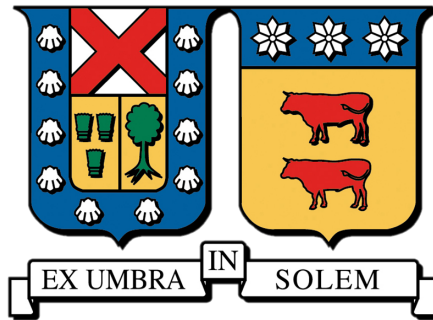


UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAÍSO - CHILE



**Diseño e implementación de una plataforma IoT de
monitoreo de vehículos de movilidad compartida**

Cristóbal Ignacio Caqueo Ledesma

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

PROFESOR GUÍA:

MOHAMED ABDELHAMID

PROFESOR CORREFERENTE:

MELISSA DIAGO

MARZO 2025

*A mis padres, quienes siempre me apoyaron y confiaron en mi,
A mis amigos, que siempre me motivaron y me ayudaron a salir adelante,
A mis profesores, que me formaron y me han guiado durante este viaje,
A todos los que se han mantenido a mi lado a lo largo de estos años,
Esto es para ustedes.*

Agradecimientos

Agradezco al Profesor Mohamed por apoyarme a lo largo de todo este proceso y a Francisca por ayudarme cada vez que lo necesité. A los que me ayudaron a hacer pruebas y aguantaron tantas horas junto a mí. Agradezco a todas las personas que me acompañaron a lo largo de mi proceso universatario. A mis compañeros y amigos con los que compartimos la experiencia de crecer y aprender como miembros de esta comunidad. A los profesores que me transmitieron la emoción y la pasión por la ingeniería y la tecnología a través de sus clases. A todas las personas que me ayudaron a sobrellevar cualquier dificultad que se haya dado durante el camino. A la gente que nos apoyó como familia cuando más lo necesitamos, durante aquellos meses de incertidumbre. Porque todos han sido parte de mi camino, y sin ustedes no estaría donde estoy. Porque siempre les estaré agradecido.

Resumen

La movilidad compartida es un tema que ha ganado importancia en los últimos años. Esto debido a la popularidad de los vehículos eléctricos y a la creciente necesidad de adoptar nuevas soluciones de movilización para las ciudades. con el fin de reducir emisiones y adaptarse a las crecientes poblaciones. Así, se ha buscado implementar distintas soluciones para promover la evolución de las ciudades y de la movilidad. Desde aplicaciones como Uber para solicitar viajes en auto, hasta plataformas como Grin para solicitar el uso de scooters eléctricos individuales. En general, estas soluciones se basan en comunicaciones 4G y dependen de los teléfonos de los usuarios para funcionar. Por otra parte, LoRaWAN (Long Range Wide Area Network) es un protocolo de comunicaciones de bajo consumo con un alto alcance ampliamente utilizado en aplicaciones IoT. Con el fin de probar el funcionamiento de LoRaWAN para redes IoT móviles, y verificar su viabilidad para el futuro de las ciudades y la movilización, se diseñó e implementó una plataforma IoT usando nodos transmisores y receptores, gateways LoRa, y también una aplicación en Android para comparar el funcionamiento. Los resultados de la implementación muestran que una red a mayor escala es viable, siempre y cuando se mantenga un número adecuado de puntos de acceso bien ubicados dentro de la zona que se desea cubrir.

Palabras claves: Movilidad compartida, LoRaWAN, LoRa, ciudades inteligentes, IoT.

Abstract

Shared mobility is a topic that has gained importance in recent years. This is due to the popularity of electric vehicles and the growing need to adopt new mobility solutions for cities in order to reduce emissions and adapt to increasing populations. As a result, various solutions have been implemented to promote the evolution of cities and mobility. These range from applications like Uber for requesting car rides to platforms like Grin for renting individual electric scooters. In general, these solutions rely on 4G communications and depend on users' smartphones to function.

On the other hand, LoRaWAN (Long Range Wide Area Network) is a low-power communication protocol with long-range capabilities that is widely used in IoT applications. To test the functionality of LoRaWAN for mobile IoT networks and evaluate its viability for the future of cities and mobility, an IoT platform was designed and implemented using transmitter and receiver nodes, LoRa gateways, and an Android application to compare performance. The implementation results show that a larger-scale network is viable, as long as an adequate number of well-placed access points are maintained within the area to be covered.

Key words: Shared mobility, LoRaWAN, LoRa, Smart Cities, IoT.

Glosario

- **Movilidad compartida:** Sistema de transporte donde los usuarios comparten vehículos para optimizar recursos y reducir costos.
- **Micromovilidad:** Uso de vehículos ligeros como bicicletas y scooters eléctricos para desplazamientos cortos en entornos urbanos.
- **Plataforma IoT:** Sistema que conecta, gestiona y analiza datos de dispositivos inteligentes a través de internet.
- **LoRaWAN:** Protocolo de comunicación de baja potencia y largo alcance usado en redes IoT.
- **LoRa:** Tecnología de modulación utilizada en redes LoRaWAN para la transmisión eficiente de datos.
- **Capa de percepción:** Componente de un sistema IoT encargado de la recolección de datos mediante sensores.
- **Capa de comunicaciones:** Infraestructura de red que permite la transmisión de datos en un sistema IoT.
- **Capa de middleware:** Interfaz intermedia entre dispositivos y aplicaciones que gestiona la comunicación y el procesamiento de datos.
- **Capa de aplicación:** Nivel donde los datos procesados se presentan a los usuarios a través de interfaces gráficas o dashboards.
- **Nodo sensor (NS):** Dispositivo que recopila información del entorno y la envía a la red IoT.
- **Gateway (GW):** Dispositivo que actúa como intermediario entre nodos sensores y servidores en la nube.

- **Servidor en la nube (SV):** Infraestructura de almacenamiento y procesamiento de datos IoT accesible desde internet.
- **Unidad de Medición de Inercia (IMU):** Sensor que mide aceleración y orientación en múltiples ejes.
- **GPS (Sistema de Posicionamiento Global):** Tecnología que permite determinar la ubicación geográfica en tiempo real.
- **The Things Network (TTN):** Plataforma LoRaWAN para la gestión y conexión de dispositivos IoT.
- **ThingSpeak:** Plataforma en la nube para almacenamiento y visualización de datos IoT.
- **Node-RED:** Herramienta de programación visual para el procesamiento y automatización de datos IoT.
- **MongoDB:** Base de datos NoSQL utilizada para gestionar grandes volúmenes de datos IoT.
- **Ubidots:** Plataforma de análisis y visualización de datos en tiempo real para dispositivos IoT.
- **Freeboard:** Aplicación de código abierto para crear dashboards de visualización de datos IoT.
- **Blynk:** Plataforma para la gestión y visualización remota de dispositivos IoT mediante una app móvil.
- **Redes celulares (4G/5G, NB-IoT):** Tecnologías de comunicación móvil utilizadas en redes IoT.
- **Fog Computing:** Modelo de computación que acerca el procesamiento de datos a los dispositivos finales para reducir latencia.

- **Sensores y actuadores:** Dispositivos que detectan cambios en el entorno y generan respuestas automáticas.
- **MQTT:** Protocolo de comunicación ligero utilizado en IoT para la transmisión de datos.
- **ABP (Activation By Personalization):** Método de activación de dispositivos LoRaWAN en el que se configuran manualmente las credenciales de autenticación, sin necesidad de un servidor de red.
- **DEVADDR:** Dirección única asignada a un dispositivo LoRaWAN dentro de una red específica.
- **DEVEUI:** Identificador único extendido de un dispositivo LoRaWAN, asignado por el fabricante.
- **GATEWAY_EUI:** Identificador único extendido asignado a un gateway LoRaWAN, usado para su autenticación en la red.

Índice de figuras

1.1. Topología general del sistema	7
2.1. Empresas operando en Santiago y tarifas referenciales	13
3.1. Modelo de tres niveles	16
3.2. Modelo de referencia	16
3.3. Elementos estándar de LoRaWAN	17
3.4. Prototipo de implementación propuesto	18
3.5. Esquema de red de sistema de transporte inteligente	19
3.6. Razón de entrega de paquetes: a)razón de entrega de paquetes para una red con 20B de payload y rapidez de los nodos de 5 m/s. b)razón de entrega de paquetes para una red de 2000x2000 usando el modelo de movilidad de Gauss-Markov	21
3.7. Razón de entrega de paquetes: a)razón de entrega de paquetes para una red de 1000x1000, 100B de payload y rapidez de los nodos de 25 m/s. b) razónn de entrega de paquetes para una red de 1000x1000 usando el modelo de movilidad random waypoint.	21
3.8. a) Consumo de energía promedio por dispositivo para una red de 2000x2000, 4000 dispositivos y 60 B de payload a distintas velocidades en m/s. b) Consumo de energía promedio por dispositivo para una red de 2000x2000 usando el modelo de movilidad random waypoint, con 20 B de payload a distintas velocidades en m/s.	22

3.9. a) Consumo de energía promedio por dispositivo para una red de 1000x1000, 20B de payload, y 5 m/s de velocidad en redes de distinto tamaño. b) Consumo de energía promedio por dispositivo para una red de 1000x1000 usando el modelo de movilidad Gauss-Markov, con 20 B de payload para diferentes velocidades en m/s.	22
3.10. Esquema de comunicación usando LoRaWAN	23
3.11. Área de cobertura de de LoRaWAN	24
3.12. Arquitectura general de ITS implementado en red 5G	25
3.13. Diagrama de flujo IoT	26
3.14. Resumen de proyectos con diferentes tecnologías de comunicación. .	27
4.1. Tecnologías por capa	28
5.1. Escenario 1	37
5.2. Escenario 2	38
5.3. Escenario 3	38
5.4. Diagrama de conexiones Tx	40
5.5. Diagrama de conexiones Rx	40
5.6. Dragino LG308	41
5.7. Interfaz de usuario de aplicación de Andriod	43
5.8. Canal de ThingSpeak creado para la aplicación	44
5.9. Configuración de radio LoRa	45
5.10. Configuración de radio LoRa	45
5.11. Configuración de ABP	45
5.12. Configuración de ABP	46
5.13. Configuración MQTT	46
5.14. Tráfico del gateway	46
5.15. Flujo para nodo receptor	47
5.16. Flujo para gateway	48
5.17. Dashboard para escenario 1	49

5.18. Dashboard para escenario 2	49
5.19. Flujo node-red para escenario 3	50
5.20. Dashboard escenario 3	50
5.21. Ubicación donde se tomó el test. Imagen extraída de Google Maps.	
Rectángulo naranja indica dónde se ubicó el gateway/nodo receptor. .	51
5.22. Google Maps	51
5.23. Ubicaciones de test estático	53
5.24. Vista satelital de ubicación de test estático	53
5.25. Resultados RSSI y SNR para DR_SF7	54
5.26. Resultados RSSI y SNR para DR_SF8	55
5.27. Resultados RSSI y SNR para DR_SF9	55
5.28. Resultados RSSI y SNR para DR_SF10	56
5.29. Porcentaje de paquetes recibidos en cada posición para cada Data Rate	57
5.30. Resultados RSSI y SNR para DR_SF7	59
5.31. Resultados RSSI y SNR para DR_SF8	59
5.32. Resultados RSSI y SNR para DR_SF9	60
5.33. Resultados RSSI y SNR para DR_SF10	60
5.34. Porcentaje de paquetes recibidos en cada posición para cada Data Rate	62
6.1. Escenario de simulación con un solo gateway	65
6.2. Escenario de simulación un nodo con varios gateways	65
6.3. Escenario de simulación con varios gateways	66
6.4. Cantidad de paquetes recibidos por cada Gateway	67
6.5. Nivel de batería del nodo transmisor	67
6.6. Consumo del nodo transmisor	68
6.7. Ruta seguida por el nodo transmisor	68
6.8. Ruta seguida en el escenario 3	69
6.9. Ruta para el nodo S70	71
6.10. Ruta para el nodo S75	71

6.11. Ruta para el nodo S74	71
6.12. Ruta del nodo S72	72
6.13. Paquetes recibidos por cada gateway para cada nodo	73
6.14. Paquetes enviados, recibidos y únicos para cada nodo	73

Índice de cuadros

4.1. Perfil de usuario	34
5.1. Soluciones implementadas	37
5.2. Configuraciones de Data Rate para la banda US915	52
5.3. Valores de atenuación para diferentes posiciones y data rates	56
5.4. Valores de SNR para diferentes posiciones y data rates	57
5.5. Porcentaje de paquetes recibidos en cada posición para cada Data Rate	58
5.6. Pérdida de paquetes en porcentaje para cada Data Rate	58
5.7. Valores de atenuación para diferentes posiciones y data rates	61
5.8. Valores de SNR para diferentes posiciones y data rates	61
5.9. Proporción de paquetes recibidos en porcentaje para diferentes posi- ciones y data rates	62
5.10. Pérdida de paquetes en porcentaje para cada Data Rate	63
6.1. Escenarios para la simulación en CupCarbon	64
6.2. Pérdida de paquetes	69
6.3. Pérdida de paquetes por cada nodo	72

Índice general

1. Introducción	5
1.1. ¿Qué es la movilidad compartida?	5
1.2. Objetivo	6
1.3. Análisis del objetivo	6
1.4. Acercamiento a la solución	8
1.5. Estructura del documento	9
2. Marco Teórico	11
2.1. Movilidad compartida [16][17]	11
2.2. Micromovilidad [18]	11
2.3. Plataforma IoT [19]	12
2.4. LoRaWAN [20][21]	12
2.5. Empresas de servicios de movilidad compartida en Chile [22]	13
3. Estado del Arte	14
3.1. Fog Computing Approach for Shared Mobility	14
3.2. Sistema de transporte público usando LoRa	18
3.3. Sobre el uso de LoRaWAN en aplicaciones IoT	20
3.4. Sobre las capacidades de LoRa	23
3.5. Sistema de transporte inteligente usando 5G	24
3.6. Monitoreo remoto de e-bikes	25
3.7. Comparación	27

4. Arquitectura propuesta	28
4.1. Capa de percepción	28
4.2. Capa de Comunicaciones	30
4.3. Capa de Middleware	30
4.4. Capa de aplicación	31
4.5. Requisitos del sistema	32
4.5.1. Requisitos funcionales	32
4.6. Requisitos no funcionales	33
4.7. Tecnologías utilizadas	33
4.8. Propuesta de solución	34
5. Diseño de prototipos y resultados	36
5.1. Hardware	39
5.1.1. LoRa Node	39
5.1.2. Gateway	40
5.2. Software	41
5.2.1. Nodo transmisor	41
5.2.2. Gateway	43
5.2.3. Nodo receptor	47
5.2.4. Node-red	47
5.3. Pruebas	50
5.4. Resultados	54
5.4.1. Escenario 1	54
5.4.2. Escenario 2	59
6. Resultados de Simulación	64
6.1. Escenario 1	66
6.1.1. Escenario 2	68
6.2. Escenario 3	69
6.3. Escenario 4	70

Capítulo 1

Introducción

La movilización y los medios de transporte son una pieza fundamental en la sociedad y las ciudades de cada país. En Chile, particularmente en la región metropolitana, el 57.4 % de la población declara usar el transporte público, compuesto de micros y buses. El 47.8 % usa el metro, el 31.5 % utiliza el automóvil y el 28.5 % declara moverse caminando [16]. Como es posible observar, la mayoría de la población se mueve utilizando el transporte público. En general, estos sistemas presentan varias ventajas para la organización de una ciudad. Desde descongestionamiento de las autovías, hasta menor cantidad de emisiones contaminantes. Por su parte, para los ciudadanos resulta una opción más económica que adquirir un vehículo particular, sin embargo, estos deben ser capaces de cumplir con sus necesidades, es decir, que les permitan llegar a su destino a tiempo. Por otro lado, está la creciente necesidad de reducir emisiones mediante la incorporación de vehículos eléctricos. En Chile, se espera que para el 2035, el 100 % de las ventas de vehículos particulares sean eléctricos, según lo propuesto en la estrategia nacional de electromovilidad [15]

1.1. ¿Qué es la movilidad compartida?

La movilidad compartida es un sistema de transporte en el cual los usuarios comparten un vehículo, ya sea de manera simultánea en grupo, o a lo largo del tiempo.

Esto crea un sistema híbrido entre vehículos particulares y transporte público. Cada sistema de movilidad compartida tiene su propio rango de efectos en el comportamiento de los viajes, el medio ambiente y el desarrollo de las ciudades. Algunos de estos efectos incluyen un mayor acceso a movilidad y una reducción de la cantidad de vehículos privados.

Con el paso del tiempo, se ha podido observar como los sistemas de movilidad compartida se han ido incorporando en distintas ciudades a lo largo del mundo. Desde autos hasta bicicletas y scooters eléctricos.

1.2. Objetivo

Con el fin de aportar en el desarrollo de la movilidad compartida, e ir moldeando el camino para la implementación de ciudades inteligentes en un futuro, **este trabajo busca diseñar una red de comunicaciones para monitorear el estado y ubicación de vehículos de movilidad compartida en ciudades y/o recintos privados y públicos**. El diseño debe incluir nodos sensores, red de comunicación, almacenamiento de datos y una interfaz de usuario para visualizar la información. El trabajo contempla el uso de tecnologías como LoRa para la red de comunicaciones, y parte de este es comprobar la eficiencia de la tecnología para este uso.

1.3. Análisis del objetivo

Cómo se mencionó, el objetivo es diseñar una plataforma de monitoreo de vehículos de movilidad compartida, **en particular scooters eléctricos**, usando tecnologías IoT (Internet of Things) como LoRa que permite la transmisión rápida y de bajo costo de manera inalámbrica. Para esto son necesarios distintos componentes, tales como los **nodos sensores (NS)**, **gateways (GW)**, **servidores en la nube (SV)**. Los NS cumplen con la función de recopilar la información necesaria para el funcionamiento de la plataforma, tales como la velocidad, aceleración y posición de los vehículos. Estos

también envían los paquetes usando LoRaWAN. Los GW son routers equipados con un concentrador LoRa, lo que les permite recibir estos paquetes. Por lo general, se pueden encontrar dos tipos:

- Los que se ejecutan en un firmware mínimo, lo que las hace de bajo costo y fáciles de usar (por ejemplo, The Things Gateway), que ejecutan solo el software de reenvío de paquetes.
- Los que ejecutan un sistema operativo, para las que el software de reenvío de paquetes se ejecuta como un programa en segundo plano (por ejemplo, Kerlink IoT Station, Multitech Conduit). Esto da más libertad al administrador del GW para administrar su puerta de enlace e instalar su propio software.

Por su parte, los SV se encargan de recibir la información para que esta pueda ser usada luego en las aplicaciones correspondientes. El objetivo consta de de crear una interfaz que permita a los administradores de la plataforma el observar los valores de las variables transmitidas, por lo que el uso de APIs (Application Programming Interface) resulta útil para recopilar la información y desplegarla de manera entendible. Para este fin existen SV como los de *The Things Network*, que permiten conexión con aplicaciones como *ThingSpeak*, para el despliegue de información en aplicaciones IoT.

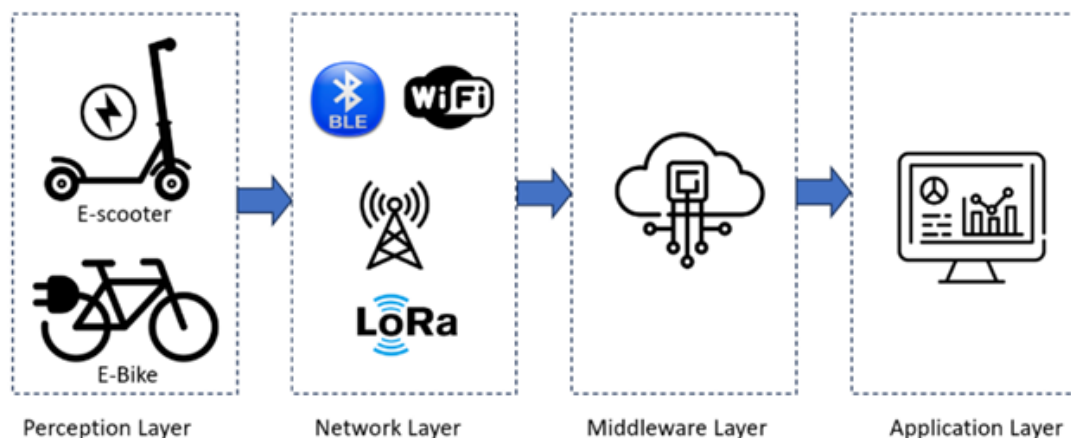


Figura 1.1: Topología general del sistema

Usando la topología propuesta en Fig. 1.1, los NS pertenecen a la capa de percepción, los GW pertenecen a la capa de red, el SV pertenece a la capa de middleware, y la aplicación que contiene la interfaz de usuario pertenece a la capa de aplicación. Teniendo esto en cuenta, el trabajo a realizar es el siguiente:

1. Identificar componentes a utilizar. Sensores, modelos, microcontroladores, vehículo para comprobar funcionamiento, además del software a utilizar para crear interfaz de usuario que permita visualizar la información transmitida.
2. Se debe diseñar el firmware para los sensores, gateway y otros equipos que se utilicen.
3. Se debe configurar el gateway LoRa.
4. Se debe configurar el SV y establecer la conexión entre el GW y SV.
5. Se debe establecer la conexión entre el SV y la aplicación de usuario. Esto incluye el diseño de la IU (Interfaz de Usuario).
6. Se deben realizar pruebas de conexión para obtener los mejores parámetros, como Spreading Factor (SF) y ancho de banda (BW).

Esta serie de hitos nos permite construir la plataforma que ofrece monitoreo constante de los vehículos que estén a disposición de los administradores.

1.4. Acercamiento a la solución

Como se ha propuesto, la movilidad compartida es una solución que facilita el desarrollo de las ciudades. Esta incluye distintos tipos de vehículos y distintos modos de operación. En este trabajo nos enfocamos en **scooters eléctricos de uso individual**, provistos por una empresa privada para los habitantes de, en este caso, la comuna de

Viña del Mar. Para la implementación de estos servicios, se requiere una plataforma que recopile la información y la transmita al servidor de la compañía. Las soluciones a esta problemática pueden tener distintas formas, incluyendo el uso de LoRa, que es el enfoque de este trabajo, pero también el uso de los teléfonos de los usuarios que buscan usar estos vehículos. Por esto, el objetivo principal es diseñar una estructura IoT basada en LoRaWAN que ofrezca las funcionalidades requeridas por la empresa. Luego, se busca ofrecer la misma solución usando una solución basada en los dispositivos móviles de los usuarios. En esta, los celulares pasan a ser los NS, y estos mismos se encargan de transmitir la información por 4G/5G al servidor de la compañía. Con esto, se busca comparar ambos enfoques y ofrecer los beneficios y las dificultades de cada modelo.

1.5. Estructura del documento

El documento se estructura de la siguiente forma: el capítulo 2 es el marco teórico, en donde se plantean y definen los conceptos más importantes para el proyecto.

El capítulo 3 postula el estado del arte, en donde se exploran distintas tecnologías y soluciones similares para la implementación del proyecto. Al final se hace una comparación entre todas las propuestas.

En el capítulo 4 se propone la arquitectura del sistema. Para esto, se exploran nuevamente las distintas opciones para cada capa de la plataforma. Al final, se realiza la propuesta con las tecnologías que se usarán para el desarrollo y la implementación.

En el capítulo 5 se describe el proceso de implementación y desarrollo de la plataforma. Se hace un paso a paso de cómo es la arquitectura de cada solución, y cómo se configuraron los dispositivos más importantes. Finalmente, se hacen pruebas de funcionamiento y se presentan los resultados para los distintos escenarios.

En el capítulo 6 se muestran los resultados de las simulaciones hechas para demostrar escalabilidad del sistema y corroborar los resultados de las pruebas físicas.

Finalmente, en el capítulo 7 se muestran las conclusiones del trabajo. Se hace una

breve discusión de los resultados obtenidos y una comparación de las soluciones. Por último, se propone un trabajo futuro.

Capítulo 2

Marco Teórico

2.1. Movilidad compartida [16][17]

La movilidad compartida es un modelo de transporte basado en la utilización conjunta de vehículos y otros medios de transporte por parte de múltiples usuarios, con el fin de optimizar los recursos y reducir costos operativos y ambientales. Este concepto abarca diversas modalidades, como el carpooling, el carsharing, el uso compartido de bicicletas y scooters, y los servicios de transporte bajo demanda. La movilidad compartida ha ganado relevancia en los últimos años debido a la creciente urbanización y la necesidad de soluciones más sostenibles para la movilidad urbana. La integración de tecnologías digitales, como aplicaciones móviles y sistemas de geolocalización, ha facilitado la implementación y expansión de estos servicios, promoviendo un uso más eficiente del espacio y los recursos de transporte.

