UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE MÁSTER

Desarrollo de una plataforma HW/SW basada en las tecnologías VLC, BLE y LoRa/LoRaWAN para aplicaciones IoT

Titulación: Máster Universitario en Ingeniería de Telecomunicación

Autor: D. Luis González Álvarez

Tutores: D. Valentín De Armas Sosa
D. José A. Rabadán Borges
D. Félix B. Tobajas Guerrero

Fecha: Marzo de 2020

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE MÁSTER

Desarrollo de una plataforma HW/SW basada en las tecnologías VLC, BLE y LoRa/LoRaWAN para aplicaciones IoT

HOJA DE EVALUACIÓN

Calificación: _____

Presidente

Fdo.: _____

Vocal **Secretario/a**

Fdo.: _____ Fdo.: _____

Fecha: Marzo de 2020

Agradecimientos

A toda mi familia y amigos, por el apoyo incondicional.

A todos mis educadores, por formarme como persona y como estudiante.

A mis tutores Félix, Valentín y José, por confiar en mí y siempre estar presentes.

A Ramón, Marcos, Fátima y Cruci, por abrirme el laboratorio.

A todos y cada uno de los compañeros que he tenido durante la carrera.

¡Muchísimas gracias a todos!

Índice

MEMORIA **1**

CAPÍTULO 1. INTRODUCCIÓN **1**

1.1 OBJETIVOS 4

1.2 PETICIONARIO 6

1.3 ESTRUCTURA DEL DOCUMENTO 6

CAPÍTULO 2. ANTECEDENTES **9**

2.1 *INTERNET OF THINGS* 9

2.1.1 Concepto *9*

2.1.2 Impacto y crecimiento de IoT *11*

2.2 ESTADO DEL ARTE DE LAS TECNOLOGÍAS *BLE*, *VLC* Y *LORA/LORAWAN* 17

2.2.1 Bluetooth Low Energy (BLE) *17*

2.2.2 Visible Light Communication (VLC) *22*

2.2.3 LoRa/LoRaWAN *26*

2.2.4 Integración de Visible Light Communication y LoRa/LoRaWAN *30*

2.3 *SMART CITIES* 31

2.3.1 Origen *31*

2.3.2 Definición de Smart City *36*

2.4 *SMART MOBILITY Y SMART PARKING* 42

2.4.1 Estado de la movilidad *43*

2.4.2 Smart Parking *44*

2.5 DESCRIPCIÓN DE LA PLATAFORMA HW/SW PROPUESTA 50

CAPÍTULO 3. VLC **53**

3.1 *VISIBLE LIGHT COMMUNICATION*: DESCRIPCIÓN 53

3.1.1 Definición *53*

3.1.2 Capa física: Modulación *58*

3.1.3 Características técnicas y estándar *62*

3.1.4 Ventajas y desventajas *65*

3.2 IMPLEMENTACIÓN DE *VLC* EN LA PLATAFORMA HW/SW 68

3.2.1 Enlace 1: Transmisor en nodo estación base y receptor en nodo móvil *68*

3.2.1.1 Planteamiento inicial: Montaje 1 69

3.2.1.2 Aumento del alcance: Montaje 2 72

3.2.1.3 Aumento de la frecuencia: Montaje 3 80

3.2.1.4 Integración con dispositivo LoPy: Montaje 4 84
3.2.2 Enlace 2: Transmisor en nodo móvil y receptor en nodo estación base 101
3.2.2.1 Planteamiento inicial: Montaje 1 102
3.2.2.2 Planteamiento final e integración con el dispositivo LoPy 106
3.2.3 Presentación y análisis de los circuitos desarrollados 110
3.2.3.1 Enlace 1 110
3.2.3.2 Enlace 2 111

CAPÍTULO 4. BLE 113

4.1 *BLUETOOTH LOW ENERGY*: DESCRIPCIÓN 113
4.1.1 Definición 113
4.1.2 Antecedentes 114
4.1.3 Impacto actual de BLE 115
4.1.4 Capa física: Modulación 116
4.1.5 Protocolos de comunicación 118
4.2 IMPLEMENTACIÓN DE *BLE* EN LA PLATAFORMA HW/SW 123
4.2.1 Transmisor-Peripheral 123
4.2.2 Receptor-Central 125
4.2.3 Resultado de la comunicación 126

CAPÍTULO 5. LORA/LORAWAN 129

5.1 *LORA/LORAWAN*: DESCRIPCIÓN 129
5.1.1 Introducción 130
5.1.2 Comparación con otros estándares 130
5.1.3 LoRa 133
5.1.4 LoRaWAN 134
5.1.5 Topología 135
5.1.6 Características técnicas 139
5.1.6.1 Modulación 139
5.1.6.2 Estudio de parámetros asociados 141
5.1.6.3 Estructura de paquetes 144
5.1.6.3.a Paquete físico: LoRa 144
5.1.6.3.b Paquete MAC: *LoRaWAN* 147
5.1.6.4 Acceso a *LoRaWAN* 151
5.1.6.5 Seguridad 153
5.2 IMPLEMENTACIÓN DE *LORA/LORAWAN* EN LA PLATAFORMA HW/SW 155
5.2.1 Implementación de LoRa/LoRaWAN en el nodo estación base 155
5.2.2 Prueba de la comunicación LoRa/LoRaWAN 158

