UART Service (NUS) (Nordic Semiconductor, s. f.). Este es un servicio GATT propietario, identificado por un UUID base de 128 bits específico, diseñado para emular un enlace serial transparente sobre la infraestructura de bajo consumo de BLE.

El propósito del NUS es actuar como un túnel de datos agnóstico. A diferencia de los perfiles estándar que definen el tipo de dato, el NUS no interpreta la información; simplemente transporta flujos de datos sin estructura predefinida desde un punto A a un punto B, lo que lo convierte en la solución ideal para adaptar sistemas heredados basados en UART (Universal Asynchronous Receiver-Transmitter) al mundo inalámbrico sin modificar la lógica de la aplicación existente.

Características de Transmisión y Flujo Transparente

El funcionamiento del NUS se basa en dos características GATT complementarias que permiten la comunicación bidireccional punto a punto:

La característica *NUS RX* permite al dispositivo central enviar datos al periférico. Técnicamente, utiliza la operación *Write Command* (escritura sin respuesta) del protocolo ATT. Esta elección de diseño maximiza el rendimiento (*throughput*), ya que no requiere un acuse de recibo en la capa de aplicación para cada paquete enviado, permitiendo ráfagas de datos continuas.

La característica *NUS TX* permite al periférico enviar datos al central. Utiliza el mecanismo de *Notification* del protocolo ATT. Las notificaciones son mensajes asíncronos que el sensor empuja al dispositivo móvil en cuanto hay datos disponibles en su buffer serial, sin que el móvil tenga que solicitarlos activamente (*polling*).

Este diseño logra un flujo transparente: los bytes que entran por el puerto físico UART del sensor son encapsulados instantáneamente en notificaciones BLE, y los paquetes BLE recibidos son desempaquetados y volcados al puerto UART. Para el procesador del sensor, la interfaz inalámbrica es indistinguible de una conexión cableada.

Arquitectura Integrada del Sistema

La implementación del NUS en este proyecto no es un componente aislado, sino que funciona como una capa intermedia crítica que conecta el mundo físico del hardware con el mundo lógico de la aplicación móvil.

Para comprender cómo se logra el flujo transparente de datos, es necesario visualizar la pila completa de tecnologías involucradas. La Figura 2.1 ilustra la arquitectura integrada, mostrando el recorrido de los datos desde que son generados por el hardware del microcontrolador hasta que son recibidos por la aplicación de usuario.

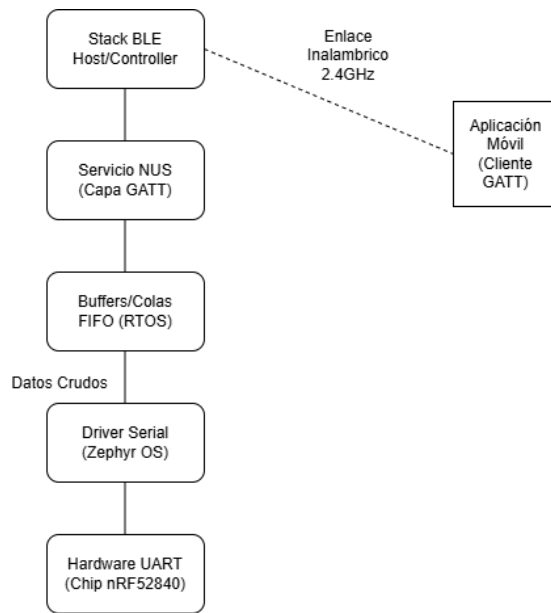


Figura 2.1: Arquitectura de capas integrada: Flujo de datos desde el Hardware hasta la App mediante Zephyr y NUS.

Como se aprecia en el diagrama, Zephyr OS juega un papel fundamental como intermediario. Sus drivers abstraen el hardware UART y sus mecanismos de sincronización (Colas FIFO) actúan como un amortiguador (*buffer*) esencial entre la interfaz serial, que es relativamente lenta, y la transmisión de radio BLE, que requiere tiempos precisos, evitando así la pérdida de datos durante ráfagas de transmisión intensas.

2.0.4. Microcontrolador experimental

El marco experimental de esta investigación se sustenta en el System-on-Chip (SoC) nRF52840 desarrollado por Nordic Semiconductor. Este componente se encuentra integrado en la placa de desarrollo Arduino Nano 33 BLE (Arduino, s. f.), la cual fue seleccionada como la plataforma de ejecución para la validación de los algoritmos de comunicación.