2.2. Micromovilidad [18]

La micromovilidad se refiere al uso de vehículos ligeros, generalmente eléctricos o impulsados por fuerza humana, diseñados para recorridos cortos en entornos urbanos. Incluye medios de transporte como bicicletas, patinetes eléctricos y monopatines, que

permiten desplazamientos eficientes en distancias menores a 10 kilómetros. La micromovilidad ha ganado popularidad como una solución sostenible para la congestión del tráfico y la reducción de emisiones de carbono. Su implementación se ha visto impulsada por avances en tecnologías de baterías, plataformas digitales de alquiler y regulaciones urbanas que fomentan el uso de estos vehículos en carriles exclusivos y zonas de tráfico reducido.

2.3. Plataforma IoT [19]

Una plataforma IoT (Internet of Things) es un sistema tecnológico que permite la conexión, gestión y análisis de dispositivos inteligentes a través de internet. Estas plataformas proporcionan una infraestructura que facilita la recopilación, procesamiento y transmisión de datos desde sensores y dispositivos remotos, permitiendo su integración con otras tecnologías como la inteligencia artificial y el big data. Existen diferentes tipos de plataformas IoT, que pueden enfocarse en la conectividad de dispositivos, la gestión de datos, la seguridad, o el análisis en tiempo real. En el ámbito industrial y de movilidad, las plataformas IoT juegan un papel fundamental al habilitar la monitorización remota, el mantenimiento predictivo y la optimización de recursos en sistemas complejos.

2.4. LoRaWAN [20][21]

LoRaWAN (Long Range Wide Area Network) es un protocolo de comunicación de red diseñado para la transmisión de datos de baja potencia a largas distancias en entornos de IoT. Utiliza la tecnología LoRa (Long Range) para proporcionar conectividad eficiente y de bajo consumo energético en redes que requieren comunicación intermitente o de baja velocidad de datos. LoRaWAN es ideal para aplicaciones como la monitorización de sensores en ciudades inteligentes, la agricultura de precisión y la telemetría industrial, ya que permite la transmisión de datos a varios kilómetros de

distancia con un consumo mínimo de energía. Además, la arquitectura de LoRaWAN está basada en una estructura de red en estrella, donde los dispositivos finales se comunican con gateways que transmiten la información a un servidor central para su procesamiento y análisis.

2.5. Empresas de servicios de movilidad compartida en Chile [22]

Existen distintas empresas en Chile que ofrecen servicios de movilidad compartida. En la figura 2.1, extraída del ministerio de energía, se presenta una tabla comparativa de distintas opciones para los usuarios, junto con tarifas referenciales al año 2020, el tipo de vehículo, entre otras características.

EMPRESAS OPERANDO EN SANTIAGO Y TARIFAS REFERENCIALES

Aquí se presentan algunas empresas que operan en Santiago con vehículos eléctricos de micromovilidad, cuya cobertura de servicio se da principalmente en el sector oriente de la capital. En esta tabla se dan tarifas referenciales y otras informaciones de interés para los usuarios

	Lime (scooter)	Grin (scooter)	Movo (scooter)	Scoot (scooter)	Scoot (E-Bike)	Uber JUMP (E-Bike)
Precio de desbloqueo	\$400	\$0	\$0	\$0	\$0	\$0
Precio por minuto	\$100	\$120	\$110	\$100	\$100	\$150
¿Permite reservas?	Si, \$100 por min	No	5 min gratis	15 min gratis	15 min gratis	5 min gratis
Multa por fuera de zona	Hasta US\$50	\$30.000	\$20.000	\$30.000	\$30.000	\$30.000
Cobertura	• Vitacura • Las Condes • La Reina	• Providencia • Vitacura • Las Condes • Lo Barnechea • Sectores de Ñuñoa	• La Reina • Vitacura • Lo Barnechea	• Vitacura • La Reina • Las Condes	• Sectores de Las Condes	• Las Condes
Horario	24/7	24/7	8:00 a 22:00	24/7	24/7	5:00 a 00:30

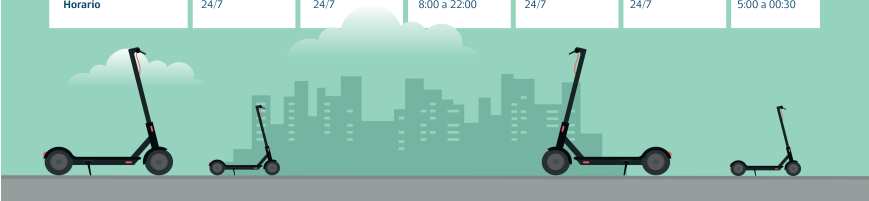


Figura 2.1: Empresas operando en Santiago y tarifas referenciales

Capítulo 3

Estado del Arte

En esta sección se presenta una revisión de la literatura científica y tecnológica relevante para el desarrollo del presente trabajo. Se analizarán diversos estudios y publicaciones que abordan tecnologías clave relacionadas con el objetivo del proyecto, permitiendo comprender el estado actual del conocimiento en el área.

Para ello, se revisarán distintos enfoques y soluciones documentadas en papers y artículos técnicos, con el propósito de identificar tendencias, metodologías y avances recientes en el campo. La revisión incluirá trabajos que han explorado el uso de plataformas IoT, protocolos de comunicación de baja potencia como LoRaWAN y estrategias aplicadas en movilidad compartida y micromovilidad.

El análisis de estas investigaciones proporcionará un marco de referencia sólido, permitiendo contextualizar las soluciones propuestas y justificar la selección de tecnologías en función de su aplicabilidad y desempeño en escenarios similares al del presente estudio.

3.1. Fog Computing Approach for Shared Mobility

En "Fog Computing Approach for Shared Mobility in Smart Cities"[1] se utiliza una arquitectura de *fog computing*. Esto lo que permite es tener el poder computacional de la red más cerca de los nodos. La motivación surge de la preocupación por la

falta de enfoque en aplicaciones que requieran de una baja latencia para operar, sobre todo con el crecimiento de 5G e IoT. La computación en la nube sirve para cumplir una gran variedad de aplicaciones, sin embargo carece de una rápida respuesta, puesto que el poder computacional se encuentra muy lejos del usuario. La arquitectura propuesta se divide en tres niveles o "tiers": *edge tier*, *fog tier* y *cloud tier*. El *edge tier* se compone de los end-users y dispositivos IoT como vehículos eléctricos, sensores, actuadores y smartphones. Los nodos en este nivel generalmente tienen espacio y poder computacional limitado. Por otro lado, el *fog tier* es el intermediario entre el nivel anterior y la nube. Este se modela, comúnmente, como miles de dispositivos que interconectan la red, como estaciones base, routers y gateways, que no solo soportan diferentes tecnologías de comunicación, como WiFi, LoRaWAN y 5G, sino que también tienen las capacidades computacionales para satisfacer las necesidades de los nodos en el *edge tier*. Finalmente, el nivel de la nube representa los centros de datos que tienen un poder computacional y capacidad de almacenamiento muy grandes. Se espera que este nivel pueda proporcionar servicios de computación on-demand con alta disponibilidad y elasticidad. Para este estudio se utilizó un arduino UNO, Dragino LoRa end-node, GPS, y la e-bike en la capa física. El gateway se implementó usando un Raspberry Pi 3bB+ con una antena de 3dBi y la placa IMSTiC880A-SPI. El modelo de movilidad compartida de referencia se compone de distintas capas: capa física, capa de comunicaciones, capa de control y manejo y capa de interfaz. Además, la arquitectura cuenta con una capa de seguridad que cubre a todo el modelo, y con una capa de continuidad de operaciones.

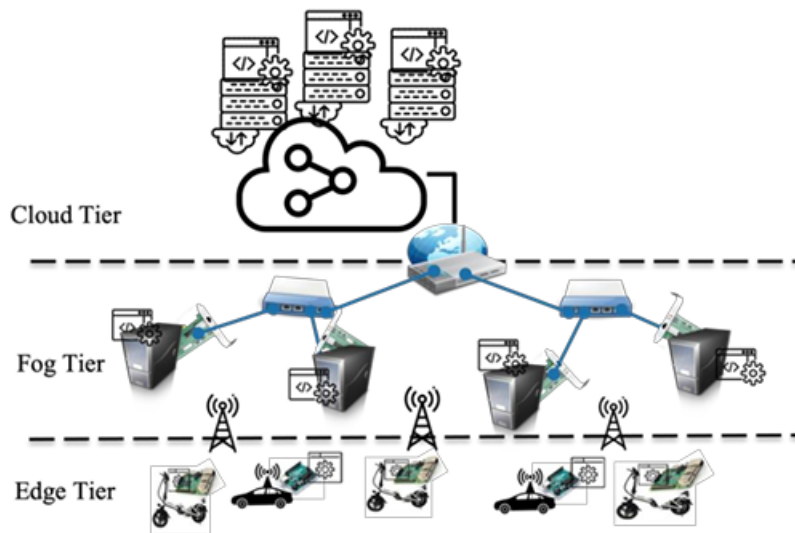


Figura 3.1: Modelo de tres niveles

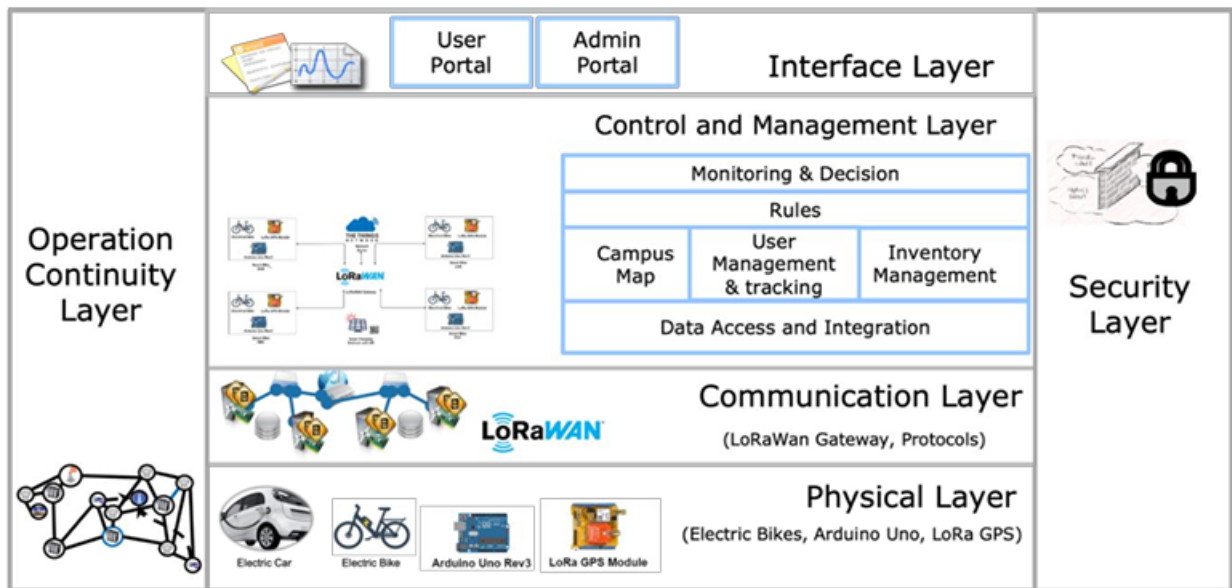


Figura 3.2: Modelo de referencia

Capa física

La capa física es la base de la infraestructura. Esta especifica las entidades físicas que operan dentro del *edge tier*, como vehículos eléctricos, microcontroladores,

módulos GPS. Esta capa especifica, también, los protocolos para permitir la integración con la capa de comunicaciones, y los procesos que permiten a los equipos realizar sus funciones y ayudar a otras capas. Una función clave de esta capa es ejecutar las solicitudes generadas desde la capa de control y manejo.

Capa de comunicaciones

Esta capa trabaja con las tecnologías de comunicación utilizadas para permitir la transmisión e interacción entre el software de la capa de control y la capa física. Esta capa debe soportar comunicación de bajo consumo, por lo que se decidió utilizar LoRaWAN, protocolo que consiste de los elementos presentados en la figura 3.3

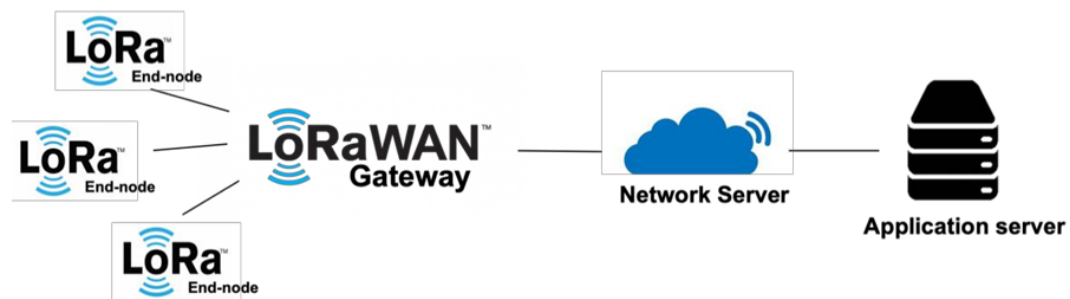


Figura 3.3: Elementos estándar de LoRaWAN

Con la implementación de LoRaWAN, el prototipo de implementación queda de la siguiente manera:

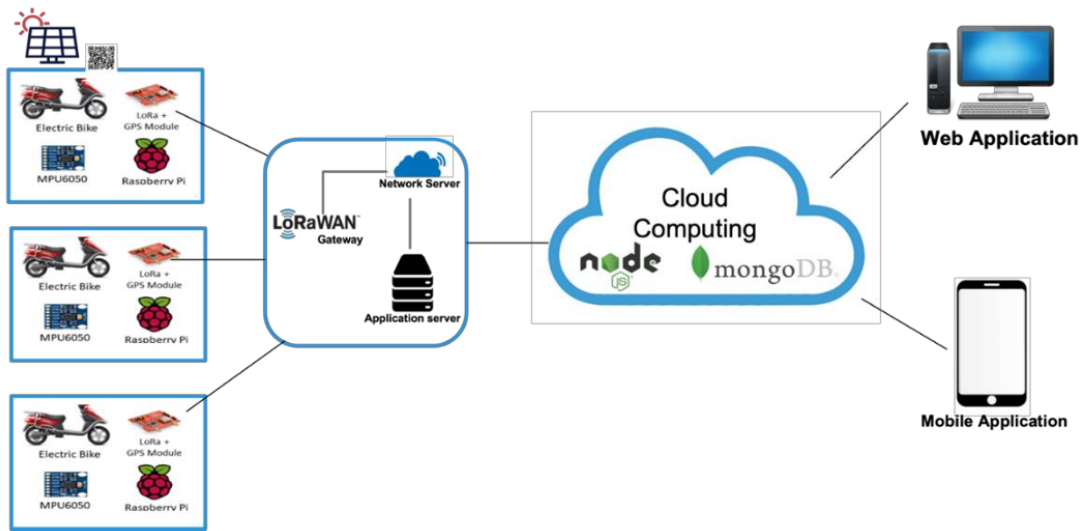


Figura 3.4: Prototipo de implementación propuesto

Capa de control y manejo

Esta capa se compone de distintos módulos que permiten el manejo de operaciones de los vehículos de movilidad compartida. Uno de los requisitos de esta capa es permitir la transformación de los vehículos eléctricos a una entidad por software que pueda ser controlada dentro de la misma capa. Así, uno de los módulos es el de acceso de datos e integración, manejo y seguimiento de usuarios, mapa del campus, entre otros.

Como resultado, el modelo propuesto demuestra una alta capacidad para manejar solicitudes de usuarios, además de una latencia aceptable para las aplicaciones requeridas de unos 270 ms en promedio.

3.2. Sistema de transporte público usando LoRa

En "Proof of Concept of an IoT-Based Public Vehicle Tracking System, Using LoRa (Long Range) and Intelligent Transportation System (ITS) Services", se propone un sistema de transporte inteligente basado en IoT para una ciudad. Este está pensado para mejorar el transporte público de una ciudad latinoamericana, y como tal, los

vehículos de interés son los buses y microbuses. De cualquier manera, la arquitectura propuesta para la red es la siguiente:

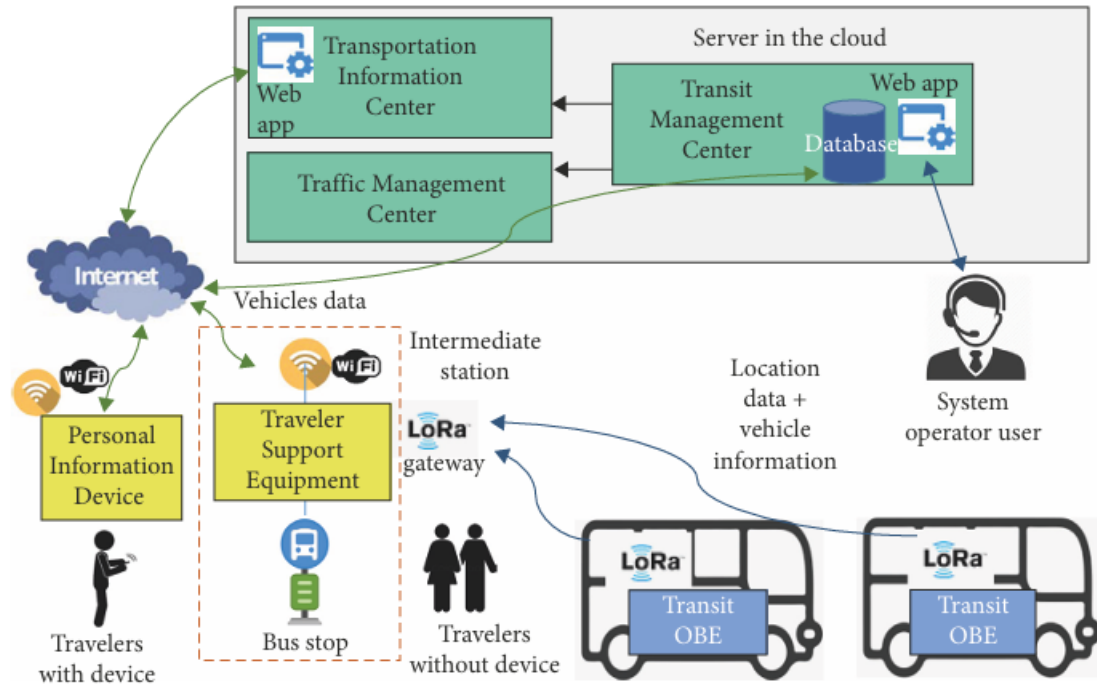


Figura 3.5: Esquema de red de sistema de transporte inteligente

En este proyecto, se utiliza la tecnología LoRaWAN como protocolo de comunicación, pero el gateway está conectado a internet y se comunica con un servidor en la nube. Desde este, un operador se encarga de actualizar la información de los vehículos y los usuarios. En este experimento no se considera tan importante la latencia y los tiempos de respuesta, puesto que lo más importante es conocer la ubicación los buses, por lo que la frecuencia de transmisión es de 1 mensaje cada 3 segundos. Así, se justifica más la arquitectura implementada. Por otro lado, los experimentos realizados para corroborar el funcionamiento del sistema se enfocan más en demostrar el funcionamiento de la tecnología LoRaWAN para transmitir datos de manera confiable. Los resultados parecen no ser un 100 por ciento satisfactorios, puesto que en el escenario más real que se plantean, con los módulos en movimiento y distancias más grandes, la pérdida de datos es de alrededor de un 28 por ciento en escenarios en obstáculos

entre los nodos y el gateway. Esto plantea la importancia de tener un ambiente que favorezca a la tecnología.

Para este estudio se utilizó el siguiente equipo: microcontrolador HiLetgo ESP32 LoRa 0.96 inch OLED display, GPS Ublox Neo 6M, gateway Dragino LG01.

3.3. Sobre el uso de LoRaWAN en aplicaciones IoT

En "On the use of LoRaWAN for Mobile Internet Of Things: The Impact of Mobility" se plantea sobre el uso de LoRaWAN en aplicaciones IoT con movilidad. Esto viene al caso en este estudio, ya que, como se vio en [3], LoRaWAN puede tener dificultades en ciertos ambientes. El trabajo consiste en probar intensivamente el uso de LoRaWAN en entornos IoT con distintos modelos de movilidad en simulaciones. Para esto, se tomaron dos escenarios, uno en un malla de 2000x2000 (en el entorno de simulación), que se considera como un espacio grande, y otro en una malla de 1000x1000, considerado un espacio pequeño. En el primero se ubicaron cuatro gateways estáticos en forma de cuadrado, y en el segundo se ubicaron dos gateways. Para cada escenario se testeó con distintas cantidades de end-nodes, entre 250 y 2500. Además, se utilizaron tres modelos de movilidad para simular el movimiento de los nodos: *Random Waypoint*, *Gauss-Markov* y estacionario. Los resultados para ambos escenarios con distinta cantidad de nodos y velocidades se pueden ver en las figuras 3.6 y 3.7. En estas se utiliza la métrica de razón de entrega de paquetes, que toma la relación entre la cantidad de paquetes recibidos exitosamente por el servidor y los paquetes enviados por todos los nodos de la red.

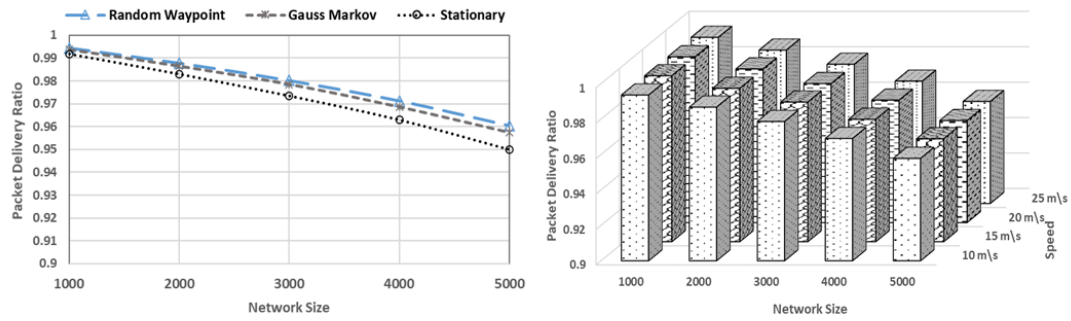


Figura 3.6: Razón de entrega de paquetes: a)razón de entrega de paquetes para una red con 20B de payload y rapidez de los nodos de 5 m/s. b)razón de entrega de paquetes para una red de 2000x2000 usando el modelo de movilidad de Gauss-Markov

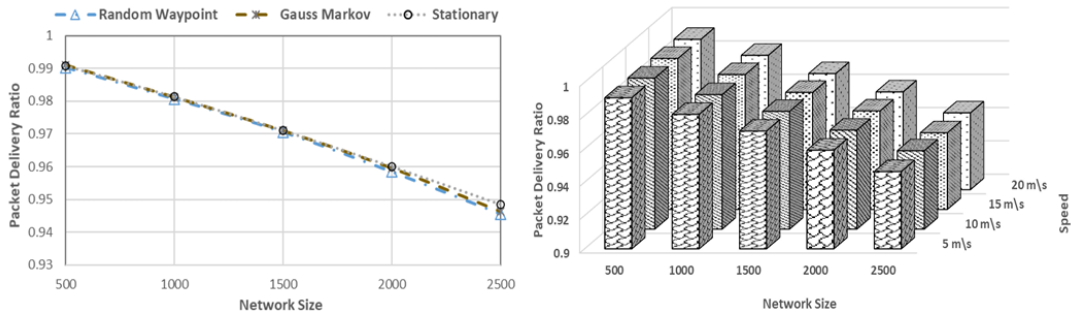


Figura 3.7: Razón de entrega de paquetes: a)razón de entrega de paquetes para una red de 1000x1000, 100B de payload y rapidez de los nodos de 25 m/s. b) razón de entrega de paquetes para una red de 1000x1000 usando el modelo de movilidad random waypoint.

Otra métrica utilizada para la evaluación del sistema es el consumo de energía. Este se puede ver en las figuras 3.8 y 3.9.

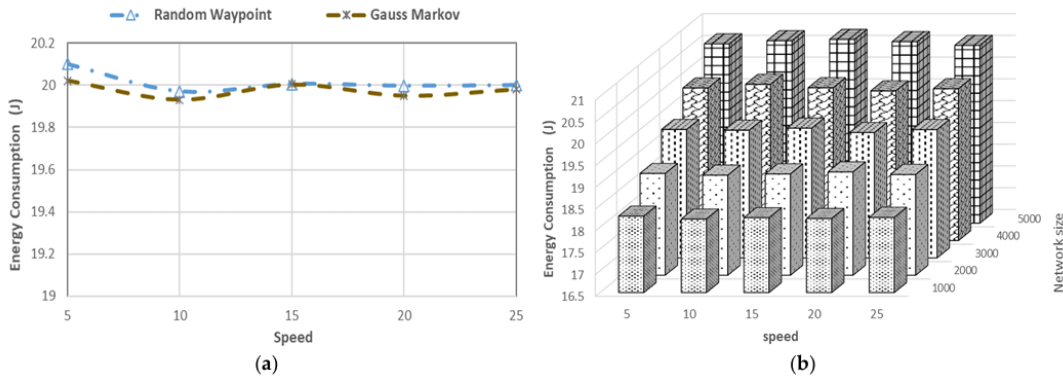


Figura 3.8: a) Consumo de energía promedio por dispositivo para una red de 2000x2000, 4000 dispositivos y 60 B de payload a distintas velocidades en m/s. b) Consumo de energía promedio por dispositivo para una red de 2000x2000 usando el modelo de movilidad random waypoint, con 20 B de payload a distintas velocidades en m/s.

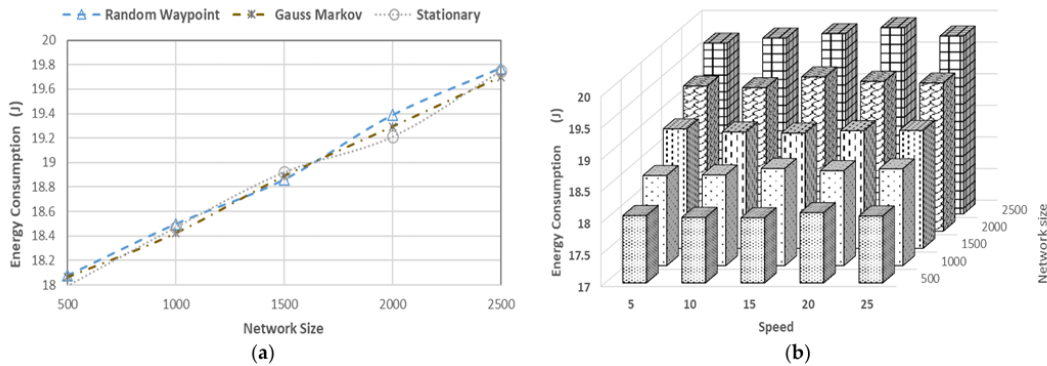


Figura 3.9: a) Consumo de energía promedio por dispositivo para una red de 1000x1000, 20B de payload, y 5 m/s de velocidad en redes de distinto tamaño. b) Consumo de energía promedio por dispositivo para una red de 1000x1000 usando el modelo de movilidad Gauss-Markov, con 20 B de payload para diferentes velocidades en m/s.

Como se puede ver en los resultados del trabajo, LoRaWAN puede presentar buenos resultados para un sistema de transmisión de datos de alto rango y bajo consumo. Se puede ver que hasta en altas velocidades y con una red de muchos usuarios, es posible lograr buenas métricas. Por ende, el uso de LoRaWAN como la principal tecnología para tareas de IoT con o sin movilidad se ve justificado dado su bajo consumo y sus capacidades.

3.4. Sobre las capacidades de LoRa

En “On the feasibility of mobile sensing and tracking applications based on LPWAN”[4] se realiza un experimento sobre IoT móvil y LoRaWAN. En la introducción se plantea como LoRaWAN es el estándar para aplicaciones IoT estáticas, más, al igual que en [3], surge la necesidad de comprobar su funcionamiento para aplicaciones móviles. La arquitectura utilizada es la que se muestra en la figura 3.10. Esta se basa en una conexión directa entre los nodos y el servidor de aplicación, a través del gateway LoRa. Para los nodos se usó un IOT2040 conectado con un microchip LoRaWAN dev kit (DM164138 con LoRa modem RN2483).

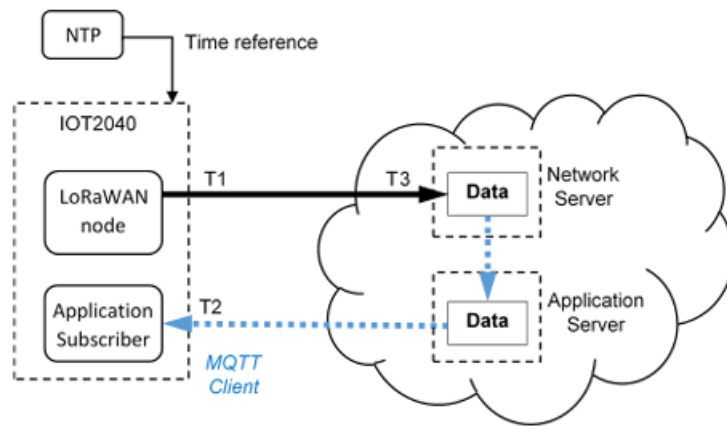


Figura 3.10: Esquema de comunicación usando LoRaWAN

El experimento se realizó en un área de $3,3 \text{ km}^2$ (ver Fig. 3.11). Las métricas de interés fueron dos. Una relacionada con el delay de la red, es decir el tiempo que demora un paquete desde salir de un nodo hasta que llegue al servidor. Esto para ver cómo afecta la movilidad a la parte más relacionada al hardware del sistema. La otra métrica está relacionada al delay de aplicación. Para este experimento se tomaron en cuenta distintas conexiones de internet. Los resultados proponen que el delay relacionado a la radio LoRaWAN se mantiene constante para nodos estáticos y móviles (unos 250 ms), mientras que la variabilidad se introduce según la conexión a internet que se tenga. Sin embargo, si se mantiene una conexión a internet estable, el delay se mantie-

ne alrededor de 300 ms, lo cual puede ser aceptable dependiendo de las necesidades de responsividad que se tengan.

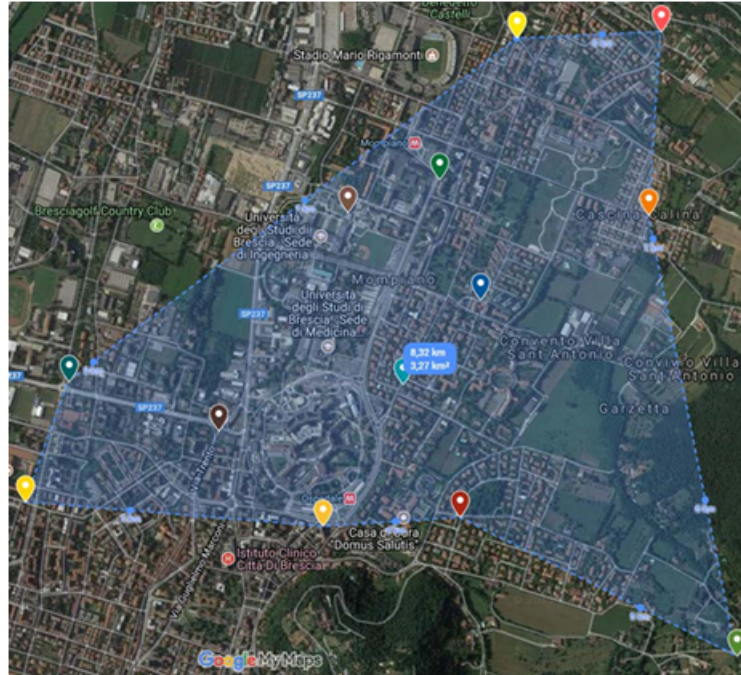


Figura 3.11: Área de cobertura de de LoRaWAN

3.5. Sistema de transporte inteligente usando 5G

En "Next generation ITS implementation aspects in 5G wireless communication network"[5] se propone un sistema de transporte inteligente usando comunicaciones inalámbricas con 5G. El enfoque de esta propuesta es crear un sistema de transporte inteligente con comunicación entre vehículos y administrado desde un servidor en la nube. Por ende, la propuesta se escapa de las necesidades y requisitos que se tienen. Sin embargo, vale la pena explorar otras propuestas. La arquitectura planteada es la que se ve en la figura 3.12. Esta oferta es interesante para ciudades futuras en donde el transporte pueda estar cien por ciento sincronizado, aprovechando los avances en comunicaciones.

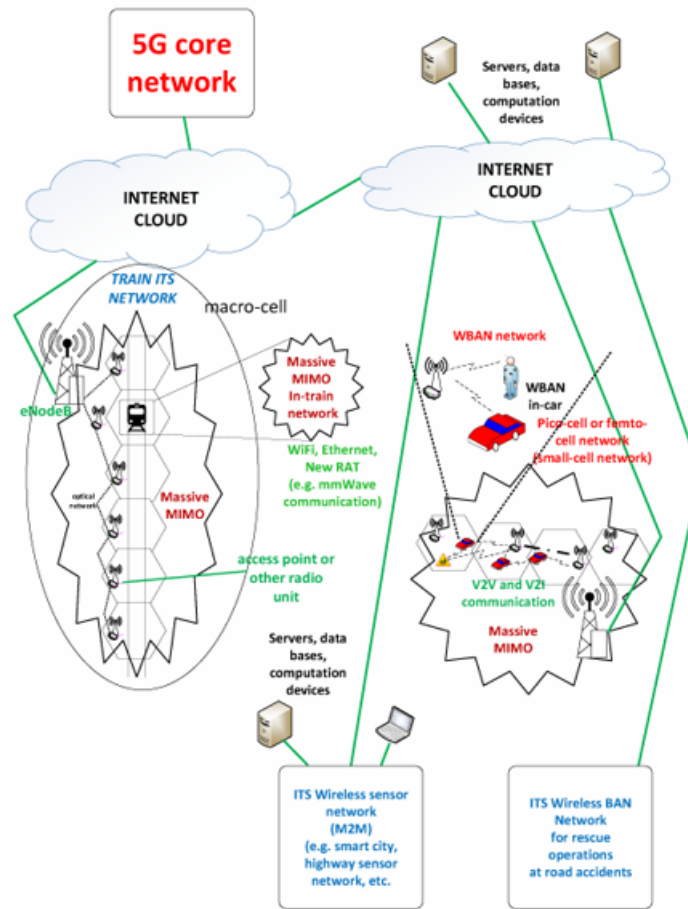


Figura 3.12: Arquitectura general de ITS implementado en red 5G

3.6. Monitoreo remoto de e-bikes

En "IoT Enabled Electronic Security and Remote Monitoring Solution For Electric Bikes"[6] se propone un sistema para monitorear bicicletas eléctricas usando IoT". La arquitectura propuesta se ve en la figura 3.13

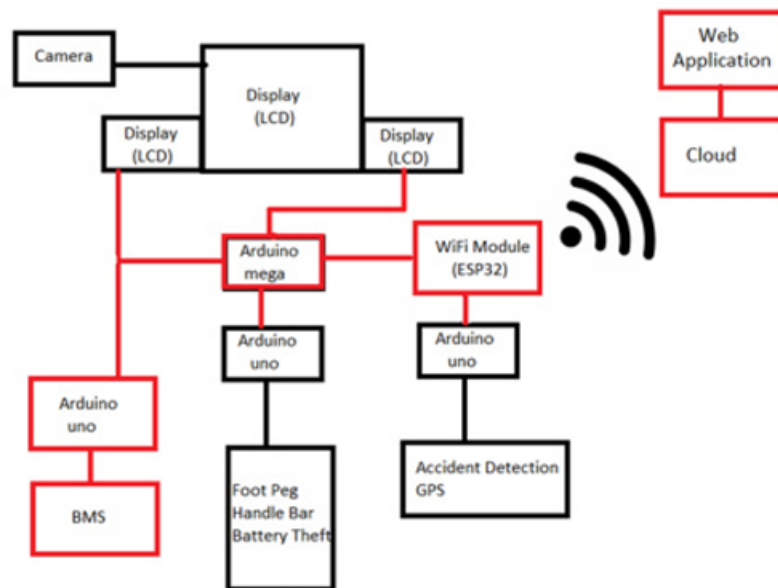


Figura 3.13: Diagrama de flujo IoT

El trabajo realizado busca permitir a usuarios monitorear aspectos de seguridad y mantenimiento de sus bicicletas electrónicas desde una aplicación web. Las características propuestas incluyen detección de choque, sistemas de predicción, sistemas de robo de batería y del vehículo y manejo de batería. El trabajo incluye, además, el diseño de las distintas interfaces para las aplicaciones en las que los usuarios pueden revisar los datos transmitidos. Para el desarrollo de esta, se usa Arduino IoT Cloud. Por otro lado, el protocolo de comunicación utilizado es MQTT. La transmisión de datos se hace mediante WiFi, utilizando el módulo ESP32 de Arduino.

El concepto propuesto pretende ofrecer distintas capacidades de monitoreo a bicicletas electrónicas, más no entra en detalle sobre temas como consumo energético, delay de transmisión, o el rango de comunicación.

3.7. Comparación

Habiendo revisado algunos trabajos relacionados, se procede a elaborar una tabla comparativa de cada paper en la que se presenta el tipo de sistema en construido, OBE (on board equipment), sistema de comunicación y tiempos de respuesta.

Proyecto	Tipo de vehículo	Tipo de arquitectura	OBE	Gateway	Sistema de comunicacion	Tiempo de respuesta/PDR/Consumo
Fog computing para movilidad compartida	e-bikes	Fog computing	Arduino UNO, Dragino LoRa + GPS end-node	Raspberry Pi 3bB + IMSTIC880A-SPI	LoRaWAN	270 ms
Transporte público basado en IoT usando LoRa	Buses	Cloud computing	HiLetgo ESP32 LoRa 0.96 inch OLED display, GPS Ublox Neo 6M	Dragino LG01	LoRaWAN	No especificado
Simulación de IoT móvil con LoRa	UAVs	Cloud computing	Simulación OMNET++	Simulación OMNET++	LoRaWAN	Ver Fig. 3.8 y 3.9
Sobre las capacidades de LoRaWAN	No especificado	Cloud computing	DM164138 con LoRa modem RN2483	LoRa Gateway	LoRaWAN	Network delay: 260 ms (promedio). App delay: 10-250 ms según movilidad
Implementación de ITS con tecnología 5G	Buses	Cloud computing con infraestructura 5G	No especificado	Estaciones base	5G	No especificado
Monitoreo remoto de e-bikes	e-bike	Cloud computing	Ver Fig. 3.13	Gateway WiFi	WiFi	No especificado

Figura 3.14: Resumen de proyectos con diferentes tecnologías de comunicación.

Capítulo 4

Arquitectura propuesta

Antes de declarar la arquitectura final, se muestran distintas tecnologías y servicios disponibles que podrían ser usados para cada capa del proyecto. Estas se pueden ver en la figura 4.1. Luego, se realiza un desglose de cada capa.

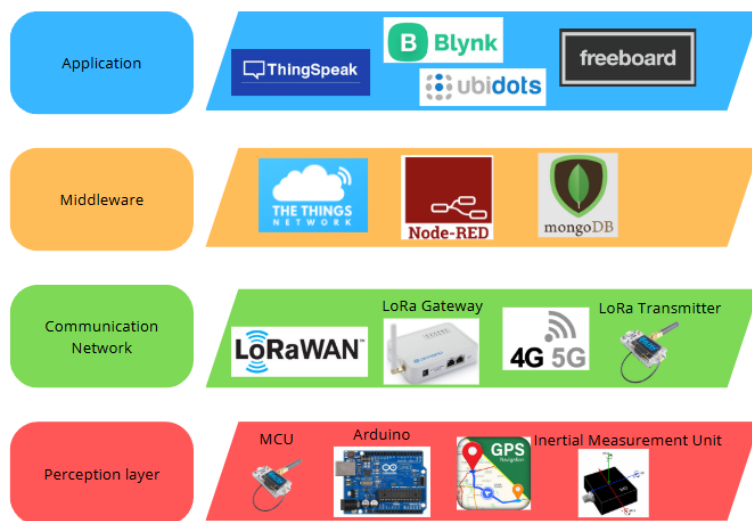


Figura 4.1: Tecnologías por capa

4.1. Capa de percepción

La capa de percepción en un sistema IoT es la encargada de la interacción directa con el entorno físico a través de sensores y actuadores. Su función principal es cap-

turar datos del mundo real, como temperatura, humedad, presión, nivel de luz, o más relevante para este caso, aceleración y posición global. Estos datos son recolectados por dispositivos como microcontroladores, sensores inteligentes y módulos de comunicación, que los procesan de manera básica antes de enviarlos a las capas superiores del sistema. Además, en algunos casos, la capa de percepción también puede incluir actuadores, que permiten que el sistema responda al entorno de manera automatizada, como encender un ventilador, ajustar una válvula o mover un motor. En general, esta capa es fundamental para la operatividad de cualquier sistema IoT, ya que proporciona la información esencial para el monitoreo, control y toma de decisiones en tiempo real.

Descripción de componentes

- **Microcontroladores:** Son el núcleo de la capa de percepción, encargados de procesar los datos capturados por los sensores y gestionar la comunicación con otras capas del sistema IoT. Ejemplos comunes incluyen ESP32, STM32 y Arduino.
- **Sensores:** Dispositivos que detectan y miden variables físicas del entorno, como temperatura, humedad, presión o luminosidad, convirtiéndolas en señales eléctricas que pueden ser interpretadas por el microcontrolador.
- **Unidad de medición de inercia (IMU):** Sensor compuesto por acelerómetros y giroscopios que mide la aceleración y orientación en múltiples ejes, permitiendo detectar movimientos, inclinaciones y vibraciones. Ejemplo: MPU9250.
- **GPS (Sistema de Posicionamiento Global):** Dispositivo que determina la ubicación geográfica en tiempo real mediante señales satelitales, proporcionando coordenadas de latitud, longitud y, en algunos casos, altitud y velocidad.

4.2. Capa de Comunicaciones

La capa de comunicaciones en un sistema IoT es la encargada de transmitir los datos recopilados por la capa de percepción hacia servidores, plataformas en la nube o dispositivos de control. Para ello, utiliza diferentes tecnologías de comunicación inalámbrica y protocolos de red que garantizan una transmisión eficiente y segura de la información. Esta capa es fundamental para la conectividad del sistema, ya que permite la integración de múltiples dispositivos distribuidos en distintas ubicaciones. Dependiendo de los requerimientos del proyecto, se pueden utilizar tecnologías como WiFi, LoRa, Bluetooth, Zigbee o redes celulares (4G/5G, NB-IoT).

Descripción de componentes

- **LoRa Gateway:** Dispositivo que actúa como un puente entre los nodos LoRa y la infraestructura de red central, permitiendo la transmisión de datos a largas distancias con bajo consumo de energía. Recibe señales de múltiples dispositivos IoT y las envía a servidores en la nube mediante conexiones a internet (Ethernet, WiFi o celular).
- **Heltec WiFi LoRa 32:** Microcontrolador que combina conectividad WiFi y LoRa, ideal para proyectos IoT de largo alcance y bajo consumo energético. Permite la comunicación inalámbrica en redes LoRaWAN o punto a punto, facilitando la integración de sensores y actuadores en entornos remotos.

4.3. Capa de Middleware

La capa de middleware es una capa intermedia entre los dispositivos IoT (sensores, actuadores, microcontroladores) y las aplicaciones finales que consumen los datos. Su función principal es facilitar la comunicación, interoperabilidad y gestión de los dispositivos de manera eficiente.

Descripción de componentes/servicios

- **The Things Network (TTN):** Plataforma basada en LoRaWAN que permite la conexión, gestión y transmisión de datos desde dispositivos IoT a la nube. Facilita la comunicación de largo alcance con bajo consumo energético y ofrece herramientas para la integración con otros servicios en la nube.
- **ThingSpeak:** Servicio en la nube que permite almacenar, procesar y visualizar datos IoT en tiempo real. Soporta la integración con MATLAB para análisis avanzados y ofrece APIs para conectar dispositivos mediante HTTP o MQTT.
- **Node-RED:** Herramienta de programación visual basada en flujo que permite conectar, procesar y transformar datos IoT sin necesidad de escribir código extenso. Es ideal para la automatización, integración de dispositivos y procesamiento de datos en tiempo real.
- **MongoDB:** Base de datos NoSQL optimizada para el almacenamiento de datos IoT, permitiendo el manejo eficiente de grandes volúmenes de información estructurada y no estructurada. Su flexibilidad y escalabilidad la hacen ideal para proyectos con múltiples fuentes de datos en tiempo real.

4.4. Capa de aplicación

La capa de aplicación en un sistema IoT es la encargada de proporcionar interfaces y herramientas que permiten a los usuarios interactuar con los datos generados por los dispositivos conectados. Esta capa traduce la información procesada en la capa de middleware en gráficos, reportes, alertas o acciones automatizadas, facilitando el monitoreo y la toma de decisiones. Dependiendo del proyecto, puede incluir aplicaciones web, móviles o dashboards personalizados que visualizan datos en tiempo real y permiten controlar dispositivos de forma remota.

Descripción de componentes/servicios

- **ThingSpeak:** Plataforma en la nube que permite visualizar y analizar datos IoT en tiempo real. Ofrece gráficos personalizables, almacenamiento de datos y herramientas de análisis mediante MATLAB.
- **Ubidots:** Servicio IoT enfocado en la visualización y análisis de datos con dashboards interactivos. Soporta protocolos como MQTT y HTTP, permitiendo la integración con múltiples dispositivos y servicios.
- **Freeboard:** Plataforma open-source para la creación de dashboards livianos y personalizables. Se integra con APIs y bases de datos externas para mostrar datos en tiempo real mediante gráficos y widgets interactivos.
- **Blynk:** Herramienta diseñada para la gestión y visualización de proyectos IoT, con una app móvil y una interfaz web que permite monitorear sensores y controlar dispositivos de forma remota sin necesidad de programar interfaces desde cero.

4.5. Requisitos del sistema

4.5.1. Requisitos funcionales

Los requisitos funcionales del sistema son aquellos que definen su comportamiento. Es lo que espera el usuario final (**end user** o **eu**) que haga la aplicación. Así, esta serie de requisitos interpretan y dictan la funcionalidad de la plataforma. Dicho esto, los requisitos funcionales se presentan a continuación:

1. **Conectividad.** El sistema debe ser capaz de conectarse a internet y/o a la red de trabajo para poder enviar los datos recopilados.
2. **Recopilación de información.** El sistema debe ser capaz de recopilar la información necesaria para el correcto funcionamiento de la aplicación. Esto incluye

la velocidad, orientación y posición de los nodos.

3. **Interfaz de usuario (UI).** La aplicación debe contar con una interfaz de usuario tipo dashboard que despliegue la información recopilada por la capa de percepción. Esta debe incluir lecturas de velocidad, posición, roll, pitch, etc.

4.6. Requisitos no funcionales

Los requisitos no funcionales definen características de valor para el usuario que no se traducen directamente en funcionalidades específicas. A continuación se muestra una lista de requisitos no funcionales.

1. Interfaz sencilla y accesible para el usuario.
2. El sistema debe tener una disponibilidad del 99 %.
3. El sistema debe ser seguro para el usuario.

4.7. Tecnologías utilizadas

A continuación se presenta una lista con tecnologías y piezas de software/hardware que se requieren para el desarrollo del sistema.

Hardware

- Heltec WiFi Lora 32 v2/v3
- Dragino LG308-EC25
- Computador/Notebook
- Android Smartphone
- GPS, Unidad de medición de inercia

Software

- Python 3.8
- C
- Matlab
- Arduino IDE
- Visual Studio Code

Perfil de usuario

En la siguiente tabla (4.1) se presenta el perfil del usuario objetivo de la plataforma

Perfil	Ocupación	Rango etario	Características psicológicas
Administrador de aplicación	Administrador. Se asume que existe una persona que vigila el comportamiento de los vehículos diariamente.	25 - 60 años	Persona que busca realizar su trabajo día a día y requiere de la información provista por los EV para su monitoreo.

Cuadro 4.1: Perfil de usuario

4.8. Propuesta de solución

EL objetivo de este trabajo de título es desarrollar una plataforma para administradores de una aplicación de movilidad compartida, que permita a los administradores un monitoreo constante y robusto de los scooters eléctricos de la empresa.

Con el fin de ofrecer distintas opciones, se desarrollarán dos soluciones. La primera consta de un sistema IoT basado en un MCU y sensores dedicados que utiliza LoRa para las comunicaciones. La segunda se basa en smartphones Android y los sensores que estos incluyen. Las comunicaciones en este caso se realizan mediante 4G.

Para la capa de aplicación se utilizará Node-Red y ThingSpeak. Estas plataformas permiten crear interfaces de usuario tipo dashboard sencillas que despliegan la información necesaria para los administradores. Además, ThingSpeak permite una fácil inclusión con sistemas IoT, funcionando en base a peticiones HTTP, pero también se puede utilizar en conjunto con The Things Network para aplicaciones basadas en LoRa.

Para el Middleware, se utilizará Node-Red. Esta aplicación permite definir y configurar flujos para sistemas IoT. En conjunto con el protocolo MQTT, y el servidor correspondiente, es posible centralizar todos los dispositivos pertenecientes a una red del internet de las cosas.