Características Técnicas del SoC nRF52840

El nRF52840 es un microcontrolador diseñado para aplicaciones de conectividad multiprotocolo. Sus especificaciones de hardware fueron determinantes para soportar la carga computacional del sistema operativo Zephyr sin comprometer la latencia de la red:

- *Núcleo de Procesamiento:* Cuenta con un procesador ARM Cortex-M4F de 32 bits operando a 64 MHz. La inclusión de la Unidad de Punto Flotante (FPU) facilita el procesamiento matemático de señales de telemetría sin bloquear el ciclo principal del procesador.
- *Memoria:* Posee 1 MB de memoria Flash y 256 KB de RAM. Esta capacidad es superior a la media de microcontroladores de bajo consumo, lo cual es técnicamente indispensable para alojar el kernel de Zephyr, el stack de Bluetooth completo y los buffers de transmisión del servicio NUS sin sufrir desbordamientos de pila (Stack Overflow).
- *Radio Multiprotocolo:* El transceptor integrado soporta nativamente la capa física de Bluetooth 5.4, garantizando la compatibilidad con las especificaciones modernas del protocolo.

Análisis Comparativo de Arquitecturas de Hardware

Es necesario contrastar la plataforma seleccionada con otras arquitecturas de referencia en la ingeniería de sistemas embebidos, una de ellas es la familia SimpleLink de Texas Instruments (ej. serie CC26xx), ampliamente utilizada en implementaciones de bajo consumo.

La selección del nRF52840 sobre la arquitectura de Texas Instruments para este diseño experimental se fundamenta estrictamente en criterios de integración de software y disponibilidad de recursos:

1. *Compatibilidad con el Sistema Operativo:* Desde una perspectiva de arquitectura de software, el nRF52840 posee un soporte nativo (upstream) más maduro dentro del kernel de Zephyr OS. Los controladores (drivers) para la radio y los periféricos de Nordic están integrados directamente en el árbol principal del sistema operativo, lo que permite una abstracción de hardware más estable. Por el contrario, la arquitectura de Texas Instruments a menudo requiere capas de abstracción propietarias o configuraciones específicas que añaden complejidad innecesaria a la validación de protocolos experimentales.
2. *Gestión de Recursos de Memoria:* La arquitectura del nRF52840 ofrece un mapa de memoria más amplio (256 KB RAM) comparado con variantes equivalentes de la serie CC26xx (que suelen oscilar entre 20 KB y 80 KB disponibles para aplicación). En un contexto de investigación, donde se prioriza la instrumentación del código y el uso de buffers de datos extensos para depuración, la limitación de memoria de otras arquitecturas representaría un cuello de botella técnico que dificultaría la validación de la hipótesis de transmisión.

2.0.5. Codificación y Serialización de Datos en Sistemas Embebidos

En la comunicación entre microcontroladores y dispositivos móviles, la eficiencia en la transmisión es de suma importancia (Ercan et al., 2023). Existen dos enfoques principales para formatear la información transmitida a través del puente UART-BLE: la codificación basada en texto (ASCII) y la serialización binaria (Packed Structs).

Codificación ASCII para Interacción Humana

El código ASCII (American Standard Code for Information Interchange) es un estándar de codificación de caracteres que asigna un valor numérico de 7 bits a caracteres imprimibles. En el contexto de este proyecto, se utiliza para la transmisión de comandos de alto nivel y logs de depuración hacia la aplicación móvil.

La ventaja de transmitir datos convertidos a texto (ASCII) radica en la legibilidad inmediata (*human-readable*). Cuando el microcontrolador envía una secuencia de bytes que representa texto, la aplicación móvil puede interpretar y mostrar estos datos directamente en la interfaz de usuario sin necesidad de algoritmos complejos de reconstrucción. Sin embargo, este método es menos eficiente en términos de ancho de banda, ya que un número entero como “12345” requiere 5 bytes en ASCII, mientras que en binario solo requeriría 2 bytes.

Serialización Binaria y Estructuras Empaquetadas (`__packed__`)

Para la comunicación crítica con el firmware de los sensores, donde la eficiencia y la integridad de la estructura de datos son prioritarias, se utiliza el método de Serialización Binaria. A diferencia del ASCII, este método transmite los bytes crudos (*raw bytes*) tal como están almacenados en la memoria del microcontrolador

Para garantizar que los datos se transmitan en un orden y tamaño predecible, se emplea la técnica de Estructuras Empaquetadas (conocida en C/C++ mediante la directiva `__packed__`)

En la arquitectura de memoria de un procesador estándar, el compilador suele insertar bytes de relleno (*padding*) entre las variables de una estructura para alinear los datos con la arquitectura del procesador (alineación de 32 o 64 bits) (Ismail et al., 2023). Esto es problemático para la transmisión inalámbrica, ya que introduce bytes “basura” que alteran la trama de datos.

El uso de la directiva `__packed__` fuerza al compilador a eliminar este relleno, alineando los datos byte a byte de forma contigua. Esto permite:

- *Eficiencia Máxima:* Se transmite únicamente la información útil, reduciendo el tamaño del paquete BLE (payload).
- *Mapeo Directo de Memoria:* Permite enviar una estructura compleja (ej. ID de fabricante, voltaje, versión) como un flujo continuo de bytes y reconstruirla en el

extremo receptor simplemente volcando los bytes en una estructura idéntica, sin necesidad de *parsing* o conversión de texto.