Para la capa de comunicaciones se usará, en primera instancia, LoRa. Este protocolo de comunicaciones, desarrollado para IoT, permite la comunicación a larga distancia con un consumo mínimo. Para su implementación, el módulo Heltec Wifi LoRa 32 incluye funcionalidades para transmitir y recibir paquetes LoRa. En conjunto con este, el gateway Dragino LG308-EC25 permite la recepción y el reenvío de los paquetes directamente a un servidor MQTT.

Para la solución basada en Android, las comunicaciones se realizarán mediante 4G utilizando la API de ThingSpeak para enviar los datos.

Para la capa de percepción, se usará un módulo Heltec Wifi LoRa 32 v2, en conjunto con un GPS ublox Neo-7m y una IMU MPU9250.

Con estos servicios y dispositivos, se espera construir una plataforma que permita a usuarios obtener una mayor seguridad a la hora de usar y monitorear EV, lo que ayudará al desarrollo e implementación de ciudades más eficientes.

Capítulo 5

Diseño de prototipos y resultados

En esta sección se presentará el desarrollo e implementación de la plataforma IoT para movilidad compartida. Recordemos que el objetivo es implementar una plataforma para impulsar la movilidad compartida y el uso de EVs. Con el fin de cumplir esta tarea, se necesita como mínimo hacer la recopilación, transmisión, procesamiento y despliegue de la información de los vehículos. Así, el trabajo consiste en implementar los *endnodes* (microcontroladores en conjunto con sensores) y generar la conexión con los servidores utilizados, para finalmente hacer el despliegue en la interfaz de usuario o capa de aplicación. Para esto se diseñaron tres soluciones o escenarios. El primero consta de una conexión nodo a nodo (transmisor - receptor) mediante protocolo LoRa. En este escenario, el nodo receptor reenvía los datos a un servidor MQTT ubicado en la misma red local en la que se encuentra. El segundo escenario es una conexión nodo - gateway con comunicación LoRaWAN. En este caso, el gateway también reenvía la información a un servidor MQTT en la red local. El tercer escenario es una aplicación de Android que utiliza los sensores del teléfono del usuario para obtener los valores de ubicación, velocidad y orientación y los reenvía a un canal de ThingSpeak. En la tabla 5.1 se muestran las distintas soluciones y sus características. Luego, las figuras 5.1, 5.2, 5.3 muestran un diagrama para cada solución.

Escenario	Transmisor	Receptor	Hardware	Comunicaciones	Middleware	Aplicación
1	Heltec Wi-Fi LoRa 32 V2	Heltec Wi-Fi LoRa 32 V3	MCU + GPS (ublox Neo 7M) + IMU (MPU9250)	LoRa	Node-Red/Broker MQTT	Node-Red Dashboard
2	Heltec Wi-Fi LoRa 32 V2	Dragino LG308	MCU + GPS (ublox Neo 7M) + IMU (MPU9250)	LoRa	Node-Red/Broker MQTT	Node-Red Dashboard
3	Android Smartphone	IP Gateway	Samsung Galaxy S9+	4G	ThingSpeak/Node-Red	Node-Red Dashboard

Cuadro 5.1: Soluciones implementadas

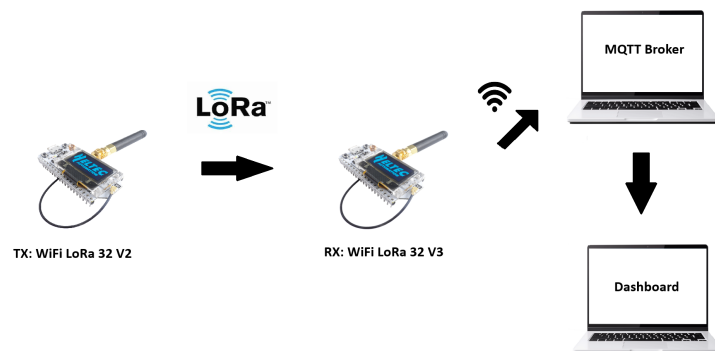


Figura 5.1: Escenario 1

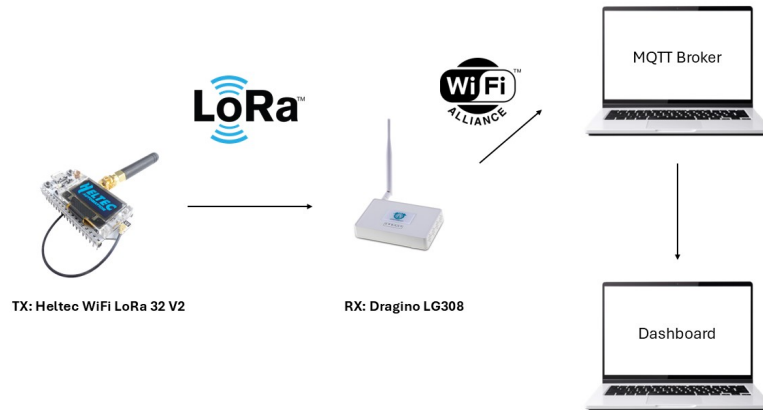


Figura 5.2: Escenario 2

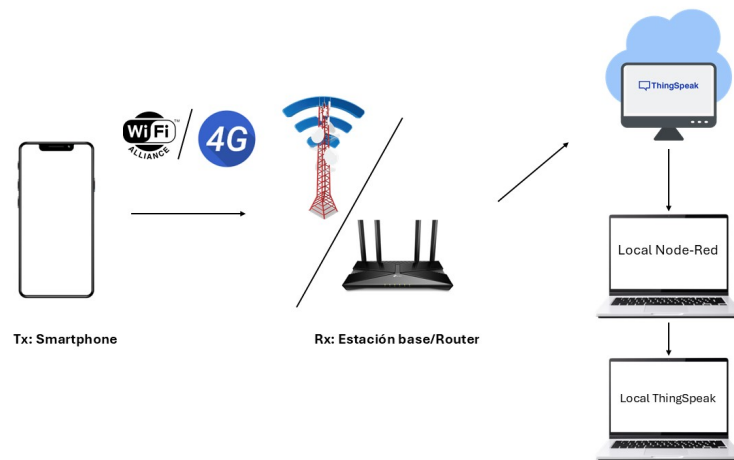


Figura 5.3: Escenario 3

- **El escenario 1** se compone de un módulo Heltec WiFi LoRa 32 V2 como transmisor, en conjunto con un GPS ublox Neo-7m y una IMU MPU9250. Como receptor se utiliza un módulo Heltec WiFi LoRa 32 V3. Este se conecta a la red WiFi local, y reenvía los datos recibidos a un broker MQTT montado en la misma red. Utilizando Node-red, se procesan los datos y se despliegan en un dashboard al que se puede acceder desde cualquier dispositivo en la LAN.
- **El escenario 2** utiliza la misma configuración para el transmisor. Como recep-

tor se utiliza en Gateway LoRaWAN Dragino LG308. Este se configura para reenviar los datos al broker MQTT y hacer el procesamiento y despliegue de la información.

- **El escenario 3** se basa que cada persona que busque arrendar un EV, lo hace con su *smartphone*, el cual, en la amplia mayoría de los casos, cuenta con los sensores necesarios para la recopilación de la información. Se desarrolló una aplicación para Android que recopila la información y la envía por internet a la plataforma de **Thingspeak**. Luego, desde node-red se recopila la información mediante solicitudes http, se procesa la información y se hace el despliegue.

5.1. Hardware

A continuación, se describe el hardware utilizado para la implementación de cada solución.

5.1.1. LoRa Node

Para los escenarios 1 y 2, el nodo LoRa se compone de un módulo **Heltec WiFi LoRa 32 V2** conectado a un GPS y una IMU. La MCU se seleccionó por su disponibilidad en el laboratorio, y por las funcionalidades que ofrece, además de la facilidad de uso. Gracias a su entorno de programación (Arduino IDE) y la biblioteca *Heltec ESP32 Dev Boards*, que ofrece capacidades de LoRa, estos módulos resultan muy útiles para distintas aplicaciones IoT. En la figura 5.4 se puede ver el diagrama de conexión para los escenarios 1 y 2.

Por otro lado, para el escenario 1 se utilizó un módulo Heltec WiFi LoRa 32 V3 como receptor. Para este caso, funciona igual que la versión anterior. La figura 5.5 muestra el diagrama de conexión del nodo receptor.

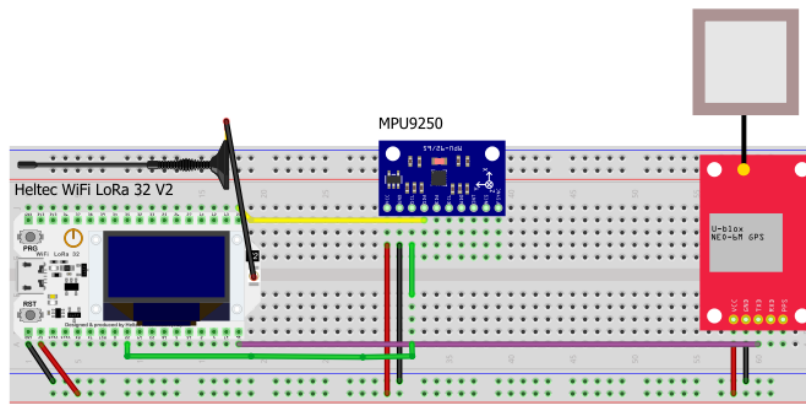


Figura 5.4: Diagrama de conexiones Tx

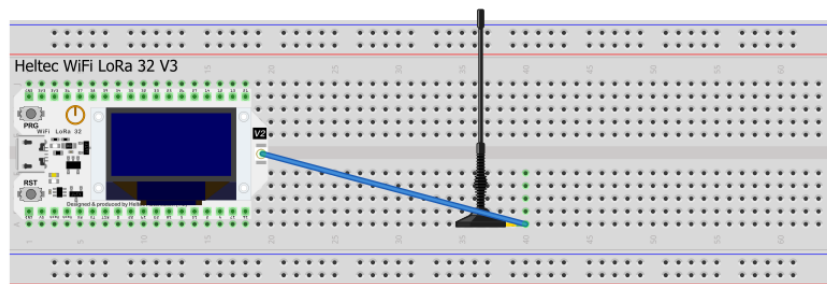


Figura 5.5: Diagrama de conexiones Rx

5.1.2. Gateway

En el escenario 2 se utiliza un gateway Dragino LG308 como receptor de los paquetes LoRa. Se eligió este modelo debido a que estaba presente en el laboratorio. Además de esto, es fácil de configurar y ofrece varias funcionalidades, como encriptación/desencriptación de mensajes, reenvío de paquetes, fácil conectividad con plataformas como *The Things Network* o brokers MQTT.

Para este trabajo, el gateway se comunica con un servidor MQTT hospedado dentro de la red local. El nodo envía los paquetes encriptados con ABP, luego, el gateway los desencripta y los publica en el tópico "dragino-gw/output".



Figura 5.6: Dragino LG308

5.2. Software

En esta sección se describe cómo se configuraron los dispositivos utilizados en cada escenario. Esto incluye descripción del software, configuraciones, etc.

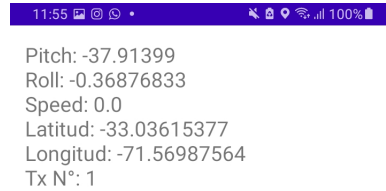
5.2.1. Nodo transmisor

Los nodos transmisores cumplen casi la misma función en ambos casos. Mientras que en el escenario 1 no se utiliza el protocolo de comunicaciones LoRaWAN, en el escenario 2 si se usa. Así, el nodo del escenario 1 no incluye encriptación de datos, mientras que el escenario 2 si. El formato de los mensajes enviados por el nodo transmisor en el escenario el escenario 1 y 2 es "*latitud,longitud,rapidez,roll,pitch,txnumber*", lo que corresponde a 50 bytes de datos. Sin embargo, en el escenario 2 se añade el encabezado, lo que corresponde a 12 bytes más, por lo que el tamaño total de los paquetes es de 62 bytes. En el escenario 3 se utiliza la api de ThingSpeak para hacer solicitudes http al canal creado para la aplicación y escribir en cada campo los datos correspondientes. Para esto, se usa la siguiente url: `$thingSpeakUrl?api_key=$apiKey&field1=$latitude&field2=$longitude&field3=$pitch&field4=$roll&field5=$speed&field6=$txNumber`.

En el anexo se encuentran los códigos enteros para cada dispositivo.

1. Para el escenario 1 se utilizó la biblioteca "LoRaWAN_APP.h" incluida en las bibliotecas de "Heltec ESP32 Dev-boards". Esta permite configurar fácilmente el uso de la radio LoRa de la MCU, incluyendo la banda de frecuencia a utilizar, el *Spreading Factor* (SF), BW, coding rate, etc. Para el manejo de sensores se utilizó la biblioteca FastIMU para la unidad de medición de inercia. El firmware del dispositivo realiza lecturas de la IMU por I2C y del GPS por puerto serial. Cuando tiene ambos valores listos, realiza la transmisión cada 1 segundo.
2. Para el escenario 2 se utilizó la biblioteca "lmic.h" de IBM, en vez de la provista por Heltec. Esta es una biblioteca optimizada para el uso de LoRaWAN, y ofrece compatibilidad con múltiples bandas de frecuencia, selección de Data Rate, autenticación y encriptación, y gestión de canales y adaptabilidad. La implementación de los sensores es igual al escenario 1.
3. Para el escenario 3 se desarrolló una aplicación de Android en **Android Studio**. La gestión de los sensores de movimiento, rotación y localización se hace mediante las APIs de Google. En este caso, se usó la API de nivel 26, que funciona con el 95.6 % de los dispositivos Android. La aplicación recopila los valores de orientación y ubicación del teléfono, los muestra en pantalla y luego los envía por internet a ThingSpeak usando la llave API provista por la plataforma para escribir en el canal creado. La figura 5.7 muestra la interfaz de usuario de la aplicación corriendo en un teléfono Samsung Galaxy S9+. Para la localización del dispositivo se usa una mezcla de proveedores. Dependiendo de la disponibilidad esta se obtiene mediante GPS, o mediante redes de internet. Esto ofrece una buena relación entre precisión y rapidez de actualización. En la versión gratis de ThingSpeak, se pueden hacer publicaciones cada 15 segundos, por lo que la aplicación está configurada para hacerlo en ese intervalo. Para poder actualizar con mayor frecuencia, y que la aplicación sea más útil aun, se debe pagar

una suscripción, cuyo precio depende de la cantidad de mensajes que se quieran realizar por hora.

A screenshot of an Android application interface. At the top, there is a status bar with the time 11:55 and various icons. Below the status bar, the application displays the following sensor data: Pitch: -37.91399, Roll: -0.36876833, Speed: 0.0, Latitud: -33.03615377, Longitud: -71.56987564, and Tx N°: 1.

```
Pitch: -37.91399
Roll: -0.36876833
Speed: 0.0
Latitud: -33.03615377
Longitud: -71.56987564
Tx N°: 1
```

Figura 5.7: Interfaz de usuario de aplicación de Andriod

5.2.2. Gateway

Para la configuración del gateway se debe acceder a su consola. Esto se hace accediendo a la red WiFi "dragino-2121 d8z" conectándose a la IP 10.130.1.1. Dentro de la consola se pueden configurar las distintas funcionalidades que ofrece el gateway, desde los parámetros de radio hasta la encriptación y reenvío de paquetes a un servidor MQTT.

En la sección de LoRa se puede elegir la banda de frecuencia en la que se va a trabajar. Es posible elegir una banda personalizada, o seleccionar de las bandas pre-

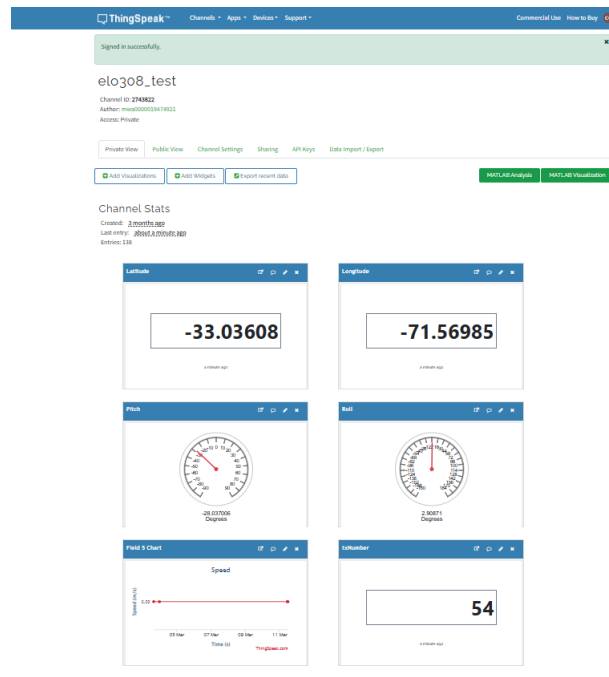


Figura 5.8: Canal de ThingSpeak creado para la aplicación

definidas y aprobadas. En este caso se selecciona la banda US915 y la segunda sub-banda. Podría elegirse cualquier sub-banda. En este caso, se elige esta siguiendo las recomendaciones que se hacen para conectarse a TTN. Esta configuración permite recibir mensajes con distintas configuraciones de Data Rate, por lo que resulta mejor para las pruebas que se realizarán.

LoRa Configuration

Debug Level: High

Radio Settings

Keep Alive Period (sec): 30

Frequency Plan: US915 United States 915Mhz (902~928)

Frequency Sub Band: 2: US915 , FSB2 (903.9~905.3)

Static GPS coordinates ?

Enable Static GPS: ☐

Latitude: 22.700000

Altitude (m): 450

Longitude: 114.240000

Current Mode: **LoRaWAN MQTT**

Save&Apply Disable Cancel

Figura 5.9: Configuración de radio LoRa

- 1: US915 , FSB1 (902.3~903.7)
- 2: US915 , FSB2 (903.9~905.3)**
- 3: US915 , FSB3 (905.5~906.9)
- 4: US915 , FSB4 (907.1~908.5)
- 5: US915 , FSB5 (908.7~910.1)
- 6: US915 , FSB6 (910.3~911.7)
- 7: US915 , FSB7 (911.9~913.3)
- 8: US915 , FSB8 (913.5~914.9)

Figura 5.10: Configuración de radio LoRa

Luego de configurar la radio, se configura la activación por personalización (ABP). Para esto, se deben generar las claves de antemano y agregarse en el firmware del nodo transmisor, así como en el gateway. Además, se debe configurar el canal MQTT para que el gateway solo acepte los mensajes de los dispositivos con las IDs almacenadas.

Primary LoRaWAN Server

Service Provider: Built-in for ABP Decode

Uplink Port: 1700

Downlink Port: 1700

Figura 5.11: Configuración de ABP

Channels Mapping:		
Local ID	Remote ID	API Keys
260C8E7B	260C8E7B	

Figura 5.12: Configuración de ABP

Por último, se debe configurar el servicio MQTT. Para esto se introduce la IP del broker al que se reenviarán los paquetes, el ID del dispositivo, el tópicos donde se publicarán, y el formato de los datos.

MQTT Client Configuration

MQTT Server Profile: General

Broker Address [-h]: 192.168.0.14
Broker Port [-p]: 1883

User ID [-u]: User ID
Password [-P]: Password Show

Certificate [--cert]:
Key [--key]:

CA File [--cafile]:

Client ID [-i]: dragino-gw

Publish

Enable Publish ☒
Quality of Service [-q]: QoS 0
Topic Format [-t]: CLIENTID/output ?
Data Format [-m]: DATA ?

Subscribe

Enable Subscribe ☒
Quality of Service [-q]: QoS 1
Topic Format [-t]: CLIENTID/# ?

Current Mode: LoRaWAN MQTT

Save&Apply Cancel

Figura 5.13: Configuración MQTT

Dentro de la consola del gateway se puede ver el tráfico de paquetes y asegurar así que funciona correctamente. Este se ve de la siguiente forma:

	Time	Message Type	Mod	Freq	Data Rate	CNT	Content
^	03/04-16:10:36	Data Unconfirmed Up	LoRa	905.3	SF7 BW125	7	Dev Addr: 260C8E7B, Size: 62
^	03/04-16:10:32	Data Unconfirmed Up	LoRa	905.1	SF7 BW125	6	Dev Addr: 260C8E7B, Size: 62
^	03/04-16:10:28	Data Unconfirmed Up	LoRa	904.9	SF7 BW125	5	Dev Addr: 260C8E7B, Size: 62

Figura 5.14: Tráfico del gateway

5.2.3. Nodo receptor

El nodo receptor utilizado en el escenario 1 está a la escucha de mensajes en una banda y data rate en específico. Cada vez que recibe un mensaje, hace la publicación en el tópico `esp32/output` del broker MQTT.

5.2.4. Node-red

Node-Red es la aplicación intermediaria donde se gestiona la recepción, procesamiento y redireccionamiento de los datos recopilados por los nodos sensores. Esta permite generar flujos de datos, desde su recolección hasta su publicación. Para esto, se usan dos métodos de recolección. El primero es un servidor MQTT, en donde se publican los datos de los sensores. Este servidor está hosteado en la misma máquina en donde está corriendo node-red, y es accesible desde cualquier dispositivo conectado a la red local. Así, se definen dos tópicos a los cuales se suscriben los respectivos flujos creados. Uno para el nodo receptor y otro para el gateway. Los flujos para los escenarios 1 y 2 se ven en las figuras 5.15 y 5.16

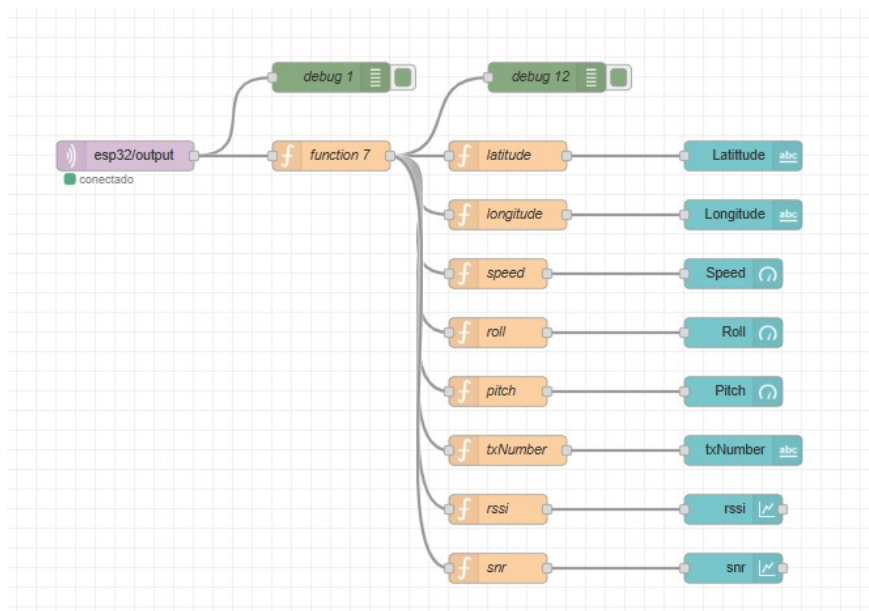


Figura 5.15: Flujo para nodo receptor

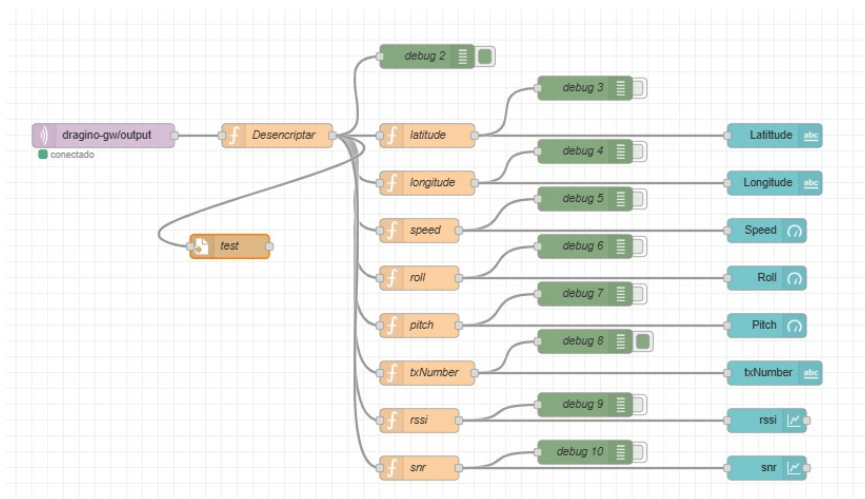


Figura 5.16: Flujo para gateway

El flujo de ambos casos es similar. Ambos se suscriben al t pico correspondiente del servidor MQTT, y procesan los datos para separarlos y luego subirlos al dashboard en sus pesta as respectivas. El flujo de 5.16 incluye una funci n de desencriptaci n. Recordar que el nodo y el gateway se comunican usando las llaves predefinidas para encriptar los datos. Este  ltimo desencripta los mensajes y los reenv a al broker MQTT como strings de ASCII que representan valores de bytes en hexadecimal. As , la funci n "desencriptar" convierte estos valores hexadecimales en enteros con signo o strings, seg n corresponda.