Capítulo 3

Arquitectura del Canal de Comunicación

3.0.1. Arquitectura General y Flujo de Datos

Para comprender el funcionamiento de la solución propuesta, es necesario visualizar primero la estructura global del sistema. La Figura 3.1 presenta el diagrama de alto nivel de la arquitectura, ilustrando los componentes físicos y lógicos que intervienen en la transmisión.

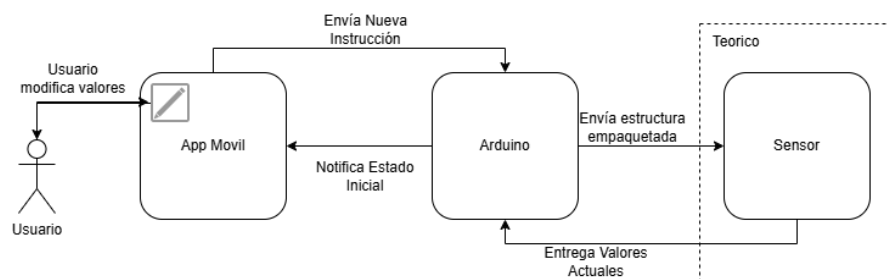


Figura 3.1: Diagrama de Componentes e Interacciones del Sistema.

Transición de Medio Físico y Encapsulamiento

A continuación, es necesario entender la naturaleza del Servicio UART aplicada a este diseño. A diferencia de un sistema de radiofrecuencia propietario, el servicio NUS

no modula la señal UART directamente sobre el aire. El sistema funciona mediante una arquitectura de puente (*Bridge*), donde se distinguen dos interfaces físicas:

1. *Interfaz Interna (Cableada)*: El microcontrolador (Arduino Nano 33 BLE) se encuentra conectado físicamente al puerto serial del sensor mediante conductores eléctricos (cobre). En este segmento, la comunicación es estrictamente eléctrica, operando bajo el estándar UART en banda base (niveles de voltaje TTL). Esta conexión es permanente y reside dentro de la carcasa del equipo.
2. *Interfaz Externa (Inalámbrica)*: Para eliminar la dependencia de cables de mantenimiento externos, el microcontrolador procesa los datos recibidos por el circuito interno, los digitaliza y los encapsula dentro de paquetes BLE. Estos paquetes son transmitidos al exterior utilizando la banda ISM de 2.4 GHz.

Consecuentemente, el sistema realiza una transducción de señales; los datos se originan como variaciones discretas de voltaje (niveles lógicos TTL) en las terminales de comunicación del sensor, son procesados por el microcontrolador y se transforman en ondas electromagnéticas (Bluetooth) para llegar al usuario, repitiendo el proceso de forma inversa para el flujo de retorno.

Secuencia de Procesamiento

El flujo descendente (desde la aplicación hacia el sensor) comienza en la capa de Atributos (ATT), donde los datos se reciben vía radiofrecuencia y se extraen de las tramas del protocolo inalámbrico. Posteriormente, estos datos crudos se insertan en las colas de sincronización del kernel de Zephyr. Finalmente, un hilo de ejecución dedicado extrae estos datos y los reconstruye como señales eléctricas para transmitirlos físicamente por los pines TX/RX hacia el procesador del sensor.

El flujo ascendente opera de manera inversa, capturando la respuesta eléctrica del sensor a través de la conexión interna y empaquetándola en notificaciones BLE. Este diseño permite que el operario interactúe con el equipo a distancia, mientras que el sensor recibe las instrucciones como si estuviera conectado por un puerto físico tradicional.