Luego de la desencriptaci n o el procesamiento inicial de los datos, se separa cada campo del mensaje. Recordar que los mensajes tienen el siguiente formato:

- **Escenario 1:** *latitud, longitud, velocidad, roll, pitch, txNumber, rssi, snr*
- **Escenario 2:** *rssi, snr, latitud, longitud, velocidad, roll, pitch, txNumber*

Esto es porque, en el escenario 2, el gateway a ade el *rssi* y *snr* en el encabezado de los paquetes.

Luego, usando los bloques de   frecidos por node-red, se puede construir una interfaz de usuario tipo dashboard en donde se puede desplegar la informaci n de los nodos. Esta se puede acceder desde cualquier dispositivo conectado a la red local

mediante la url `http://192.168.0.14:1880/ui`, o la ip del host. Las figuras 5.17, 5.18 muestran los resultados de interfaz de usuario para los dos primeros escenarios.

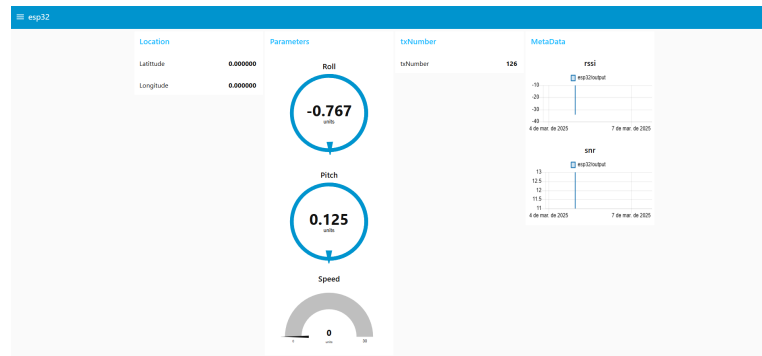


Figura 5.17: Dashboard para escenario 1

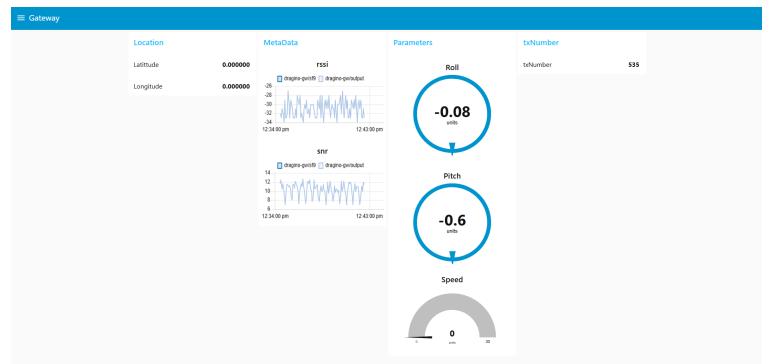


Figura 5.18: Dashboard para escenario 2

En el escenario 3, el flujo se basa en solicitudes http mediante la api de ThingSpeak, similar a lo hecho en el transmisor. Para esto, se ocupa una `.api_key` provista por la página. Ya que el canal se actualiza cada 15 segundos, el flujo hace la solicitud en este mismo intervalo de tiempo, ya que cualquier solicitud entre medio sería redundante. En la figura 5.19 se puede ver el flujo diseñado, y en 5.20 el dashboard.

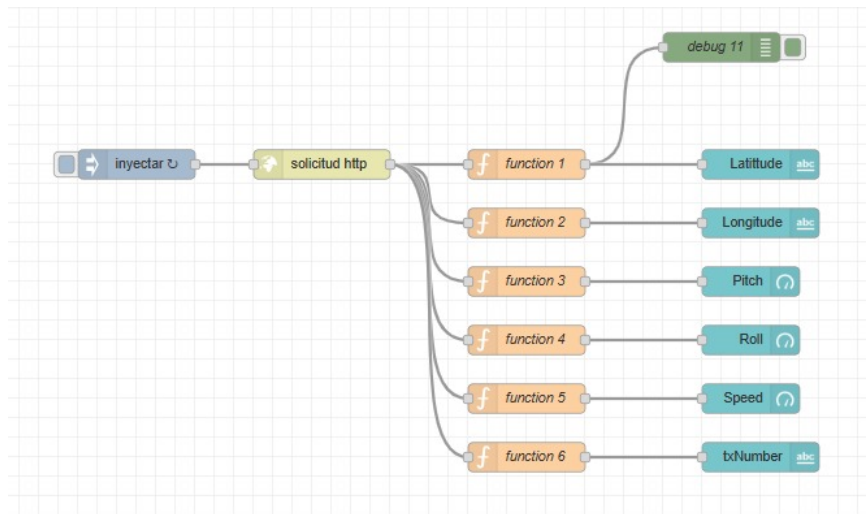


Figura 5.19: Flujo node-red para escenario 3

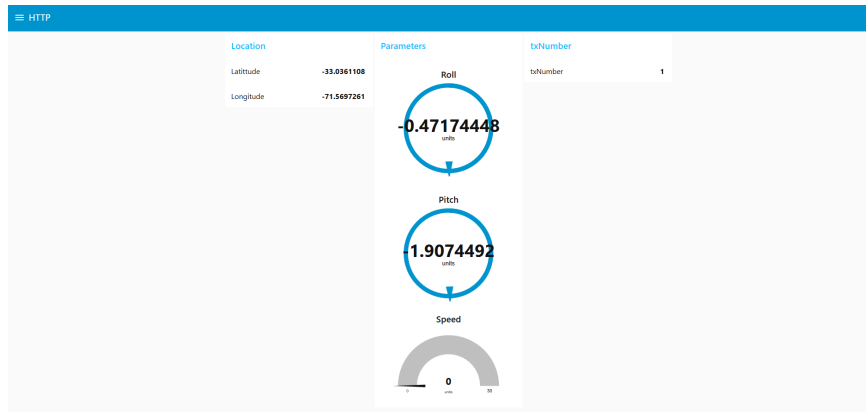


Figura 5.20: Dashboard escenario 3

5.3. Pruebas

A continuación se presentará el escenario y las condiciones bajo las cuales se realizarán las pruebas para los escenarios 1 y 2. Ya que el escenario 3 utiliza comunicaciones basadas en 4G, WiFi y protocolo de internet, las pruebas de conexión están fuera del escopo de este trabajo. Por esta razón, en esta sección nos enfocaremos en comprobar el funcionamiento de LoRa en redes IoT móviles.



Figura 5.21: Ubicación donde se tomó el test. Imagen extraída de Google Maps. Rectángulo naranja indica dónde se ubicó el gateway/nodo receptor.

El test se realizó en la ciudad de Con Cón, en la región de Valparaíso, y busca medir la conectividad de los nodos LoRa con el nodo receptor/gateway. El recorrido es el siguiente:

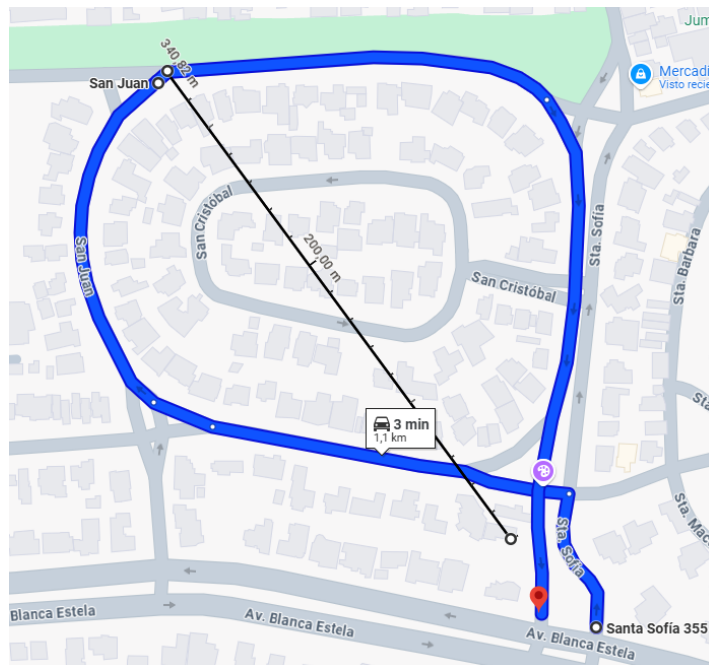


Figura 5.22: Google Maps

En este, la distancia más lejana del nodo al gateway es de 340 [m]. El test se

realizará en dos etapas. Primero, se realiza un test de nodo transmisor estático en distintas posiciones. En este, se miden los valores de "snr rssi"promedios en cada posición, así como la pérdida de paquetes total. En ambos escenarios se envían los valores de longitud, latitud, velocidad, roll, pitch, txNumber. Esto equivale a 50 bytes para el escenario 1 y 62 para el escenario 2, puesto que se agrega el encabezado por el protocolo LoRaWAN.

Se realizará el test en cada posición para los siguientes valores de *Data Rate*, que son los valores para *uplink* ofrecidos por la biblioteca *lmic.h* (LoRaWAN MAC in C) para la banda US915:

Data Rate (DR)	SF (Spreading Factor)	BW (Bandwidth)	Bitrate (bit/s)
DR_SF10	SF10	125 kHz	980
DR_SF9	SF9	125 kHz	1,760
DR_SF8	SF8	125 kHz	3,125
DR_SF7	SF7	125 kHz	5,470

Cuadro 5.2: Configuraciones de Data Rate para la banda US915

El test se realiza en las siguientes ubicaciones:

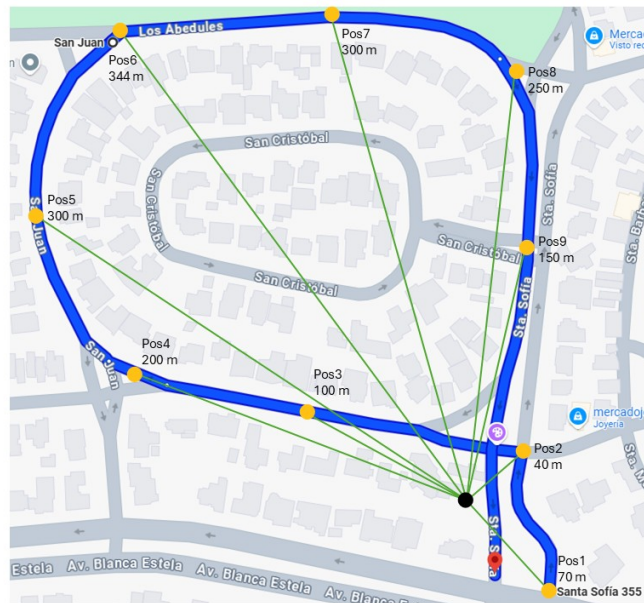


Figura 5.23: Ubicaciones de test estático



Figura 5.24: Vista satelital de ubicación de test estático

En la segunda parte del test, se realiza las pruebas de conectividad móvil. El recorrido se hace sobre un scooter eléctrico a una velocidad de entre 15 a 20[km/h]. El objetivo es medir la pérdida de paquetes para cada valor de data rate.

5.4. Resultados

Las pruebas de funcionamiento se realizaron en la misma ubicación, pero a distintas horas del día. El escenario corresponde a un barrio de la comuna de Con Con que cuenta con gran cantidad de casas y árboles, así como diferencias de altitud.

5.4.1. Escenario 1

A continuación, se presentan los resultados para el test del escenario 1. Estos fueron realizados un día soleado entre las 11:00 am y las 14:00 pm.

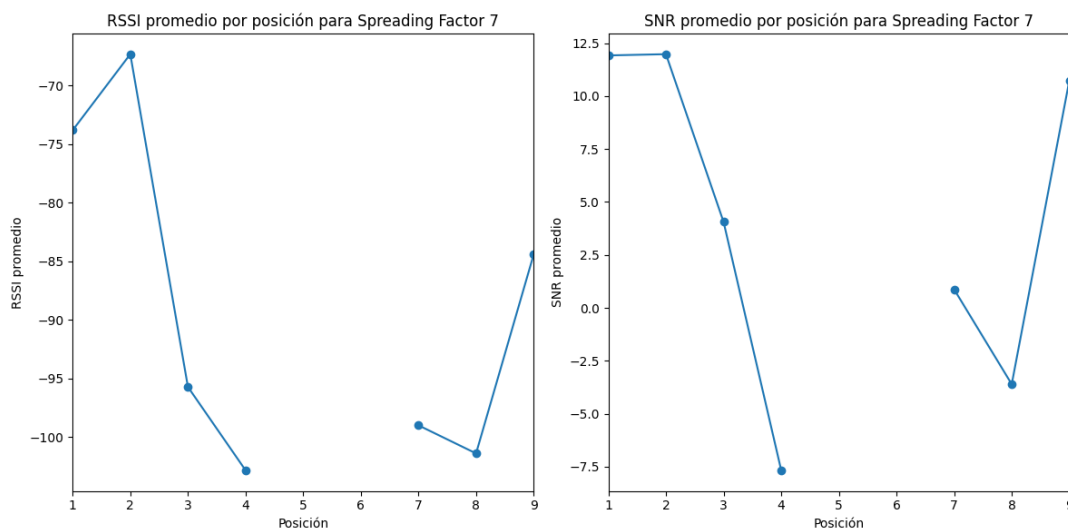


Figura 5.25: Resultados RSSI y SNR para DR_SF7

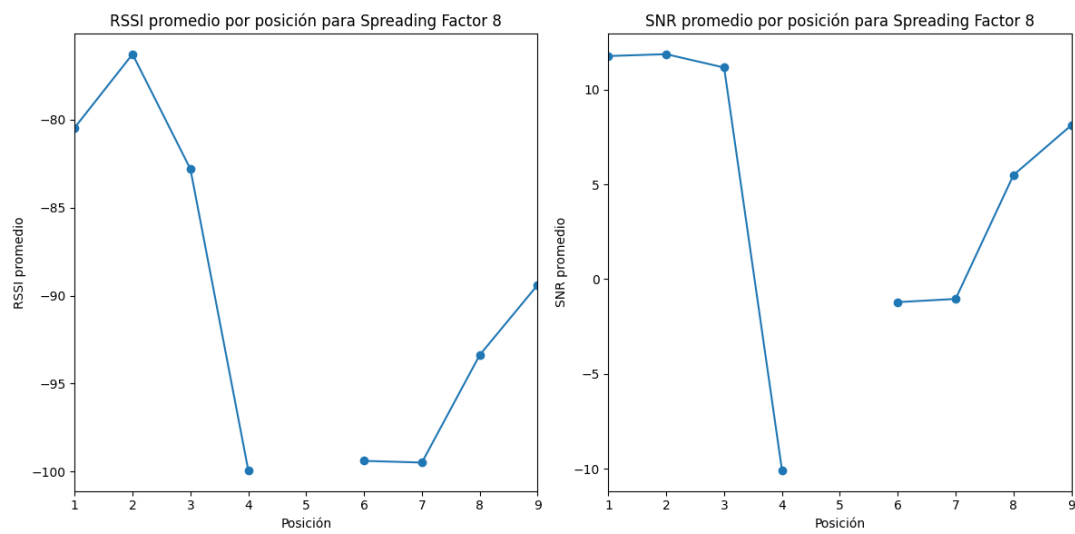


Figura 5.26: Resultados RSSI y SNR para DR_SF8

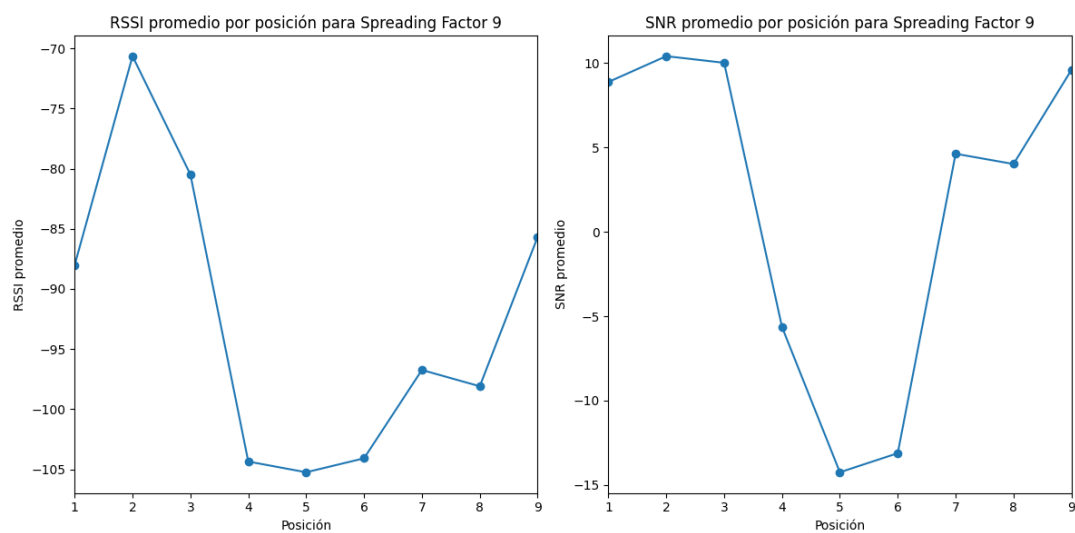


Figura 5.27: Resultados RSSI y SNR para DR_SF9

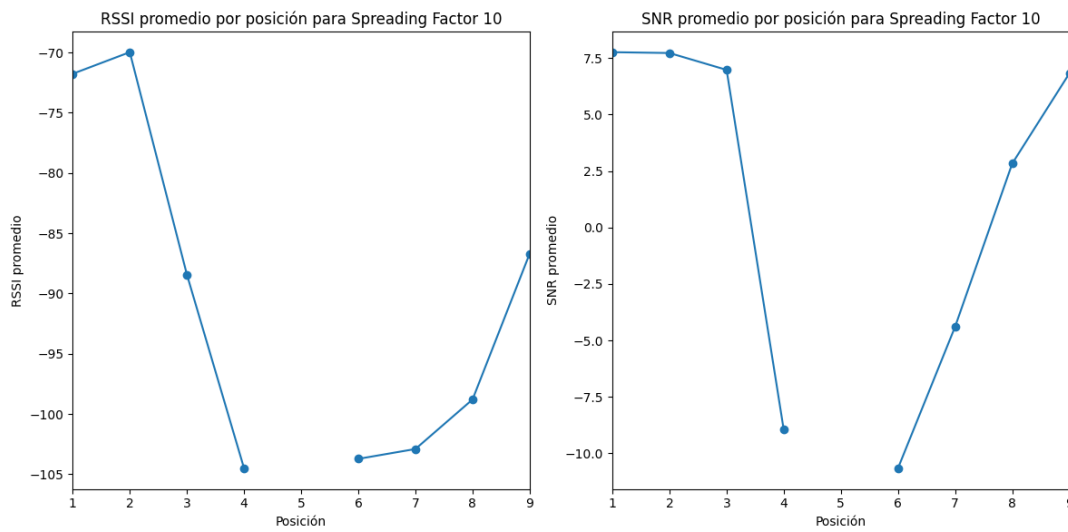


Figura 5.28: Resultados RSSI y SNR para DR_SF10

Posición	DR_SF7	DR_SF8	DR_SF9	DR_SF10
Pos 1 (70 m)	-73.82	-80.45	-88.01	-71.76
Pos 2 (40 m)	-67.33	-76.27	-70.64	-69.96
Pos 3 (100 m)	-95.73	-82.80	-80.49	-88.43
Pos 4 (200 m)	-102.83	-99.92	-104.36	-104.5
Pos 5 (300 m)	-	-	-105.25	-
Pos 6 (344 m)	-	-99.39	-104.09	-103.72
Pos 7 (300 m)	-99.0	-99.49	-96.74	-102.90
Pos 8 (250 m)	-101.39	-93.37	-98.09	-98.79
Pos 9 (150 m)	-84.41	-89.39	-85.68	-86.68

Cuadro 5.3: Valores de atenuación para diferentes posiciones y data rates

Posición	DR_SF7	DR_SF8	DR_SF9	DR_SF10
Pos 1 (70 m)	11.92	11.76	8.88	7.76
Pos 2 (40 m)	11.98	11.86	10.41	7.73
Pos 3 (100 m)	4.058	11.15	10.02	6.98
Pos 4 (200 m)	-7.66	-10.07	-5.64,	82.35
Pos 5 (300 m)	-	-	-14.25	-
Pos 6 (344 m)	-	-1.20	-13.12	-10.66
Pos 7 (300 m)	0.877	-1.03	4.63	-4.39
Pos 8 (250 m)	-3.60	5.490	4.02	2.83
Pos 9 (150 m)	10.74	8.117	9.59	6.82

Cuadro 5.4: Valores de SNR para diferentes posiciones y data rates

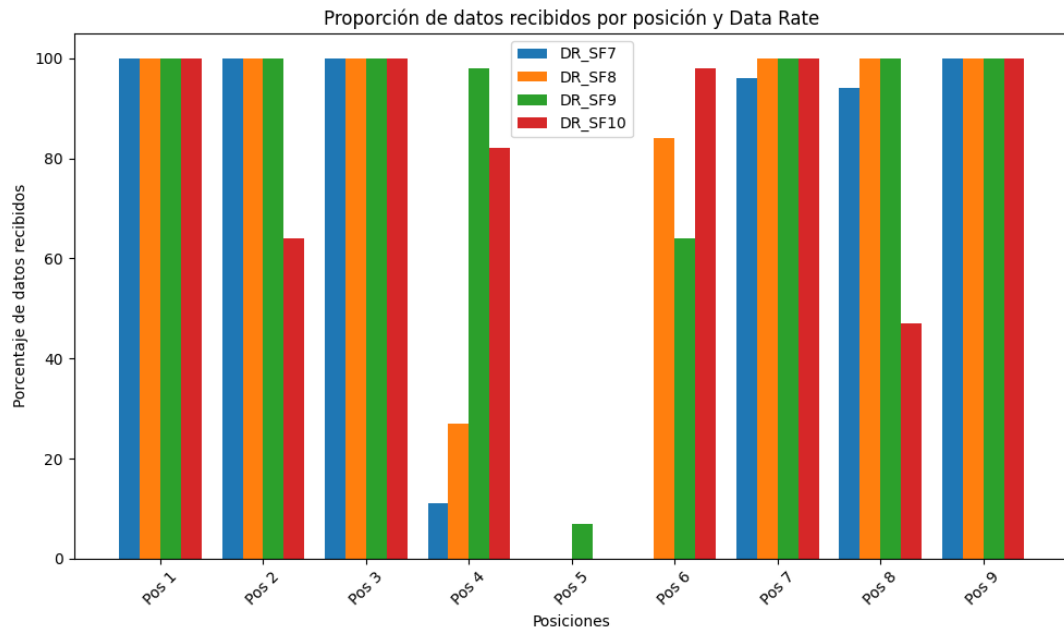


Figura 5.29: Porcentaje de paquetes recibidos en cada posición para cada Data Rate

Posición	DR_SF7	DR_SF8	DR_SF9	DR_SF10
Pos 1 (70m)	100 %	100 %	100 %	100 %
Pos 2 (40m)	100 %	100 %	100 %	64 %
Pos 3 (100m)	100 %	100 %	100 %	100 %
Pos 4 (200m)	11 %	27 %	98 %	82 %
Pos 5 (300m)	0 %	0 %	7 %	0 %
Pos 6 (344m)	0 %	84 %	64 %	98 %
Pos 7 (300m)	96 %	100 %	100 %	100 %
Pos 8 (250m)	94 %	100 %	100 %	47 %
Pos 9 (150m)	100 %	100 %	100 %	100 %

Cuadro 5.5: Porcentaje de paquetes recibidos en cada posición para cada Data Rate

Para el test en movimiento se hizo una vuelta por el recorrido en un scooter eléctrico con el nodo transmisor encendido. La cantidad de paquetes totales enviados depende del Data Rate seleccionado, por lo que para un spreading factor mayor, el tiempo en el aire y el bitrate son mayores, por lo que se envían menos paquetes que con un spreading factor menor. Así, la cantidad de paquetes enviados varía entre x y z. Luego, los resultados de pérdida de paquetes se pueden ver en la siguiente tabla:

Data Rate	Packet Loss (%)
DR_SF7	34.38 %
DR_SF8	27.70 %
DR_SF9	25.79 %
DR_SF10	38.38 %

Cuadro 5.6: Pérdida de paquetes en porcentaje para cada Data Rate

5.4.2. Escenario 2

A continuación, se muestran los resultados para el test estático del escenario 2. Las pruebas se realizaron un día despejado entre las 19:00 y las 23:00 hrs.

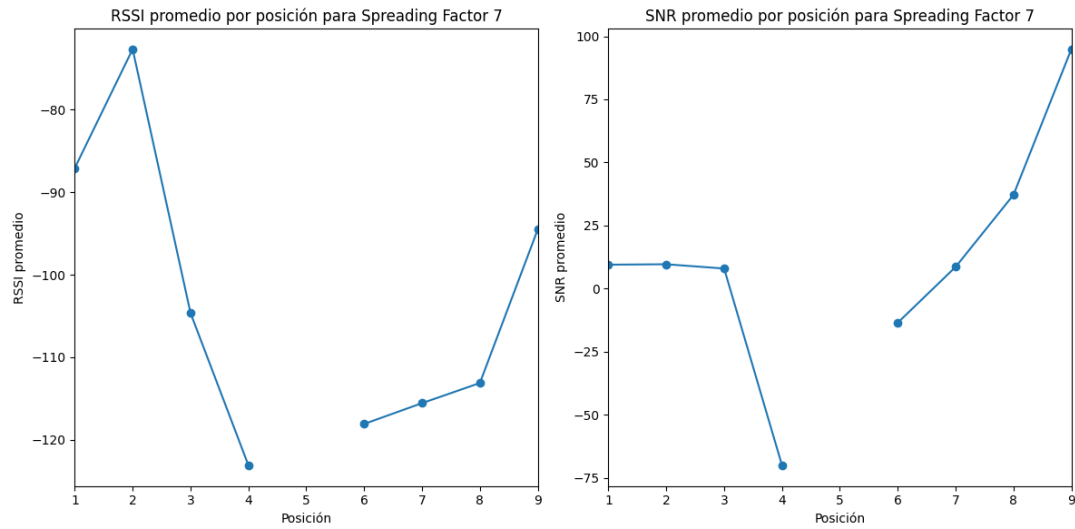


Figura 5.30: Resultados RSSI y SNR para DR_SF7

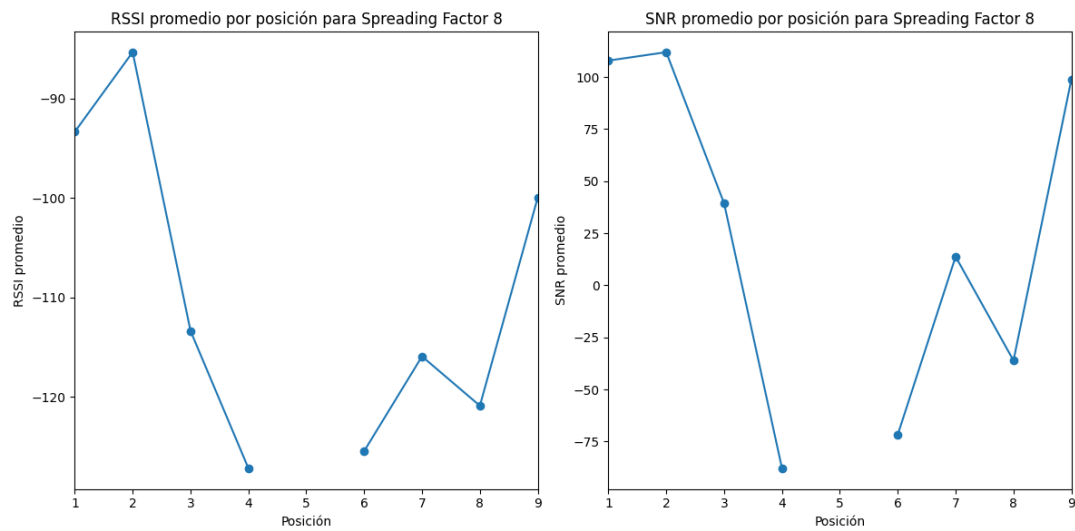


Figura 5.31: Resultados RSSI y SNR para DR_SF8

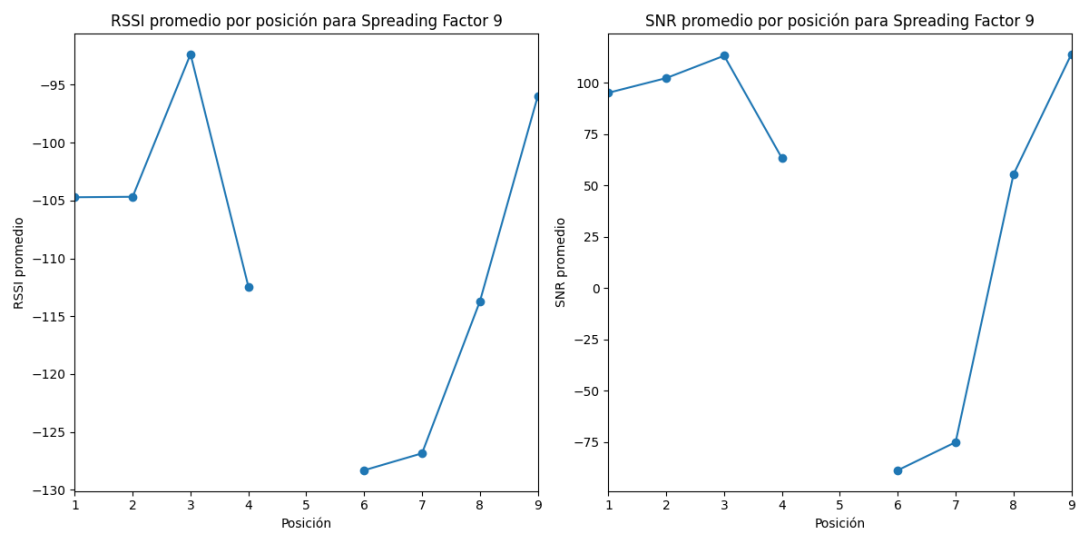


Figura 5.32: Resultados RSSI y SNR para DR_SF9

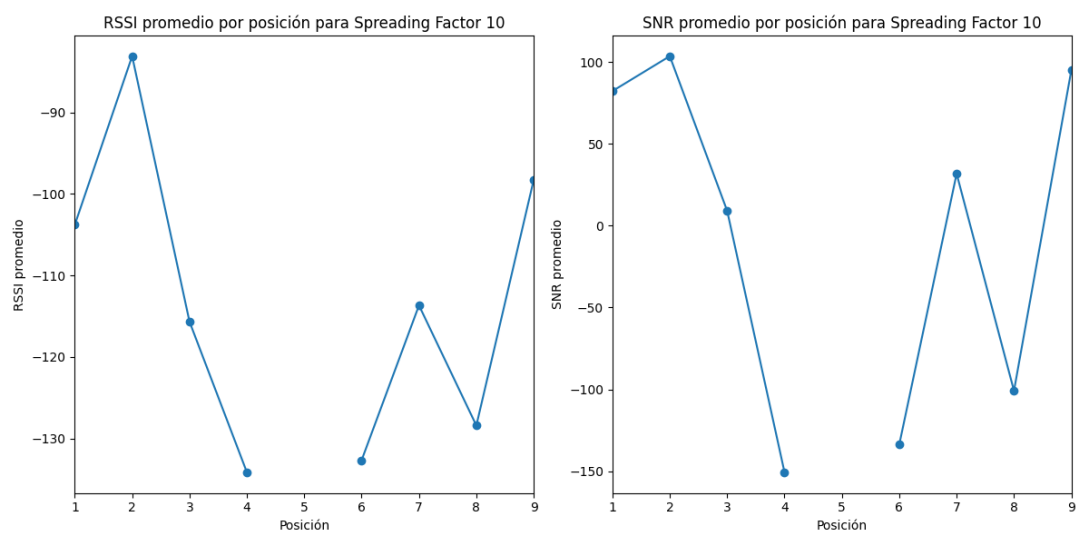


Figura 5.33: Resultados RSSI y SNR para DR_SF10

Posición	DR_SF7	DR_SF8	DR_SF9	DR_SF10
Pos 1 (70 m)	-87.08	-93.36	-104.72	-103.8
Pos 2 (40 m)	-72.68	-85.36	-104.68	-83.12
Pos 3 (100 m)	-104.64	-113.36	-92.36	-115.68
Pos 4 (200 m)	-123.08	-127.16	-112.44	-134.14
Pos 5 (300 m)	-	-	-	-
Pos 6 (344 m)	-118.08	-125.44	-128.30	-132.67
Pos 7 (300 m)	-115.56	-115.92	-126.84	-113.68
Pos 8 (250 m)	-113.12	-120.88	-113.72	-128.41
Pos 9 (150 m)	-94.52	-99.96	-96.00	-98.28

Cuadro 5.7: Valores de atenuación para diferentes posiciones y data rates

Posición	DR_SF7	DR_SF8	DR_SF9	DR_SF10
Pos 1 (70 m)	9.48	107.88	95.08	82.16
Pos 2 (40 m)	9.65	111.92	102.24	103.48
Pos 3 (100 m)	7.95	39.28	113.12	9.08
Pos 4 (200 m)	-70.0	-87.84	63.16	-150.57
Pos 5 (300 m)	-	-	-	-
Pos 6 (344 m)	-13.52	-71.72	-88.78	-133.67
Pos 7 (300 m)	8.6	13.76	-75.11	31.72
Pos 8 (250 m)	37.16	-36.12	55.32	-100.82
Pos 9 (150 m)	94.88	98.8	113.88	94.96

Cuadro 5.8: Valores de SNR para diferentes posiciones y data rates

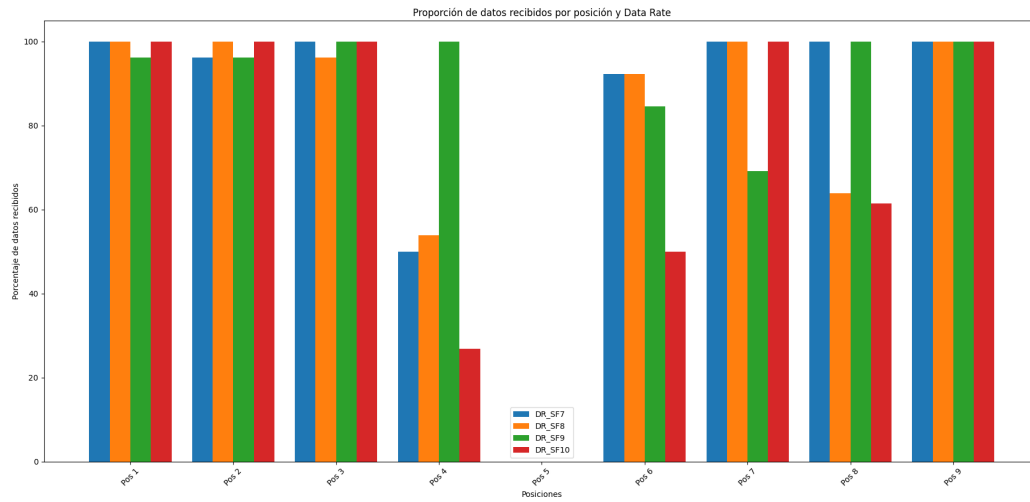


Figura 5.34: Porcentaje de paquetes recibidos en cada posición para cada Data Rate

Posición	DR_SF7	DR_SF8	DR_SF9	DR_SF10
Pos 1 (70 m)	100.00 %	100.00 %	96.15 %	100.00 %
Pos 2 (40 m)	96.15 %	100.00 %	96.15 %	100.00 %
Pos 3 (100 m)	100.00 %	96.15 %	100.00 %	100.00 %
Pos 4 (200 m)	50.00 %	53.85 %	100.00 %	26.92 %
Pos 5 (300 m)	0.00 %	0.00 %	0.00 %	0.00 %
Pos 6 (344 m)	92.31 %	92.31 %	84.62 %	50.00 %
Pos 7 (300 m)	100.00 %	100.00 %	69.23 %	100.00 %
Pos 8 (250 m)	100.00 %	63.89 %	100.00 %	61.54 %
Pos 9 (150 m)	100.00 %	100.00 %	100.00 %	100.00 %

Cuadro 5.9: Proporción de paquetes recibidos en porcentaje para diferentes posiciones y data rates

Para el test de movimiento se aplica la misma lógica que en el escenario 1, en donde la cantidad de paquetes enviados depende de factor de dispersión. En este caso, el total de paquetes transmitidos varía entre 53 y 42. Los resultados se ven en la siguiente tabla.

Data Rate	Packet Loss (%)
DR_SF7	41.51 %
DR_SF8	42.00 %
DR_SF9	30.23 %
DR_SF10	33.33 %

Cuadro 5.10: Pérdida de paquetes en porcentaje para cada Data Rate

Capítulo 6

Resultados de Simulación

En este capítulo se realiza una simulación con el fin de estudiar la escalabilidad de la plataforma al utilizar distintos nodos transmisores y receptores. Para esto se utilizó la aplicación **CupCarbon**. CupCarbon es una herramienta de simulación para redes IoT y redes de sensores inalámbricos (WSN). Permite modelar, visualizar y analizar la comunicación entre nodos en un entorno urbano o rural, considerando aspectos como consumo energético, propagación de señales y cobertura de la red. Se utiliza para diseñar y optimizar redes IoT antes de su implementación real, facilitando la evaluación de topologías, protocolos de comunicación y estrategias de despliegue de nodos y gateways.

La simulación se realizará en el campus casa central de la Universidad Técnica Federico Santa María. Se realizarán 4 escenarios, descritos en la siguiente tabla:

Escenario	Descripción
1	Conexión de un nodo estático con todos los gateways
2	Conexión de un nodo móvil con un solo gateway
3	Conexión de un nodo móvil con todos los gateways
4	Conexión de 4 nodos móviles con todos los gateways

Cuadro 6.1: Escenarios para la simulación en CupCarbon



Figura 6.1: Escenario de simulación con un solo gateway

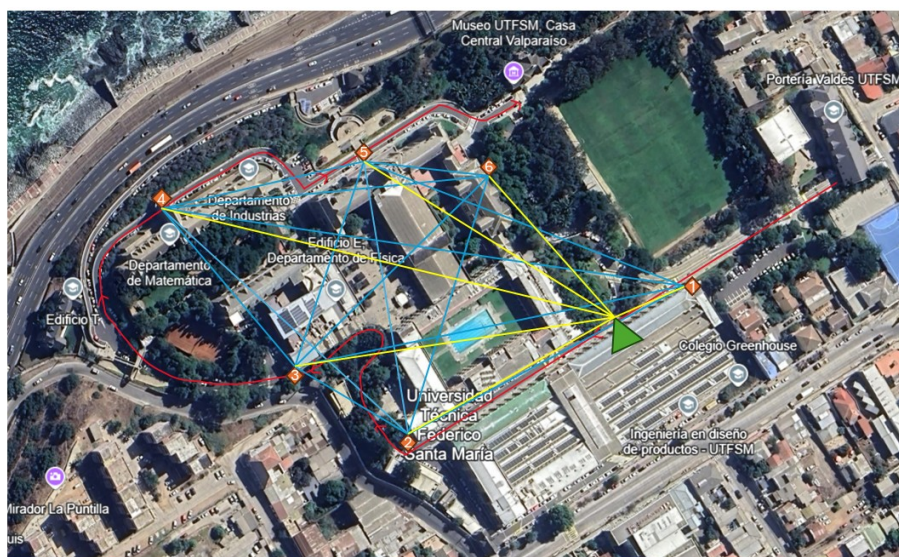


Figura 6.2: Escenario de simulación un nodo con varios gateways



Figura 6.3: Escenario de simulación con varios gateways

6.1. Escenario 1

En este escenario se simula la transmisión de mensajes durante una hora. Los nodos transmisores envían paquetes cada 5 segundos, lo que da un total de 710 mensajes transmitidos (tomando en cuenta delay de inicialización). Como parámetros de radio se escogió un Data Rate de 25000 (por defecto) y un Spreading Factor 7. El objetivo de esta simulación y de las siguientes es observar el efecto de interferencias y la escalabilidad del sistema, por lo que se usarán los mismo parámetros de radio para todas las simulaciones.

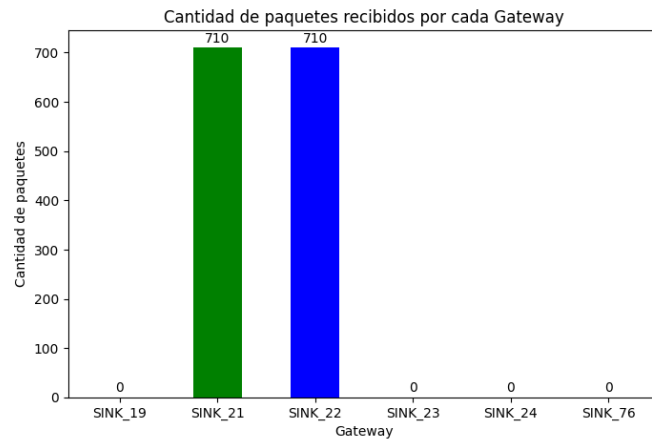


Figura 6.4: Cantidad de paquetes recibidos por cada Gateway

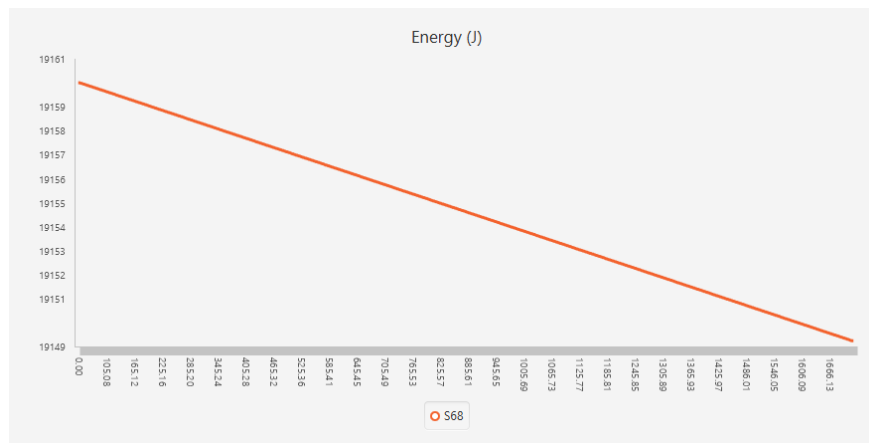


Figura 6.5: Nivel de batería del nodo transmisor

En este escenario se mide además el nivel de la batería de los nodos transmisores, así como el consumo en el tiempo de los mismos. Es posible observar un consumo uniforme a lo largo de la simulación.

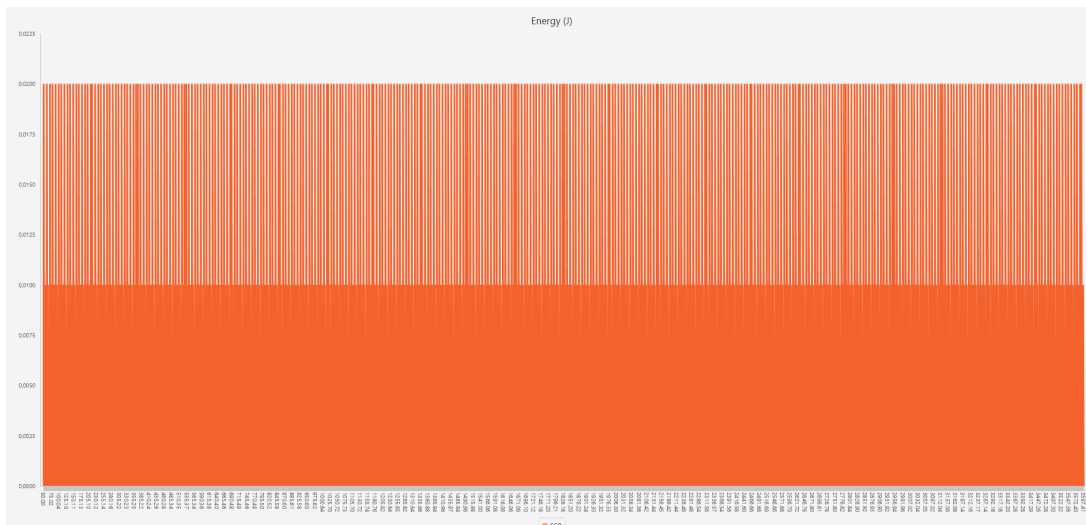


Figura 6.6: Consumo del nodo transmisor

6.1.1. Escenario 2

En este escenario se busca imitar las condiciones en las que se hicieron las pruebas prácticas de implementación. EL objetivo es medir la recepción de los paquetes con un nodo transmisor en movimiento que pierde línea de visión directa del único punto de acceso disponible.

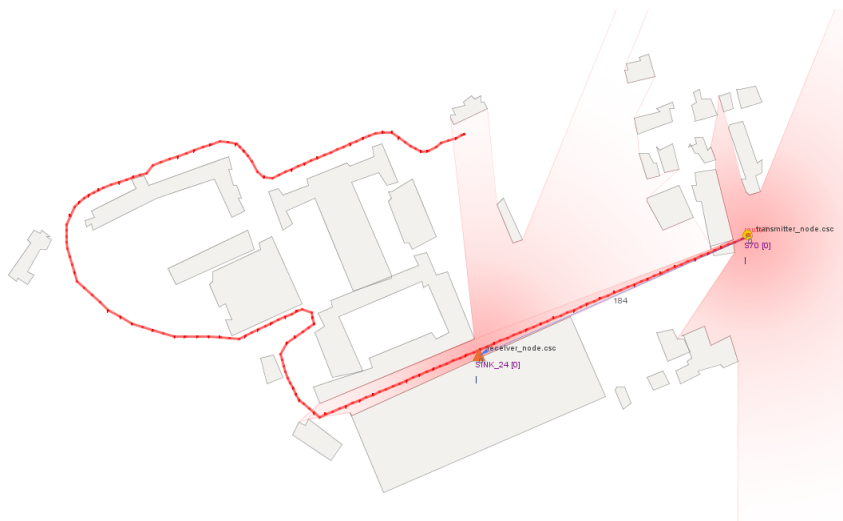


Figura 6.7: Ruta seguida por el nodo transmisor

Nodo	Pérdida de paquetes
1	64.21 %
Paquetes enviados	Paquetes recibidos
95	34

Cuadro 6.2: Pérdida de paquetes

6.2. Escenario 3

En este escenario se busca mejorar las condiciones impuestas en la práctica, añadiendo distintos puntos de acceso, tal que el nodo transmisor siempre tenga uno a la vista. Se espera mejorar la recepción de los paquetes y el porcentaje de pérdida.

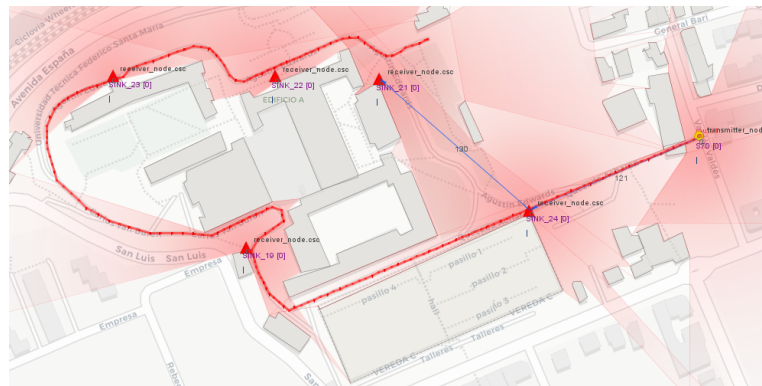
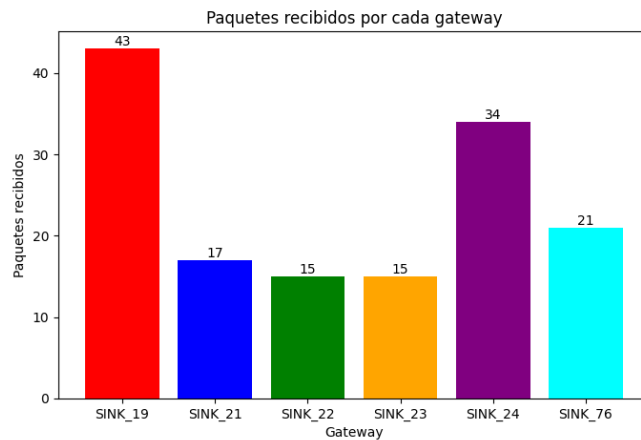
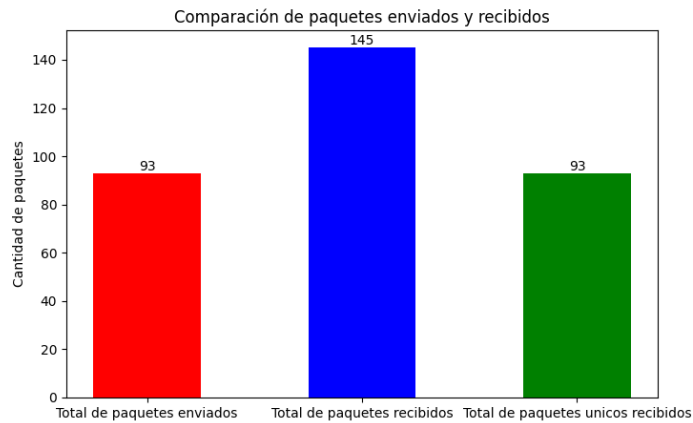


Figura 6.8: Ruta seguida en el escenario 3



6.3. Escenario 4

En el escenario cuatro se agregaron nodos transmisores. Cada uno sigue una ruta distinta, y se deja el arreglo de puntos de acceso como en el escenario anterior. El objetivo es medir la escalabilidad del sistema y verificar interferencias en las comunicaciones.