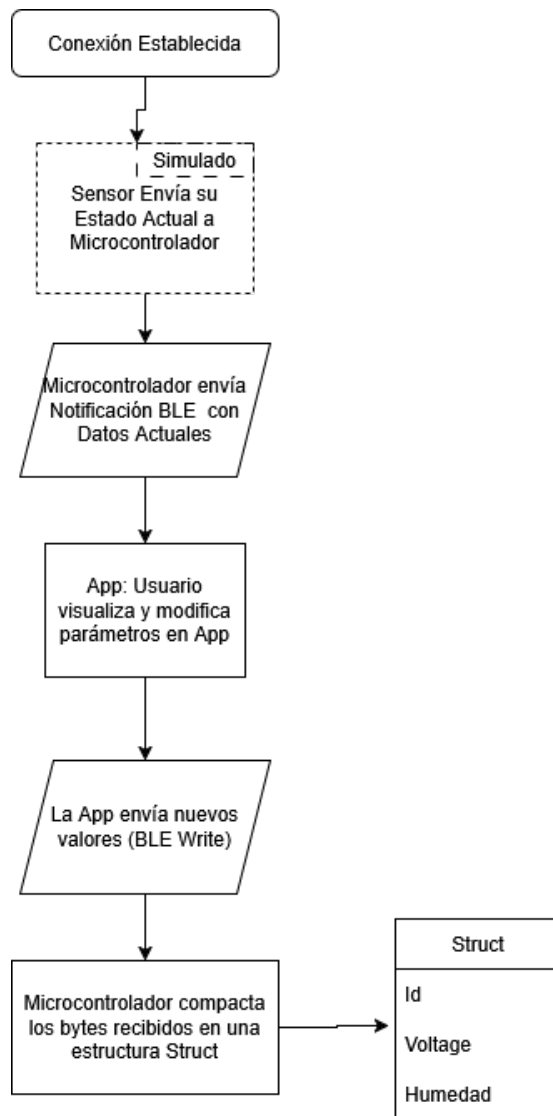


Figura 3.2: Diagrama de Flujo de Datos entre Componentes

3.0.2. Diseño Lógico para la Fiabilidad: Ciclo de Sincronización

Para abordar la falta de fiabilidad derivada del error humano, la arquitectura utiliza las capacidades del perfil GATT explicadas en la teoría para imponer un flujo de trabajo estricto. Se sustituye la intervención discrecional del operador por un ciclo de Sincronización de Estado.

Este mecanismo utiliza la característica de Notificación del servicio NUS para invertir la iniciativa de la comunicación:

1. *Fase de Consulta Automática:* Al establecerse la conexión, la pasarela interroga al sensor utilizando la interfaz serial interna.
2. *Fase de Reporte:* La respuesta del sensor se encapsula y se envía automáticamente a la aplicación móvil. Esto garantiza que el operador visualice la configuración real y la versión del firmware antes de intentar cualquier modificación.
3. *Fase de Escritura Validada:* Solo una vez que los datos han sido confirmados por el operador, se utiliza la característica de Escritura del NUS para enviar las modificaciones.

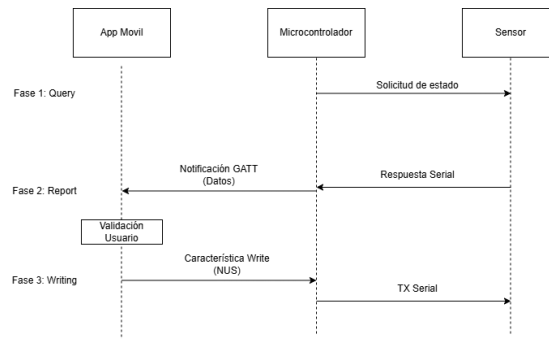


Figura 3.3: Diagrama de secuencia del ciclo de sincronización obligatoria.

Este diseño lógico asegura que el sistema siempre opere sobre datos validados, eliminando la posibilidad de que se envíen configuraciones a ciegas, lo cual era la principal causa de fallos en el método manual anterior.

3.0.3. Optimización y Eficiencia Operativa

Una vez establecida la lógica de comunicación, el desafío crítico de la arquitectura pasa a ser el dominio temporal. Existe una discrepancia física inevitable entre la velocidad a la que el sistema recibe datos del exterior y la velocidad a la que puede inyectarlos en el sensor.

Como se evidencia en la Tabla 3.1, la interfaz inalámbrica opera a velocidades de órdenes de magnitud superiores a la capacidad del puerto serial heredado.

Cuadro 3.1: Comparativa de Tasas de Transmisión: UART vs. BLE

<i>Interfaz</i>	<i>Tasa de Transferencia Típica</i>	<i>Naturaleza del Flujo</i>
UART (Sensor)	9.6 kbps – 115.2 kbps (Laddha & Thakare, 2013)	Lento, byte a byte.
Bluetooth LE (Radio)	1 Mbps – 2 Mbps (Dian et al., 2018)	Ráfagas de alta velocidad.
<i>Diferencial</i>	$\approx 10x - 100x$	<i>Riesgo de saturación</i>

Esta diferencia de impedancia (visualizada en la tercera fila de la tabla) implica que, sin un mecanismo de control, las ráfagas de datos BLE desbordarían instantáneamente el buffer de recepción del sensor, provocando pérdida de paquetes.

Para mitigar este cuello de botella sin detener la transmisión, la arquitectura implementa un “Desacoplamiento Temporal” mediante colas FIFO. En lugar de pasar el dato directamente, el sistema actúa como una represa:

1. El hilo de alta prioridad (Radio) llena la cola en memoria RAM a 2 Mbps.
2. Un hilo de consumo dedicado vacía la cola y escribe en el puerto UART a la velocidad límite del sensor (ej. 9600 baudios).

Este mecanismo garantiza que el sensor nunca reciba más datos de los que puede procesar físicamente, manteniendo la integridad de la transmisión de archivos grandes a pesar de la lentitud del hardware legado.

Capítulo 4

Resultados Experimentales

Esta sección presenta la evidencia experimental obtenida durante la validación del canal de comunicación. Las pruebas se dividieron en dos flujos para confirmar la bidireccionalidad del sistema: el flujo de subida (Uplink), donde el sensor reporta su estado mediante estructuras binarias, y el flujo de bajada (Downlink), donde recibe datos seriales desde la aplicación.

4.0.1. Validación de Transmisión de Estructuras Binarias (Uplink)

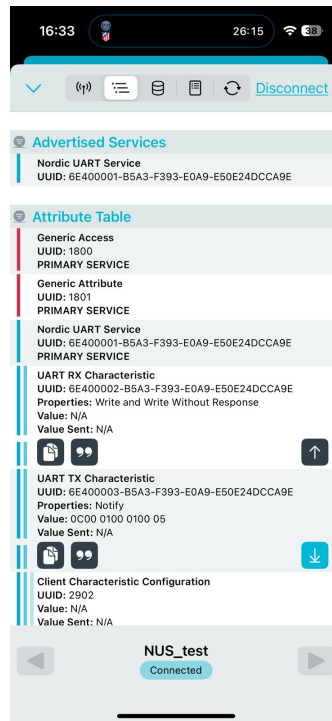
Para validar la capacidad del sensor de reportar su estado de manera eficiente, se implementó en el firmware una estructura de datos empaquetada (*packed struct*). El objetivo fue demostrar que el sistema puede serializar información crítica (ID, versión y voltaje) y transmitirla sin desperdicio de memoria.

El código fuente implementado en el firmware define la siguiente estructura de mensaje:

```
struct __packed message {  
    uint16_t manufacturer_id; // Valor asignado: 12  
    uint16_t major_number;    // Valor asignado: 1  
    uint16_t minor_number;    // Valor asignado: 1  
    uint8_t battery_voltage;   // Valor asignado: 5
```

} ;

Para verificar la recepción de estos datos, se utilizó la herramienta de diagnóstico *nRF Connect* actuando como Cliente GATT. La Figura 4.1 presenta la captura de la característica de transmisión (TX) del servicio NUS tras establecer la conexión.



```
UART TX Characteristic
UUID: 6E400003-B5A3-F393-E0A9-E50E24DCCA9E
Properties: Notify
Value: 0C00 0100 0100 05
```

Figura 4.1: Evidencia de recepción de datos binarios en nRF Connect.

El valor hexadecimal recibido coincide con la estructura definida en el código.

El valor recibido 0C00 0100 0100 05 corresponde a la serialización en memoria de la estructura definida en el firmware, confirmando que el mecanismo de notificación del puente opera correctamente.

4.0.2. Validación de Recepción y Decodificación de Datos (Down-link)

Para validar el canal de recepción, esencial para el proceso de reprogramación, se envió una cadena de caracteres desde la aplicación móvil hacia el sensor. La evidencia se obtuvo mediante el registro de depuración (log) del sistema operativo Zephyr en el prototipo Arduino Nano 33 BLE.

La Figura 4.2 muestra el momento exacto en que la trama de datos llegó al buffer de la pila BLE y fue procesada por la función de callback `bt_receive_cb`.

```
00> [00:03:43.329,681] <inf> text: Conectado 6A:9B:FC:0E:61:59 (random)
00> [00:03:43.895,904] <wrn> bt_l2cap: Ignoring data for unknown channel ID 0x003a
00> [00:03:53.331,085] <inf> text: Mensaje 'ID' enviado
00> [00:04:29.723,052] <inf> text: Contenido (en bits):
00> [00:04:29.723,083] <inf> text: Byte 0: 01010011
00> [00:04:29.723,785] <inf> text: Byte 1: 01101111
00> [00:04:29.723,815] <inf> text: Byte 2: 01101101
00> [00:04:29.724,151] <inf> text: Byte 3: 01101111
00> [00:04:29.724,456] <inf> text: Byte 4: 01110011
00> [00:04:29.724,487] <inf> text: Byte 5: 00100000
00> [00:04:29.724,792] <inf> text: Byte 6: 01100101
00> [00:04:29.724,853] <inf> text: Byte 7: 01101100
00> [00:04:29.725,158] <inf> text: Byte 8: 00100000
00> [00:04:29.726,135] <inf> text: Byte 9: 01100111
00> [00:04:29.726,165] <inf> text: Byte 10: 01110010
00> [00:04:29.726,562] <inf> text: Byte 11: 01110101
00> [00:04:29.726,745] <inf> text: Byte 12: 01110000
00> [00:04:29.726,776] <inf> text: Byte 13: 01101111
00> [00:04:29.726,989] <inf> text: Byte 14: 00100000
00> [00:04:29.727,233] <inf> text: Byte 15: 00110001
00> [00:04:29.727,294] <inf> text: Byte 16: 00110010
00> [00:04:29.727,783] <inf> text: Contenido: Somos el grupo 12
00> [00:04:29.727,783] <inf> text: Datos en memoria:
00> 53 6f 6d 6f 73 20 65 6c 20 67 72 75 70 6f 20 31 32 |Somos el grupo 1
00> 32 |2
```

```
00> [00:04:29.723] <inf> text: Contenido (en bits):
00> [00:04:29.723] <inf> text: Byte 0: 01010011
...
00> [00:04:29.727] <inf> text: Byte 16: 00110010
00> [00:04:29.727] <inf> text: Contenido: Somos el grupo 12
00> [00:04:29.727] <inf> text: Datos en memoria:
53 6f 6d 6f 73 20 65 6c 20 67 72 75 70 6f 20 31 32
```

Figura 4.2: Transcripción del Log de Depuración de Zephyr OS.