Figura 6.9: Ruta para el nodo S70



Figura 6.10: Ruta para el nodo S75

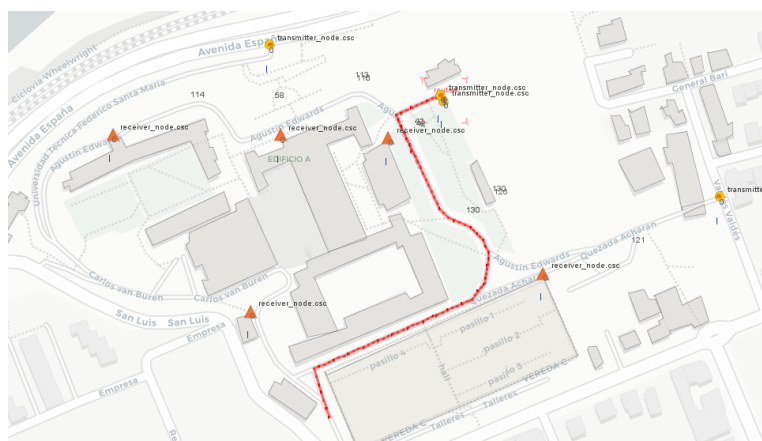


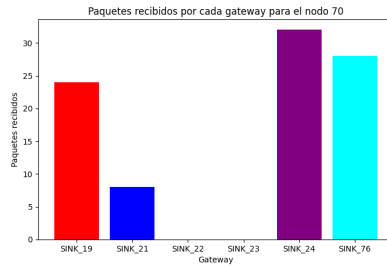
Figura 6.11: Ruta para el nodo S74



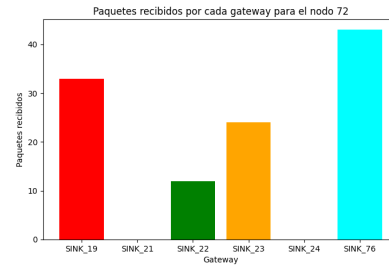
Figura 6.12: Ruta del nodo S72

Nodo	Pérdida de paquetes
1	0 %
2	0 %
3	0 %
4	0 %

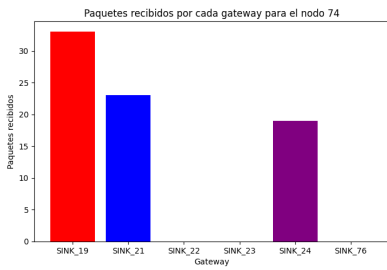
Cuadro 6.3: Pérdida de paquetes por cada nodo



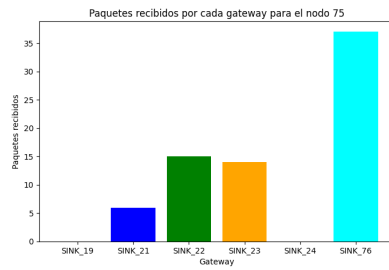
(a) Nodo S70



(b) Nodo S72

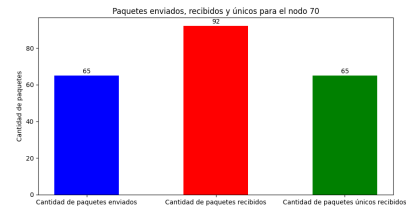


(c) Nodo S74

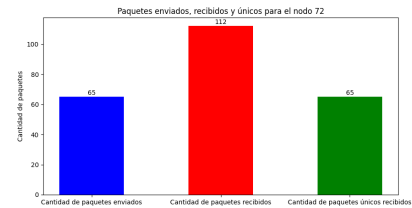


(d) Nodo S75

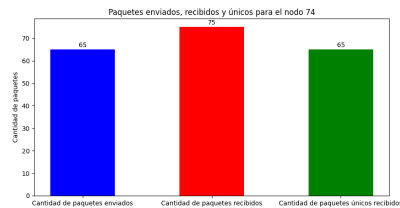
Figura 6.13: Paquetes recibidos por cada gateway para cada nodo



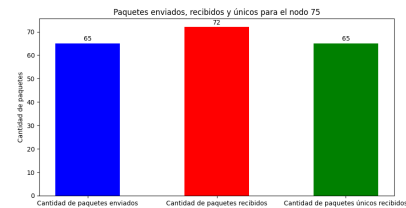
(a) Nodo S70



(b) Nodo S72



(c) Nodo S74



(d) Nodo S75

Figura 6.14: Paquetes enviados, recibidos y únicos para cada nodo

Capítulo 7

Conclusiones

Luego de haber diseñado e implementado el sistema, se puede destacar la variedad de soluciones propuestas que, aún variando en distintos aspectos, proponen soluciones válidas y funcionales para cumplir con el objetivo del proyecto.

Con respecto a las soluciones 1 (nodo a nodo) y 2 (nodo a gateway), se observa una mejor calidad de señal en el primero en comparación con el Escenario 2, especialmente en distancias menores a 200 m. Los valores de RSSI en este escenario son significativamente más altos (menos negativos), indicando una recepción más estable. Lo mismo se puede decir del SNR y, por ende, de la pérdida de paquetes.

Al trabajar con la librería "lmic.h" de IBM para entorno Arduino, se ofrecen distintas ventajas y facilidades para implementar LoRaWAN. La biblioteca viene optimizada para mantenerse dentro de las regulaciones de las bandas seleccionadas. Por esta razón, no es viable trabajar con configuraciones de factor de dispersión y ancho de banda que no se encuentren configuradas de antemano. Lo mismo sucede con los intervalos de transmisión y con la potencia de transmisión. La banda **US915** tiene un máximo de potencia de transmisión de 14 dBm (pese a que se configure un valor distinto en el código), por lo que resulta desfavorecida a la hora de comparar con el escenario 1. Así, a la hora de implementar un sistema dentro de las normas y regulaciones, una configuración con más puntos de acceso es necesaria para evitar pérdidas

de paquetes muy altas. A pesar de esto, es posible ver que los resultados para las pruebas estáticas ofrecen valores de RSSI y SNR dentro de los rangos aceptables.

En resumen, teniendo en cuenta las restricciones y uso justo de las bandas de frecuencia en las que se participa, es necesario usar más gateways con el fin de mejorar la conexión y asegurar la estabilidad de la plataforma. Esto se ve reflejado en los resultados, dado que las posiciones más cercanas al gateway y con menor interferencia ofrecieron mejores resultados. Además, trabajos externos como *“On the Use of LoRaWAN for Mobile Internet of Things: The Impact of Mobility”* [[3]] demuestran la capacidad de LoRaWAN para aplicaciones IoT móviles en simulaciones de red, lo que refuerza la propuesta de una mejor red para la plataforma.

Por otro lado, las simulaciones también sugieren lo mismo, puesto que la pérdida de paquetes se da únicamente por interferencias de edificios o estructuras.

Dados los escenarios puestos a prueba en CupCarbon, se puede concluir que la interferencia juega un papel importante a la hora de trabajar con LoRa. Luego, sumado a las restricciones de banda, se vuelve evidente la necesidad de mantener una línea de visión despejada (dentro de lo posible) o bien una cercanía con los puntos de acceso de la red implementada.

A pesar de esto, se considera que la implementación y funcionamiento de la plataforma es exitosa, dado que en un entorno profesional, las limitaciones de hardware utilizadas en las pruebas realizadas para este trabajo son menores, y la disponibilidad de puntos de acceso es mayor. Además, la solución implementada en Android ofrece un método distinto, con una tecnología de comunicación ampliamente disponible que cualquier usuario que participe en el uso de movilidad compartida tendrá disponible.

Finalmente, para un trabajo futuro quedan los siguientes puntos a tratar:

- Añadir más cantidad de puntos de acceso a la red.
- Optimizar la velocidad de transmisión de los paquetes.
- Hacer pruebas con distintos *Coding Rates*.

- Hacer pruebas con *Adaptive Data Rate (ADR)*.
- Conexión bilateral Nodo - Gateway.
- Enviar y ejecutar comandos en los nodos desde el servidor local.
- Implementar acceso al servidor local desde fuera de la red usando VPN.

Bibliografía

- [1] Aburukba, R.; Al-Ali, A.R.; Riaz, A.H.; Al Nabulsi, A.; Khan, D.; Khan, S.; Amer, M. *Fog Computing Approach for Shared Mobility in Smart Cities*. *Energies* 2021, 14, 8174.
- [2] Ricardo Salazar-Cabrera; Alvaro Pachon de la Cruz; Juan Manuel Madrid Molina *Proof of Concept of an IoT-Based Public Vehicle Tracking System, Using LoRa (Long Range) and Intelligent Transportation System (ITS) Services*. *Journal of Computer Networks and Communications* Volume 2019
- [3] Al mojamed, M. *On the Use of LoRaWAN for Mobile Internet of Things: The Impact of Mobility*. *Appl. Syst. Innov.* 2022, 5, 5.
- [4] D. F. Carvalho, A. Depari, P. Ferrari, A. Flammini, S. Rinaldi and E. Sisinni, *On the feasibility of mobile sensing and tracking applications based on LPWAN*, 2018 IEEE Sensors Applications Symposium (SAS), Seoul, Korea (South), 2018, pp. 1-6, doi: 10.1109/SAS.2018.8336765.
- [5] G. Slawomir, *Next generation ITS implementation aspects in 5G wireless communication network*, 2017 15th International Conference on ITS Telecommunications (ITST), Warsaw, Poland, 2017, pp. 1-7, doi: 10.1109/ITST.2017.7972230.
- [6] A. M. Joy, A. Ravi, J. Roy, G. V. Karthikeyan and T. P. Rajan, *IoT Enabled Electronic Security and Remote Monitoring Solution For Electric Bikes*, 2022 IEEE

- International Power and Renewable Energy Conference (IPRECON), Kollam, India, 2022, pp. 1-6, doi: 10.1109/IPRECON55716.2022.10059526.
- [7] Y. Ayuba, S. H. Supangkat and S. S. Wibowo, *Mobility as a Service: A Problems and Research Opportunities*, 2023 10th International Conference on ICT for Smart Society (ICISS), Bandung, Indonesia, 2023, pp. 1-7, doi: 10.1109/ICISS59129.2023.10292068.
- [8] S. Garcia-Perez et al., *Alternative mobility system using IoT and smart-cities approaches through the use of the bike for a sector of the city of Bogota within a simulated environment*, 2020 Smart City Symposium Prague (SCSP), Prague, Czech Republic, 2020, pp. 1-6, doi: 10.1109/SCSP49987.2020.9134025.
- [9] Hamidreza Alai, Woongsun Jeon, Lee Alexander, Rajesh Rajamani, *A smart e-scooter with embedded estimation of rear vehicle trajectories for rider protection*, Mechanical Systems and Signal Processing, Volume 222, 2025, 111786, ISSN 0888-3270, <https://doi.org/10.1016/j.ymssp.2024.111786>.
- [10] A. Cismas, M. Ioana, C. Vlad and G. Casu, *Crash Detection Using IMU Sensors*, 2017 21st International Conference on Control Systems and Computer Science (CSCS), Bucharest, Romania, 2017, pp. 672-676, doi: 10.1109/CSCS.2017.103.
- [11] Simone Gelmini, Silvia Strada, Mara Tanelli, Sergio Savaresi, Claudio De Tommasi, *Automatic crash detection system for two-wheeled vehicles: design and experimental validation*, IFAC-PapersOnLine, Volume 52, Issue 5, 2019, Pages 498-503, ISSN 2405-8963, <https://doi.org/10.1016/j.ifacol.2019.09.079>.
- [12] R. Hartmann, M. Abdelhamid, J. Manuel Martínez, *LoRa based communications for internet of electric vehicles*, Biblioteca USM, 2023.
- [13] Arduino-LMIC library, <https://github.com/matthijskooijman/arduino-lmic>, 13 de Marzo de 2025.

- [14] Material CupCarbon, <https://cupcarbon.com/>, 13 de Marzo de 2025.
- [15] Ministerio de Energía, Gobierno de Chile. (s.f.). Plataforma de Electromovilidad, https://energia.gob.cl/sites/default/files/estrategia-nacional-electromovilidad_ministerio-de-energia.pdf (document), 2.3
- [16] Radiografía al uso de medios de transporte de los chilenos, <https://aimchile.cl/radiografia-al-uso-de-medios-de-transportede-los-chilenos/>, 13 de Marzo de 2025.
- [17] Shared use mobility center, *What is Shared Mobility?*, <https://sharedusemobilitycenter.org/what-is-shared-mobility/>, 14 de Marzo de 2025.
- [18] Navit, *What is Shared Mobility?*, <https://www.navit.com/resources/what-is-shared-mobility>, 14 de Marzo de 2025.
- [19] Wikipedia, *Micromobility*, <https://en.wikipedia.org/wiki/Micromobility>, 14 de Marzo de 2025.
- [20] Cumulocity, *What is an IoT Platform?*, <https://www.cumulocity.com/resource-library/what-is-an-iot-platform/>, 14 de Marzo de 2025.
- [21] The Things Network, *What are LoRa and LoRaWAN?*, <https://www.thethingsnetwork.org/docs/lorawan/what-is-lorawan/>, 14 de Marzo de 2025.
- [22] Wikipedia, *LoRaWAN*, <https://es.wikipedia.org/wiki/LoRaWAN>, 14 de Marzo de 2025.

- [23] Ministerio de Energía, *Plataforma de electromovilidad*, <https://energia.gob.cl/electromovilidad/modelos-de-negocios/empresas-de-micromovilidad>, 14 de Marzo de 2025.
- [24] Android Developers, <https://developer.android.com/develop/sensors-and-location/>

Apéndice A

Códigos

A continuación se presenta el código fuente utilizado en el proyecto:

```
1  /* Heltec Automation send communication test example
2  *
3  * Function:
4  * 1. Send data from a esp32 device over hardware
5  *
6  * Description:
7  *
8  * HelTec AutoMation, Chengdu, China
9  *
10 * www.heltec.org
11 *
12 * this project also realles in GitHub:
13 * https://github.com/Heltec-Aaron-Lee/WiFi\_Kit\_series
14 * */
15
16 #include "LoRaWan_APP.h"
17 #include "Arduino.h"
18 #include "FastIMU.h"
19 #include "Wire.h"
20 #include "HardwareSerial.h"
21 #include "HT_TinyGPS++.h"
```

```

22
23 //Definicion de puertos seriales
24 #define RX_PIN 16
25 #define TX_PIN 17
26 #define BAUD_RATE 9600
27
28 //Definicion de pines I2C
29 #define IMU_ADDRESS 0x68
30
31 //Buffer de lectura de datos
32 #define BUFFER_LENGTH 10
33
34 // Constante de aceleracion de gravedad
35 #define G 9.805
36
37 #define BUFFER_SIZE 50
38
39 //Definicion de parametros y variables para I2C (MPU9250)
40 MPU9250 IMU;
41
42 //Pines SCL y SDA
43 const int I2C_SDA = 21;
44 const int I2C_SCL = 22;
45
46 //Definicion de variables para la informacion de IMU
47 calData calib = { 0 };
48 AccelData accelData;
49
50 float accelXbuf[BUFFER_LENGTH] = {0};
51 float accelYbuf[BUFFER_LENGTH] = {0};
52 float accelZbuf[BUFFER_LENGTH] = {0};
53 int writeIndex = 0;
54 float smoothX, smoothY, smoothZ, rollAcc, pitchAcc, absXY, accelZdeb
;

```

```

55
56 //Definicion de variables para GPS
57 TinyGPSPlus GPS;
58 bool gpsReady = false;
59 float speed;
60 float lat;
61 float lon;
62
63 //Definicion de parametros para LoRa
64 #define RF_FREQUENCY 904300000 // Hz
65 #define TX_OUTPUT_POWER 22      // dBm
66 #define LORA_BANDWIDTH 0        // 125 kHz
67 #define LORA_SPREADING_FACTOR 7 // SF7
68 #define LORA_CODINGRATE 1       // 4/5
69 #define LORA_PREAMBLE_LENGTH 8  // Same for Tx and Rx
70 #define LORA_SYMBOL_TIMEOUT 0   // Symbols
71 #define LORA_FIX_LENGTH_PAYLOAD_ON false
72 #define LORA_IQ_INVERSION_ON false
73
74 #define RX_TIMEOUT_VALUE 1000
75 #define BUFFER_SIZE 50
76
77 char txpacket[BUFFER_SIZE];
78 char rxpacket[BUFFER_SIZE];
79
80 int txNumber=1;
81 bool lora_idle=true;
82
83 static RadioEvents_t RadioEvents;
84 void OnTxDone( void );
85 void OnTxTimeout( void );
86
87 void setup() {
88     Serial.begin(115200);

```

```

89   Mcu.begin(HELTEC_BOARD, SLOW_CLK_TPYE);
90
91   pinMode(2, OUTPUT);
92   pinMode(0, INPUT_PULLUP);
93   Wire.begin(I2C_SDA, I2C_SCL);
94   Wire.setClock(400000);
95   imuInit();
96   txNumber=0;
97
98   RadioEvents.TxDone = OnTxDone;
99   RadioEvents.TxTimeout = OnTxTimeout;
100  Radio.Init( &RadioEvents );
101  Radio.SetChannel( RF_FREQUENCY );
102  Radio.SetTxConfig( MODEM_LORA, TX_OUTPUT_POWER, 0,
103                    LORA_BANDWIDTH,
104                    LORA_SPREADING_FACTOR,
105                    LORA_CODINGRATE,
106                    LORA_PREAMBLE_LENGTH,
107                    LORA_FIX_LENGTH_PAYLOAD_ON,
108                    true, 0, 0, LORA_IQ_INVERSION_ON,
109                    3000 );
110 }
111
112 void loop()
113 {
114   imuUpdate();
115   gpsUpdate();
116
117   if(gpsReady) {
118     if(txNumber <=50) {
119       if(lora_idle == true)
120       {
121         sprintf(txpacket, "0.000000,0.000000,0.0, %.3f, %.3f, %d",
122               rollAcc, pitchAcc, txNumber);

```

```

118     Serial.printf("\r\nsending packet \"%s\" , length %d\r\n",
txpacket, strlen(txpacket));
119     Radio.Send( (uint8_t *)txpacket, strlen(txpacket));
120     txNumber += 1;
121     lora_idle = false;
122 }
123 Radio.IrqProcess();
124 }
125 }
126 else{
127     digitalWrite(2,HIGH);
128     while(1);
129 }
130 delay(500);
131 }
132
133 void imuInit(){
134     int err = IMU.init(calib, IMU_ADDRESS);
135     if (err != 0) {
136         Serial.print("Error initializing IMU: ");
137         Serial.println(err);
138         while (true);
139     }
140     err = IMU.setGyroRange(250);
141     err = IMU.setAccelRange(8);
142     if (err != 0) {
143         Serial.print("Error Setting range: ");
144         Serial.println(err);
145         while (true);
146     }
147 }
148
149 void imuUpdate(){
150     IMU.update();

```

```

151 IMU.getAccel(&accelData);
152 accelData.accelX = (accelData.accelX + 1.3) * G;
153 accelData.accelY = (accelData.accelY + 0.96) * G;
154 accelData.accelZ = (accelData.accelZ + 0.26) * G;
155 accelXbuf[writeIndex] = accelData.accelX;
156 accelYbuf[writeIndex] = accelData.accelY;
157 accelZbuf[writeIndex] = accelData.accelZ;
158 writeIndex = (writeIndex + 1) % BUFFER_LENGTH;
159 getRollPitch(smoothX, smoothY, smoothZ);
160 }
161
162 void gpsUpdate() {
163     if (Serial2.available()) {
164         if (Serial2.peek() != '\n') {
165             GPS.encode(Serial2.read());
166         }
167         else {
168             gpsReady = true;
169             speed = GPS.speed.mps();
170             lat = GPS.location.lat();
171             lon = GPS.location.lng();
172         }
173     }
174 }
175
176 void getRollPitch(float accelX, float accelY, float accelZ) {
177     pitchAcc = atan2(accelY, accelZ) * 180 / PI;
178     rollAcc = atan2(-accelX, sqrt(accelY * accelY + accelZ * accelZ))
        * 180 / PI;
179 }
180
181 void OnTxDone(void) {
182     Serial.println("TX done.....");
183     lora_idle = true;

```

```

184 }
185
186 void OnTxTimeout(void) {
187     Radio.Sleep();
188     Serial.println("TX Timeout.....");
189     lora_idle = true;
190 }

```

Listing A.1: Código .ino para nodo transmisor escenario 1

```

1  /*
    *****
2  * Copyright (c) 2015 Thomas Telkamp and Matthijs Kooijman
3  *
4  * Permission is hereby granted, free of charge, to anyone
5  * obtaining a copy of this document and accompanying files,
6  * to do whatever they want with them without any restriction,
7  * including, but not limited to, copying, modification and
   redistribution.
8  * NO WARRANTY OF ANY KIND IS PROVIDED.
9  *
10 * This example sends a valid LoRaWAN packet with payload "Hello,
11 * world!", using frequency and encryption settings matching those
   of
12 * the The Things Network.
13 *
14 * This uses ABP (Activation-by-personalisation), where a DevAddr
   and
15 * Session keys are preconfigured (unlike OTAA, where a DevEUI and
16 * application key is configured, while the DevAddr and session keys
   are
17 * assigned/generated in the over-the-air-activation procedure).
18 *
19 * Note: LoRaWAN per sub-band duty-cycle limitation is enforced (1%

```

```

    in
20 * g1, 0.1% in g2), but not the TTN fair usage policy (which is
    probably
21 * violated by this sketch when left running for longer)!
22 *
23 * To use this sketch, first register your application and device
    with
24 * the things network, to set or generate a DevAddr, NwkSKey and
25 * AppSKey. Each device should have their own unique values for
    these
26 * fields.
27 *
28 * Do not forget to define the radio type correctly in config.h.
29 *
30
    *****
    */
31
32 #include <lmic.h>
33 #include <hal/hal.h>
34 #include "heltec.h"
35 #include "FastIMU.h"
36 #include "Wire.h"
37 #include "HardwareSerial.h"
38 #include "HT_TinyGPS++.h"
39
40 //Definicion de puertos seriales
41 #define RX_PIN 16
42 #define TX_PIN 17
43 #define BAUD_RATE 9600
44
45 //Definicion de pines I2C
46 #define IMU_ADDRESS 0x68
47

```

```

48 //Buffer de lectura de datos
49 #define BUFFER_LENGTH 10
50
51 // Constante de aceleracion de gravedad
52 #define G 9.805
53
54 #define BUFFER_SIZE 50
55
56 // LoRaWAN NwkSKey, network session key
57 // This is the default Semtech key, which is used by the early
    prototype TTN
58 // network.
59 static const PROGMEM u1_t NWKSKEY[16] = { 0x70, 0xFC, 0x6C, 0x66, 0
    x91, 0x34, 0x84, 0x3B, 0xD1, 0x84, 0xAA, 0xE5, 0x2E, 0xAB, 0x44,
    0xB1 };
60
61 // LoRaWAN AppSKey, application session key
62 // This is the default Semtech key, which is used by the early
    prototype TTN
63 // network.
64 static const u1_t PROGMEM APPSKEY[16] = { 0xE7, 0x09, 0xFC, 0x20, 0
    x0A, 0x4E, 0x1C, 0x3A, 0x25, 0x4C, 0x56, 0x20, 0x46, 0xFE, 0x13,
    0xD5 };
65
66 // LoRaWAN end-device address (DevAddr)
67 static const u4_t DEVADDR = 0x260C8E7B ; // <-- Change this address
    for every node!
68
69 // These callbacks are only used in over-the-air activation, so they
    are
70 // left empty here (we cannot leave them out completely unless
71 // DISABLE_JOIN is set in config.h, otherwise the linker will
    complain).
72 void os_getArtEui (u1_t* buf) { }