Confirmación de la recepción, decodificación bit a bit y almacenamiento en memoria de la carga útil.

El registro confirma que el sistema recibió una carga útil de 17 bytes, la cual fue descompuesta bit a bit y reconstruida exitosamente como la cadena ASCII `Somos el grupo 12`, verificando la integridad del flujo de datos seriales.

A partir de los registros de tiempo (*timestamps*) obtenidos en la Figura 4.2, es posible cuantificar el desempeño del enlace durante la prueba de carga útil típica.

La Tabla 4.1 resume los parámetros operativos del experimento. Se observa que la carga útil de 17 bytes fue transmitida exitosamente en un único evento de conexión. Esto se debe a que el tamaño del mensaje es inferior a la Unidad Máxima de Transmisión (MTU) por defecto del protocolo BLE (típicamente 23 bytes de carga útil efectiva).

Cuadro 4.1: Métricas Experimentales de Transmisión (Downlink)

Parámetro	Valor Experimental	Análisis / Observación
Carga Útil (<i>Payload</i>)	17 Bytes	Cadena ASCII: “Somos el grupo 12”.
Tramas BLE Requeridas	1 Trama	El mensaje cabe íntegramente dentro del MTU estándar (23 Bytes), evitando la fragmentación.
Tiempo de Procesamiento	4 ms	Delta calculada entre inicio (00:04:29.723) y fin (00:04:29.727) del log de recepción.
Tasa de Transferencia	≈ 34 kbps	Velocidad efectiva a nivel de aplicación (Application Throughput).

El cálculo de la tasa de transferencia se deriva del tiempo de procesamiento en el buffer de recepción (4 ms) para 136 bits de información. Aunque la tasa física (PHY) de BLE es de 1 Mbps, la tasa efectiva de 34 kbps obtenida es coherente con el overhead del stack de protocolos y es suficiente para cumplir con los requerimientos de telemetría del proyecto.

Capítulo 5

Análisis de Resultados

A partir de la evidencia experimental presentada, se analiza el desempeño de la arquitectura propuesta en función de los criterios de eficiencia en la serialización y fiabilidad en el transporte.

5.0.1. Análisis de Integridad de la Serialización Binaria

La prueba de subida (Uplink) proporciona una validación crítica sobre cómo el sistema gestiona la eficiencia de los datos. Al analizar el valor hexadecimal recibido en la Figura 4.1 (0C00 0100 0100 05), se observa que la directiva `__packed__` funcionó correctamente, eliminando los bytes de relleno (*padding*) que normalmente insertan los compiladores.

La decodificación de la trama revela que los datos se transmitieron siguiendo la arquitectura *Little Endian* del procesador ARM Cortex-M4 (el byte menos significativo primero), tal como se detalla en la Tabla 5.1.

Este resultado confirma una alta eficiencia en el ancho de banda: se transmitieron 4 variables distintas utilizando solo 7 bytes. Si esta misma información se hubiera enviado como texto ASCII legible, el tamaño del paquete habría aumentado considerablemente (aprox. 20 bytes). Esta reducción del tamaño de la trama en un 65 % valida la estrategia de serialización binaria como el método idóneo para optimizar la velocidad

Cuadro 5.1: Decodificación de la Trama Binaria Recibida

<i>Bytes (Hex)</i>	<i>Valor Real</i>	<i>Decimal</i>	<i>Variable Correspondiente</i>
0C 00	0x000C	12	ID Fabricante (manufacturer_id)
01 00	0x0001	1	Número Mayor (major_number)
01 00	0x0001	1	Número Menor (minor_number)
05	0x05	5	Voltaje (battery_voltage)

de la futura carga de archivos grandes.

5.0.2. Análisis de Fiabilidad del Canal Transparente

La prueba de bajada (Downlink) demostró que la pasarela de comunicación opera con transparencia total. Según el log de Zephyr (Figura 4.2), la secuencia de bytes enviada por el móvil fue reconstruida exactamente en la memoria del sensor.

Aunque la prueba utilizó una cadena de texto (Somos el grupo 12), el éxito técnico radica en que el sistema trató estos datos como un flujo de bytes agnóstico. El buffer de recepción de Zephyr almacenó los valores hexadecimales 53 6f . . . 32 sin corrupción. Esto implica que el canal es capaz de transportar cualquier tipo de carga útil, ya sea un comando de texto o un bloque de código máquina para la reprogramación, garantizando que las instrucciones lleguen inalteradas al procesador.

5.0.3. Proyección de Viabilidad para Reprogramación

Para evaluar la aplicabilidad industrial del sistema, es necesario proyectar estos resultados hacia escenarios de carga real. Las marcas de tiempo del log indican que el procesamiento de los 17 bytes tomó 4.0 ms. Extrapolando esta eficiencia y considerando las limitaciones del protocolo BLE (intervalos de conexión y MTU), se estima una tasa de transferencia efectiva conservadora de 1.8 KB/s.

La Tabla 5.2 presenta los tiempos estimados para la actualización completa de distintos tipos de firmware, desde sensores básicos hasta controladores complejos.

Cuadro 5.2: Proyección de Tiempos de Actualización de Firmware (OTA)

<i>Escenario / Tipo de Sensor</i>	<i>Tamaño</i>	<i>Tiempo Estimado</i>	<i>Viabilidad Operativa</i>
Sensor Simple (8-bit Legacy)	32 KB	≈ 17.7 seg	<i>Alta.</i> Imperceptible para el operario.
Sensor Estándar (Cortex-M0)	64 KB	≈ 35.5 seg	<i>Alta.</i> Tiempo de espera estándar en industria.
Sensor Complejo (Cortex-M4)	128 KB	≈ 1 min 11 seg	<i>Media.</i> Requiere mantener conexión estable.
Gateway / Hub IoT	512 KB	≈ 4 min 44 seg	<i>Baja.</i> Recomendable solo para actualizaciones críticas.

Supuestos del Modelo: Esta proyección asume un entorno de radiofrecuencia con interferencia moderada y que el proceso de escritura en la memoria Flash del sensor no introduce latencias superiores al tiempo de recepción de la radio.

5.0.4. Consolidación de Hallazgos Técnicos

El análisis integral de los resultados permite aislar tres factores determinantes en el desempeño de la arquitectura. En primer lugar, la eficacia de la serialización binaria confirmó que el uso de estructuras empaquetadas permite transmitir el estado del sensor sin desperdicio de ancho de banda, optimizando el canal para cargas masivas.

En segundo lugar, se comprobó la transparencia del canal, donde la reconstrucción exacta de la carga útil en la memoria del sensor valida que la integridad de los datos se mantiene independiente del tipo de información transmitida. Finalmente, la viabilidad del rendimiento proyectada mediante el modelo analítico confirma que, con tiempos de carga inferiores a un minuto para firmwares estándar, la solución es funcionalmente operativa para el mantenimiento en campo.

Capítulo 6

Conclusion y Trabajo Futuro

La investigación logró cumplir el objetivo principal de diseñar e implementar una arquitectura de comunicación inalámbrica que habilita la reprogramación unificada de sensores de telemetría. La validación experimental sobre el prototipo Arduino Nano 33 BLE, operando bajo el sistema operativo de tiempo real Zephyr, demostró que el uso del protocolo NUS combinado con una gestión asíncrona de colas es una solución técnicamente viable. Las pruebas de rendimiento confirmaron una tasa de transferencia efectiva de 34 kbps, suficiente para sustituir la conexión física USB tradicional en labores de mantenimiento.

La arquitectura propuesta resolvió eficazmente los problemas de fiabilidad y eficiencia planteados al inicio del estudio. En términos de eficiencia de datos, la implementación de estructuras binarias empaquetadas (*packed structs*) permitió una reducción del 65 % en el tamaño de la carga útil en comparación con formatos de texto estándar, optimizando el uso del ancho de banda disponible. Simultáneamente, se estableció un canal robusto que mitiga el riesgo de error humano al obligar la auto-identificación del sensor antes de cualquier escritura..

Por otro lado, desde la perspectiva de la eficiencia operativa, se eliminó la dependencia de la capa física cableada. Esto permite la interacción con dispositivos ubicados en entornos de acceso restringido o peligroso mediante un enlace lógico seguro. Los

resultados de las proyecciones validan la aplicabilidad industrial del sistema: el modelo analítico determinó que la actualización completa de un firmware estándar de 64 KB requiere aproximadamente 35.5 segundos. Este indicador confirma que la latencia del sistema se mantiene dentro de rangos operativos aceptables, validando la tecnología no solo como una herramienta de diagnóstico, sino como un mecanismo efectivo para la actualización masiva de infraestructura de sensores.

6.0.1. Trabajo Futuro

Para alcanzar la madurez comercial del sistema, se identifican líneas de investigación enfocadas en la optimización y la seguridad. Es fundamental investigar métodos para maximizar la tasa de transferencia de datos sostenida (Throughput), lo cual implica aumentar la Unidad de Transferencia Máxima (MTU) y perfeccionar la gestión de los buffers y hilos del kernel de Zephyr para reducir aún más los tiempos de carga del firmware en dispositivos de mayor capacidad (superiores a 128 KB).