```

```

73 void os_getDevEui (ul_t* buf) { }
74 void os_getDevKey (ul_t* buf) { }
75
76 static uint8_t mydata[BUFFER_SIZE];
77 static osjob_t sendjob;
78 int16_t txNumber;
79
80 String txpacket = "";
81
82
83 //Definicion de parametros y variables para I2C (MPU9250)
84 MPU9250 IMU;
85
86 //Pines SCL y SDA
87 const int I2C_SDA = 21;
88 const int I2C_SCL = 22;
89
90 //Definicion de variables para la informacion de IMU
91 calData calib = { 0 };
92 AccelData accelData;
93
94 float accelXbuf[BUFFER_LENGTH] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
95 float accelYbuf[BUFFER_LENGTH] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
96 float accelZbuf[BUFFER_LENGTH] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
97 int writeIndex = 0;
98 float smoothX, smoothY, smoothZ, rollAcc, pitchAcc, absXY, accelZdeb
    ;
99
100 //Definicion de variables para GPS
101 TinyGPSPlus GPS;
102 bool gpsReady = false;
103 float speed;
104 float lat;
105 float lon;

```

```

106
107
108
109 // Schedule TX every this many seconds (might become longer due to
    duty
110 // cycle limitations).
111 const unsigned TX_INTERVAL = 2;
112
113 // Pin mapping
114 const lmic_pinmap lmic_pins = {
115     .nss = 18,
116     .rxtx = LMIC_UNUSED_PIN,
117     .rst = 14,
118     .dio = {26, 35, 34},
119 };
120
121 void onEvent (ev_t ev) {
122     Serial.print(os_getTime());
123     Serial.print(": ");
124     switch(ev) {
125         case EV_SCAN_TIMEOUT:
126             Serial.println(F("EV_SCAN_TIMEOUT"));
127             break;
128         case EV_BEACON_FOUND:
129             Serial.println(F("EV_BEACON_FOUND"));
130             break;
131         case EV_BEACON_MISSED:
132             Serial.println(F("EV_BEACON_MISSED"));
133             break;
134         case EV_BEACON_TRACKED:
135             Serial.println(F("EV_BEACON_TRACKED"));
136             break;
137         case EV_JOINING:
138             Serial.println(F("EV_JOINING"));

```

```

139         break;
140     case EV_JOINED:
141         Serial.println(F("EV_JOINED"));
142         break;
143     case EV_RFU1:
144         Serial.println(F("EV_RFU1"));
145         break;
146     case EV_JOIN_FAILED:
147         Serial.println(F("EV_JOIN_FAILED"));
148         break;
149     case EV_REJOIN_FAILED:
150         Serial.println(F("EV_REJOIN_FAILED"));
151         break;
152     case EV_TXCOMPLETE:
153         Serial.println(F("EV_TXCOMPLETE (includes waiting for RX
154 windows)"));
155         if (LMIC.txrxFlags & TXRX_ACK)
156             Serial.println(F("Received ack"));
157         if (LMIC.dataLen) {
158             Serial.println(F("Received "));
159             Serial.println(LMIC.dataLen);
160             Serial.println(F(" bytes of payload"));
161         }
162         // Schedule next transmission
163         os_setTimedCallback(&sendjob, os_getTime()+sec2osticks(
164 TX_INTERVAL), do_send);
165         break;
166     case EV_LOST_TSYNC:
167         Serial.println(F("EV_LOST_TSYNC"));
168         break;
169     case EV_RESET:
170         Serial.println(F("EV_RESET"));
171         break;
172     case EV_RXCOMPLETE:

```

```

171         // data received in ping slot
172         Serial.println(F("EV_RXCOMPLETE"));
173         break;
174     case EV_LINK_DEAD:
175         Serial.println(F("EV_LINK_DEAD"));
176         break;
177     case EV_LINK_ALIVE:
178         Serial.println(F("EV_LINK_ALIVE"));
179         break;
180     default:
181         Serial.println(F("Unknown event"));
182         break;
183     }
184 }
185
186 void do_send(osjob_t* j){
187     // Check if there is not a current TX/RX job running
188     if (LMIC.opmode & OP_TXRXPEND) {
189         Serial.println(F("OP_TXRXPEND, not sending"));
190     } else {
191         // Prepare upstream data transmission at the next possible
192         // time.
193
194         imuUpdate();
195         txpacket = "";
196
197         txpacket.concat(String(GPS.location.lat(), 6));
198         txpacket.concat(", ");
199         txpacket.concat(String(GPS.location.lng(), 6));
200         txpacket.concat(", ");
201         txpacket.concat(String(GPS.speed.mps(), 1));
202         txpacket.concat(", ");
203         txpacket.concat(String(rollAcc));
204         txpacket.concat(", ");

```

```

204     txpacket.concat(String(pitchAcc));
205     txpacket.concat(",");
206     txpacket.concat(String(txNumber));
207     txpacket.getBytes(mydata, txpacket.length() + 1);
208
209     LMIC_setTxData2(1, mydata, sizeof(mydata)-1, 0);
210
211     Serial.printf("\r\nPacket %d queued\n", txNumber);
212     Serial.printf("\r\nMessage: %s\n", mydata);
213     txNumber++;
214 }
215
216 // Next TX is scheduled after TX_COMPLETE event.
217 }
218
219
220 void setup() {
221
222     Serial.begin(115200);
223     //Tiempo para que se inicialice el puerto serial
224     while(!Serial);
225
226     //Inicializacion del segundo puerto serial (del GPS)
227     Serial2.begin(BAUD_RATE, SERIAL_8N1, RX_PIN, TX_PIN);
228     Serial.println("Serial2 inicializado. Esperando datos...");
229
230     Wire.begin(I2C_SDA, I2C_SCL);
231     Wire.setClock(400000);
232
233     imuInit();
234
235     txNumber = 0;
236
237

```

```

238 // LMIC init
239 os_init();
240 // Reset the MAC state. Session and pending data transfers will be
    discarded.
241 LMIC_reset();
242 // Set static session parameters. Instead of dynamically
    establishing a session
243 // by joining the network, precomputed session parameters are be
    provided.
244 #ifdef PROGMEM
245 // On AVR, these values are stored in flash and only copied to RAM
246 // once. Copy them to a temporary buffer here, LMIC_setSession
    will
247 // copy them into a buffer of its own again.
248 uint8_t appskey[sizeof(APPSKEY)];
249 uint8_t nwkskey[sizeof(NWKSKEY)];
250 memcpy_P(appskey, APPSKEY, sizeof(APPSKEY));
251 memcpy_P(nwkskey, NWKSKEY, sizeof(NWKSKEY));
252 LMIC_setSession(0x13, DEVADDR, nwkskey, appskey);
253 #else
254 // If not running an AVR with PROGMEM, just use the arrays
    directly
255 LMIC_setSession(0x13, DEVADDR, NWKSKEY, APPSKEY);
256 #endif
257 #if defined(CFG_eu868)
258 // Set up the channels used by the Things Network, which
    corresponds
259 // to the defaults of most gateways. Without this, only three base
260 // channels from the LoRaWAN specification are used, which
    certainly
261 // works, so it is good for debugging, but can overload those
262 // frequencies, so be sure to configure the full frequency range
    of
263 // your network here (unless your network autoconfigures them).

```

```

264 // Setting up channels should happen after LMIC_setSession, as
    that
265 // configures the minimal channel set.
266 // NA-US channels 0-71 are configured automatically
267 LMIC_setupChannel(0, 868100000, DR_RANGE_MAP(DR_SF12, DR_SF7),
    BAND_CENTI); // g-band
268 LMIC_setupChannel(1, 868300000, DR_RANGE_MAP(DR_SF12, DR_SF7B),
    BAND_CENTI); // g-band
269 LMIC_setupChannel(2, 868500000, DR_RANGE_MAP(DR_SF12, DR_SF7),
    BAND_CENTI); // g-band
270 LMIC_setupChannel(3, 867100000, DR_RANGE_MAP(DR_SF12, DR_SF7),
    BAND_CENTI); // g-band
271 LMIC_setupChannel(4, 867300000, DR_RANGE_MAP(DR_SF12, DR_SF7),
    BAND_CENTI); // g-band
272 LMIC_setupChannel(5, 867500000, DR_RANGE_MAP(DR_SF12, DR_SF7),
    BAND_CENTI); // g-band
273 LMIC_setupChannel(6, 867700000, DR_RANGE_MAP(DR_SF12, DR_SF7),
    BAND_CENTI); // g-band
274 LMIC_setupChannel(7, 867900000, DR_RANGE_MAP(DR_SF12, DR_SF7),
    BAND_CENTI); // g-band
275 LMIC_setupChannel(8, 868800000, DR_RANGE_MAP(DR_FSK, DR_FSK),
    BAND_MILLI); // g2-band
276 // TTN defines an additional channel at 869.525Mhz using SF9 for
    class B
277 // devices' ping slots. LMIC does not have an easy way to define
    set this
278 // frequency and support for class B is spotty and untested, so
    this
279 // frequency is not configured here.
280 #elif defined(CFG_us915)
281 // NA-US channels 0-71 are configured automatically
282 // but only one group of 8 should (a subband) should be active
283 // TTN recommends the second sub band, 1 in a zero based count.
284 // https://github.com/TheThingsNetwork/gateway-conf/blob/master/

```

```

    US-global_conf.json
285 LMIC_selectSubBand(1);
286 #endif
287
288 // Disable link check validation
289 LMIC_setLinkCheckMode(0);
290 // TTN uses SF9 for its RX2 window.
291 LMIC.dn2Dr = DR_SF9;
292 // Set data rate and transmit power for uplink (note: txpow seems
    to be ignored by the library)
293 LMIC_setDrTxpow(DR_SF10,22);
294 // Start job
295 do_send(&sendjob);
296 }
297
298 void loop() {
299
300     imuUpdate();
301     gpsUpdate();
302
303     if(Serial2.available()){
304         if(Serial2.peek() != '\n') {
305             GPS.encode(Serial2.read());
306         }
307         else{
308             os_runloop_once();
309         }
310     }
311 }
312
313
314
315 // Realiza la inicializacion de la IMU. Setea rangos de giroscopio y
    acelerometro.

```

```

316 void imuInit(){
317     //Inicializacion de la IMU
318     int err = IMU.init(calib, IMU_ADDRESS);
319     if (err != 0) {
320         Serial.print("Error initializing IMU: ");
321         Serial.println(err);
322         while (true) {
323             ;
324         }
325     }
326
327     //Configuracion de los rangos de la IMU
328     err = IMU.setGyroRange(250);          //USE THESE TO SET THE RANGE, IF
        AN INVALID RANGE IS SET IT WILL RETURN -1
329     err = IMU.setAccelRange(8);          //THESE TWO SET THE GYRO RANGE
        TO 500 DPS AND THE ACCELEROMETER RANGE TO 2g
330     if (err != 0) {
331         Serial.print("Error Setting range: ");
332         Serial.println(err);
333         while (true) {
334             ;
335         }
336     }
337 }
338
339 // Actualizacion de la IMU, ajuste de las lecturas segun sesgos de
        las mediciones, calculo de Pitch y Roll.
340 void imuUpdate(){
341     //Obtencion de datos de IMU MPU9250
342     IMU.update();
343     IMU.getAccel(&accelData);
344
345     //Calibracion manual
346     accelData.accelX = (accelData.accelX + 1.3) * G;    //AccelX bias

```

```

347 accelData.accelY = (accelData.accelY + 0.96) * G; //AccelY bias
348 accelData.accelZ = (accelData.accelZ + 0.26) * G; //AccelZ bias
349
350 //Buffer aceleraciones
351 accelXbuf[writeIndex] = accelData.accelX;
352 accelYbuf[writeIndex] = accelData.accelY;
353 accelZbuf[writeIndex] = accelData.accelZ;
354
355 writeIndex = (writeIndex + 1) % BUFFER_LENGTH;
356
357 for(int i = 0; i < 10; i++){
358     smoothX += accelXbuf[i];
359     smoothY += accelYbuf[i];
360     smoothZ += accelZbuf[i];
361 }
362
363 smoothX = smoothX/BUFFER_LENGTH;
364 smoothY = smoothY/BUFFER_LENGTH;
365 smoothZ = smoothZ/BUFFER_LENGTH;
366
367
368 //Calculo de Roll, Yaw y Pitch
369 getRollPitch(smoothX, smoothY, smoothZ);
370 absXY = sqrt(smoothX * smoothX + smoothY * smoothY);
371 accelZdeb = smoothZ - 1.0;
372
373
374 }
375
376 // Actualizacion del GPS.
377 void gpsUpdate() {
378     //Se revisa puerto serial para obtener informacion del GPS.
379     if(Serial2.available()){
380         if(Serial2.peek() != '\n') {

```

```

381     GPS.encode(Serial2.read());
382 }
383 else{
384     gpsReady = true;
385     speed = GPS.speed.mps();
386     lat = GPS.location.lat();
387     lon = GPS.location.lng();
388 }
389
390 }
391 }
392
393 // Calculo del Roll y Pitch del cuerpo.
394 void getRollPitch(float accelX, float accelY, float accelZ){
395     pitchAcc = atan2(accelY, accelZ) * 180 / PI;
396     rollAcc = atan2(-accelX, sqrt(accelY * accelY + accelZ * accelZ))
        * 180 / PI;
397 }
398
399 void printData(){
400     Serial.print(rollAcc); Serial.print(",");
401     Serial.print(pitchAcc); Serial.print(",");
402     Serial.print(smoothX); Serial.print(",");
403     Serial.print(smoothY); Serial.print(",");
404     Serial.print(smoothZ); Serial.print(",");
405     Serial.print(speed); Serial.print(",");
406     Serial.print(absXY); Serial.print(",");
407     Serial.print(accelZdeb);
408     Serial.println();
409 }

```

Listing A.2: Código .ino para nodo transmisor escenario 2

```

1 #include <WiFi.h>
2 #include <PubSubClient.h>

```

```

3 #include <Wire.h>
4 #include "LoRaWan_APP.h"
5 #include "Arduino.h"
6 #include "string.h"
7 #include "stdio.h"
8 #include "HT_SSD1306Wire.h"
9
10 extern SSD1306Wire display;
11
12 #define RF_FREQUENCY 904300000 //Hz
13 #define TX_OUTPUT_POWER 14 //dBm
14 #define LORA_BANDWIDTH 0 // [0:125kHz,
15 // 1:250kHz,
16 // 2:500kHz,
17 // 3:Reserved]
18 #define LORA_SPREADING_FACTOR 7 // [SF7..SF12]
19 #define LORA_CODINGRATE 1 // [1:4/5,
20 // 2:4/6,
21 // 3:4/7,
22 // 4:4/8]
23 #define LORA_PREAMBLE_LENGTH 8 //SameforTxandRx
24 #define LORA_SYMBOL_TIMEOUT 0 //Symbols
25 #define LORA_FIX_LENGTH_PAYLOAD_ON false
26 #define LORA_IQ_INVERSION_ON false
27 #define RX_TIMEOUT_VALUE 1000
28 #define BUFFER_SIZE 256 //
29 //Definethepayloadsizehere
30 char txpacket[BUFFER_SIZE];
31 char rxpacket[BUFFER_SIZE];
32
33 const char ssid[] = ""; //yournetworkSSID(name)
34 const char password[] = ""; //yournetworkpassword
35 const char* mqtt_server = "";

```

```

36
37 WiFiClient espClient;
38 PubSubClient client(espClient);
39
40 static RadioEvents_t RadioEvents;
41 int16_t txNumber, rxSize;
42
43 int16_t rssi;
44 int16_t snr;
45 char buffer[1024];
46 bool lora_idle = true;
47
48 void setup() {
49     Mcu.begin(HELTEC_BOARD, SLOW_CLK_TPYE);
50     Serial.begin(115200);
51     setup_wifi();
52     client.setServer(mqtt_server, 1883);
53     client.setCallback(callback);
54
55     txNumber=0;
56     rssi=0;
57
58     RadioEvents.RxDone = OnRxDone;
59     Radio.Init( &RadioEvents );
60     Radio.SetChannel( RF_FREQUENCY );
61     Radio.SetRxConfig( MODEM_LORA, LORA_BANDWIDTH,
62         LORA_SPREADING_FACTOR,
63         LORA_CODINGRATE, 0,
64         LORA_PREAMBLE_LENGTH,
65         LORA_SYMBOL_TIMEOUT,
66         LORA_FIX_LENGTH_PAYLOAD_ON,
67         0, true, 0, 0, LORA_IQ_INVERSION_ON,
68         true );
69 }

```

```

66
67 void loop() {
68     if (!client.connected()) {
69         reconnect();
70     }
71     client.loop();
72
73     if (lora_idle)
74     {
75         lora_idle = false;
76         Serial.println("Entering RX mode... ");
77         Radio.Rx(0);
78     }
79     Radio.IrqProcess( );
80
81     delay(100);
82 }
83
84
85 void setup_wifi() {
86     delay(10);
87     Serial.println();
88     Serial.print("Connecting to ");
89     Serial.println(ssid);
90
91     WiFi.begin(ssid, password);
92
93     while (WiFi.status() != WL_CONNECTED) {
94         delay(500);
95         Serial.print(".");
96     }
97
98     Serial.println("");
99     Serial.println("WiFi connected");

```

```

100 Serial.println("IP address: ");
101 Serial.println(WiFi.localIP());
102 }
103
104 void callback(char* topic, byte* message, unsigned int length) {
105     Serial.print("Message arrived on topic: ");
106     Serial.print(topic);
107     Serial.print(". Message: ");
108     String messageTemp;
109
110     for(int i=0; i<length; i++){
111         Serial.print((char)message[i]);
112         messageTemp += (char)message[i];
113     }
114     Serial.println();
115
116     //If a message is received on the topic esp32/output,
117     //you check if the message is either "on" or "off".
118     //Changes the output state according to the message
119     if(String(topic) == "esp32/output") {
120         Serial.print("Changing output to ");
121         if(messageTemp == "on"){
122             Serial.println("on");
123         }
124         else if(messageTemp == "off"){
125             Serial.println("off");
126         }
127     }
128 }
129
130 void reconnect() {
131     // Loop until we're reconnected
132     while (!client.connected()) {
133         Serial.print("Attempting MQTT connection...");

```

```

134 //Attempt to connect
135 if (client.connect("ESP32_Client")) {
136     Serial.println("connected");
137 } else {
138     Serial.print("failed, rc=");
139     Serial.print(client.state());
140     Serial.println(" try again in 5 seconds");
141     //wait 5 seconds before retrying
142     delay(5000);
143 }
144 }
145 }
146
147 void OnRxDone( uint8_t *payload, uint16_t size, int16_t rssi, int8_t
    snr )
148 {
149     rssi=rssi;
150     snr = snr;
151     rxSize=size;
152     memcpy(rxpacket, payload, size );
153     rxpacket[size]='\0';
154     Radio.Sleep( );
155     Serial.printf("\r\nreceived packet \"%s\" with rssi %d , snr %d,
        length %d\r\n",rxpacket,rssi,snr,rxSize);
156
157
158     snprintf(buffer, sizeof(buffer), "%s,%d,%d", rxpacket, rssi, snr
    );
159     client.publish("esp32/output", buffer);
160
161     lora_idle = true;
162 }

```

Listing A.3: Código .ino para nodo receptor escenario 1

```

1 package com.example.sensorapp
2
3 import android.Manifest
4 import android.content.pm.PackageManager
5 import android.hardware.Sensor
6 import android.hardware.SensorEvent
7 import android.hardware.SensorEventListener
8 import android.hardware.SensorManager
9 import android.location.Location
10 import android.location.LocationListener
11 import android.location.LocationManager
12 import android.os.Bundle
13 import android.os.Handler
14 import android.os.Looper
15 import android.util.Log
16 import android.widget.TextView
17 import androidx.appcompat.app.AppCompatActivity
18 import androidx.core.app.ActivityCompat
19 import okhttp3.*
20 import java.io.IOException
21 import java.util.concurrent.TimeUnit
22
23 class MainActivity : AppCompatActivity(), SensorEventListener,
    LocationListener {
24     private lateinit var locationManager: LocationManager
25     private lateinit var sensorManager: SensorManager
26     private var rotationVectorSensor: Sensor? = null
27
28     private val rotationMatrix = FloatArray(9)
29     private val orientationAngles = FloatArray(3)
30
31     private var roll: Float = 0.0f
32     private var pitch: Float = 0.0f
33     private var latitude: Double? = null

```

```

34     private var longitude: Double? = null
35     private var speed: Float = 0.0f
36     private var txNumber: Int = 0
37
38
39     private lateinit var textView: TextView
40     private val handler = Handler(Looper.getMainLooper())
41     private val apiKey = "" // Reemplaza con tu clave de ThingSpeak
42     private val thingSpeakUrl = "https://api.thingspeak.com/update"
43
44     override fun onCreate(savedInstanceState: Bundle?) {
45         super.onCreate(savedInstanceState)
46         setContentView(R.layout.activity_main)
47
48         textView = findViewById(R.id.textView)
49         locationManager = getSystemService(LOCATION_SERVICE) as
LocationManager
50         sensorManager = getSystemService(SENSOR_SERVICE) as
SensorManager
51         rotationVectorSensor = sensorManager.getDefaultSensor(Sensor
.TYPE_ROTATION_VECTOR)
52
53
54         // Verificar permisos de ubicaci n
55         if (ActivityCompat.checkSelfPermission(this, Manifest.
permission.ACCESS_FINE_LOCATION) != PackageManager.
PERMISSION_GRANTED) {
56             ActivityCompat.requestPermissions(this, arrayOf(Manifest
.permission.ACCESS_FINE_LOCATION), 1)
57             return
58         }
59
60         //locationManager.requestLocationUpdates(LocationManager.
GPS_PROVIDER, 1000, 1f, this)

```

```

61         // Solicitar ubicaci3n de diferentes proveedores
62         try {
63             // GPS: Mayor precisi3n pero m3s lento
64             if (locationManager.isProviderEnabled(LocationManager.
GPS_PROVIDER)) {
65                 locationManager.requestLocationUpdates(
66                     LocationManager.GPS_PROVIDER, 1000, 1f, this
67                 )
68             }
69
70             // Red: M3s rpido pero menos preciso
71             if (locationManager.isProviderEnabled(LocationManager.
NETWORK_PROVIDER)) {
72                 locationManager.requestLocationUpdates(
73                     LocationManager.NETWORK_PROVIDER, 1000, 1f, this
74                 )
75             }
76         } catch (e: SecurityException) {
77             e.printStackTrace()
78         }
79
80
81         // Enviar datos a ThingSpeak cada 15 segundos
82         handler.postDelayed(object : Runnable {
83             override fun run() {
84                 enviarDatosAThingSpeak()
85                 handler.postDelayed(this, 15000)
86             }
87         }, 15000)
88     }
89
90     override fun onResume() {
91         super.onResume()
92         rotationVectorSensor?.let { sensor ->

```

```

93         sensorManager.registerListener(this, sensor,
SensorManager.SENSOR_DELAY_UI)
94     }
95 }
96
97 override fun onPause() {
98     super.onPause()
99     sensorManager.unregisterListener(this)
100 }
101
102 override fun onSensorChanged(event: SensorEvent?) {
103     if (event?.sensor?.type == Sensor.TYPE_ROTATION_VECTOR) {
104         SensorManager.getRotationMatrixFromVector(rotationMatrix
, event.values)
105         SensorManager.getOrientation(rotationMatrix,
orientationAngles)
106
107         pitch = Math.toDegrees(orientationAngles[1].toDouble()).
toFloat()
108         roll = Math.toDegrees(orientationAngles[2].toDouble()).
toFloat()
109     }
110     actualizarTexto()
111 }
112
113 override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int)
{}
114
115 override fun onLocationChanged(location: Location) {
116     latitude = location.latitude
117     longitude = location.longitude
118     speed = location.speed
119     actualizarTexto()
120 }

```

```

121
122     private fun actualizarTexto() {
123         runOnUiThread {
124             textView.text = "Pitch: $pitch\nRoll: $roll\nSpeed:
125             $speed\nLatitud: ${latitude ?: "N/A"}\nLongitud: ${longitude ?:
126             "N/A"}\nTx N : $txNumber"
127         }
128     }
129
130     private fun enviarDatosAThingSpeak() {
131         if (latitude != null && longitude != null) {
132             val url = "$thingSpeakUrl?api_key=$apiKey&field1=
133             $latitude&field2=$longitude&field3=$pitch&field4=$roll&field5=
134             $speed&field6=$txNumber"
135
136             val request = Request.Builder().url(url).build()
137             val client = OkHttpClient.Builder().connectTimeout(10,
138             TimeUnit.SECONDS).build()
139             txNumber++
140
141             client.newCall(request).enqueue(object : Callback {
142                 override fun onFailure(call: Call, e: IOException) {
143                     Log.e("ThingSpeak", "Error al enviar datos: ${e.
144                     message}")
145                 }
146
147                 override fun onResponse(call: Call, response:
148                 Response) {
149                     if (response.isSuccessful) {
150                         Log.d("ThingSpeak", "Datos enviados
151                         correctamente")
152                     }
153                 }
154             })
155         }
156     }

```

```
147     }  
148 }
```

Listing A.4: Código en Kotlin para aplicación Android