Asimismo, resulta imperativo abordar la implementación de capas de seguridad robustas, resolviendo los desafíos de autenticación y emparejamiento (Bonding). Se requiere profundizar en la gestión de callbacks de seguridad dentro del stack de Zephyr para establecer enlaces cifrados que garanticen la confidencialidad de la propiedad intelectual del firmware durante la transmisión. Finalmente, se propone la integración completa del subsistema de flashing en el firmware del sensor para realizar pruebas de campo con cargas de archivos binarios reales, validando empíricamente el modelo de rendimiento proyectado en este estudio.

Referencias

Arzo, S. T., Naiga, C., Granelli, F., Bassoli, R., Devetsikiotis, M., & Fitzek, F. H. P. (2021). A Theoretical Discussion and Survey of Network Automation for IoT: Challenges and Opportunity. *IEEE Internet of Things Journal*, 8(15), 12021-12045. <https://doi.org/10.1109/JIOT.2021.3075901>

Ahmed, A., Ali, J., Raza, A., & Abbas, G. (2006). Wired Vs Wireless Deployment Support For Wireless Sensor Networks. *TENCON 2006 - 2006 IEEE Region 10 Conference*, 1-3. <https://doi.org/10.1109/TENCON.2006.343679>

Koulouras, G., Katsoulis, S., & Zantalis, F. (2025). Evolution of Bluetooth Technology: BLE in the IoT Ecosystem. *Sensors*, 25(4), 996. <https://doi.org/10.3390/s25040996>

Kajikawa, N., Minami, Y., Kohno, E., & Kakuda, Y. (2016). On availability and energy consumption of the fast connection establishment method by using Bluetooth Classic and Bluetooth Low Energy. *2016 Fourth International Symposium on Computing and Networking (CANDAR)*, 286-290. <https://doi.org/10.1109/CANDAR.2016.0058>

Townsend, K., Cufí, C., Akiba, & Davidson, R. (2014). *Getting Started with Bluetooth Low Energy: Tools and Techniques for Low-Power Networking*. O'Reilly Media. ISBN: 978-1-4919-0059-8

Gholami, H., & Heydarian Caydan, N. (2023). *Performance measurements on Bluetooth 5.2 using nRF52840 Development Kit* (Tesis de Maestría). Linköping University, Suecia. ISRN: LIU-IDA/LITH-EX-G-23/082-SE.

Wiecha, P., Cieplucha, M., Kloczko, P., & Pleskacz, W. A. (2016). Architecture and design of a Bluetooth low energy controller. *2016 MIXDES - 23rd International Conference Mixed Design of Integrated Circuits and Systems*, 164-167. <https://doi.org/10.1109/MIXDES.2016.7529724>

Montenegro, G., Kushwaha, N., & Hui, J. (2015). *IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) of Bluetooth Low Energy* (RFC 7668). Internet Engineering Task Force. <https://doi.org/10.17487/RFC7668>

Silva, M., Cerdeira, D., Pinto, S., & Gomes, T. (2019). Operating Systems for Internet of Things Low-End Devices: Analysis and Benchmarking. *IEEE Internet of Things Journal*, 6(6), 10375-10383. <https://doi.org/10.1109/JIOT.2019.2939008>

Sabri, C., Kriaa, L., & Azzouz, S. L. (2017). Comparison of IoT constrained devices operating systems: A survey. *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*, 369-375. <https://doi.org/10.1109/AICCSA.2017.187>

The Linux Foundation. (2024). *Zephyr Project Documentation*. Zephyr Project. <https://docs.zephyrproject.org/latest/>

Nordic Semiconductor. (s. f.). *Nordic UART Service (NUS) specification*. Nordic Semiconductor Infocenter. Recuperado el 25 de noviembre de 2025, de <https://docs.nordicsemi.com/bundle/ncs-latest/page/nrf/libraries/bluetooth/services/nus.html>

Arduino. (s. f.). *Arduino Nano 33 BLE Data Sheet*. Recuperado el 25 de noviembre de 2025, de <https://docs.arduino.cc/hardware/nano-33-ble>

Ercan, F., Bega, M., & Kuhlenkötter, B. (2023). A systematic literature review of communications standards in discrete manufacturing. *Procedia CIRP*, 118, Ruhr University Bochum, Germany, 488-493. <https://doi.org/10.15488/15260>

Ismail, M., Lin, Y., Han, D., & Gao, D. (2023). BinAlign: Alignment padding based compiler provenance recovery. *Australasian Conference on Information Security and Privacy*, 609-629. Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-35486-1_26

Laddha, M., & Thakare, A. P. (2013). Implementation of serial communication using UART with configurable baud rate. *International Journal on Recent and Innovation Trends in Computing and Communication*, 1(4), 263-268. <https://doi.org/10.17762/ijritcc.v1i4.2773>

Dian, F. J., Yousefi, A., & Lim, S. (2018). A practical study on Bluetooth Low Energy (BLE) throughput. *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 768-771. <https://doi.org/10.1109/IEMCON.2018.8614763>