CAPÍTULO 6. NANOGATEWAY 161

6.1 ARCHIVO: NANOGATEWAY.PY 164
6.2 ARCHIVO: CONFIG.PY 178
6.3 ARCHIVO: MAIN.PY 179

CAPÍTULO 7. ESTACIÓN BASE 181

CAPÍTULO 8. NODO FINAL 191

CAPÍTULO 9. THE THINGS NETWORK 199

CAPÍTULO 10. USE OF CASE: SMART PARKING 205

10.1 ADAPTACIÓN DEL NODO FINAL. 208
10.1.1 Integración del sensor 209
10.1.2 Integración de la batería 213
10.1.3 Diseño y desarrollo de la funcionalidad del nodo final 214
10.1.4 Formato de los paquetes 217
10.1.5 Adaptación del firmware del nodo final 222
10.2 ADAPTACIÓN DEL NODO ESTACIÓN BASE Y DEL NODO *GATEWAY* 230
10.3 ADAPTACIÓN DEL SERVICIO *THE THINGS NETWORK* 232
10.4 VALIDACIÓN DE LA PLATAFORMA HW/SW SOBRE EL *USE OF CASE* 235

CAPÍTULO 11. CONCLUSIONES Y LÍNEAS FUTURAS 243

11.1 CONCLUSIONES 243
11.2 LÍNEAS FUTURAS 246

BIBLIOGRAFÍA 249

PLIEGO DE CONDICIONES 259

PC.1 CONDICIONES HARDWARE 259
PC.2 CONDICIONES SOFTWARE 260
PC.3 CONDICIONES FIRMWARE 261

PRESUPUESTO 263

P.1 TRABAJO TARIFADO POR TIEMPO EMPLEADO 263
P.2 AMORTIZACIÓN DEL INMOVILIZADO MATERIAL 265
P.2.1 Amortización del material hardware 265
P.2.2 Amortización del material software 266
P.3 REDACCIÓN DEL TRABAJO 267
P.4 DERECHOS DE VISADO DEL COIT 268
P.5 GASTOS DE TRAMITACIÓN Y ENVÍO 270
P.6 MATERIAL FUNGIBLE 270
P.7 APLICACIÓN DE IMPUESTOS Y COSTE TOTAL 270

ANEXO 273

ANEXO A. CONTENIDO DE LA DOCUMENTACIÓN DEL TFM. 275

Índice de Figuras

Figura 1.1: Número de dispositivos conectados a Internet. 2
Figura 2.1: Número de conexiones activas a nivel mundial desde 2015 a 2025. 11
Figura 2.2: Arquitectura genérica de IoT. 13
Figura 2.3: Análisis de diferentes sensores destinados a Smart Parking. 14
Figura 2.4: Desglose de tecnologías utilizadas en las conexiones IoT. 15
Figura 2.5: Radar de tecnologías empleadas en IoT. 16
Figura 2.6: Diagrama de funcionamiento del sistema de control de asistencia por BLE. 18
Figura 2.7: Porcentaje de comunicaciones BLE con acierto y error en un vehículo. 20
Figura 2.8: Triangulación de un vehículo a partir de cuatro radios BLE. 20
Figura 2.9: Coexistencia de ZigBee, KNX-RF, EnOcean y BLE en aplicación Smart Home. 21
Figura 2.10: Sistema de iluminación y conectividad en entorno de oficina. 22
Figura 2.11: Estudio de comunicación VLC en un habitáculo. 23
Figura 2.12: Resultados de la simulación. SNR y BER para los tres DUT. 24
Figura 2.13: Relación velocidad del vehículo-distancia de seguridad-ángulo de comunicación VLC. 24
Figura 2.14: Comparativa de tensión recibida en agua del grifo y en agua mineral. 25
Figura 2.15: Estructura de comunicaciones en BeitMisk. 27
Figura 2.16: Dashboard web de la solución Smart Environment de BeitMisk. 29
Figura 2.17: Arquitectura utilizada por la solución InteliLIGHT. 29
Figura 2.18: Arquitectura de interfaz VLC-LoRa. 30
Figura 2.19: Estudio de población rural y urbana en las grandes regiones. 33
Figura 2.20: Porcentaje de urbanización y nivel de población de las ciudades a nivel mundial. 34
Figura 2.21: Nivel de población en los diferentes tipos de ciudades. 35
Figura 2.22: Smart City como sistema de subsistemas. 38
Figura 2.23: Heathrow pod. 39
Figura 2.24: Contenedores de basura inteligentes. 41
Figura 2.25: Arquitectura de SmartSantander. 42
Figura 2.26: Porcentaje de Km recorridos al año por cada tipo de medio de transporte en Madrid. 44
Figura 2.27: Beneficios directos de Smart Parking. 47
Figura 2.28: SmartRep. Software de aparcamiento inteligente. 48
Figura 2.29: Arquitectura de 3SCPARK. 48
Figura 2.30: Arquitectura de un sistema Smart Parking Indoor. 50
Figura 2.31: Arquitectura de la plataforma HW/SW propuesta. 51

Figura 3.1: Ubicación del espectro visible en el espectro electromagnético. 54
Figura 3.2: Diagrama de bloques genérico de una comunicación VLC. 54
Figura 3.3: Modos de transmisión. 58
Figura 3.4: Modulación OOK con código Manchester. 59
Figura 3.5: Modulación 4-PPM. 59
Figura 3.6: Modulación D-PPM. 60
Figura 3.7: Modulación I-PPM. 60
Figura 3.8: Modulación V-PPM. 61
Figura 3.9: Espacio de color CIE 1931. 62
Figura 3.10: Circuito transmisor para el montaje 1. 69
Figura 3.11: Lámpara de 5V conectada al circuito transmisor del montaje 1. 70
Figura 3.12: Circuito receptor para el montaje 1. 71
Figura 3.13: Circuito transmisor para el montaje 2. 73
Figura 3.14: Segundo montaje completo. 73
Figura 3.15: Relación distancia - tensión recibida para el montaje 2. 75
Figura 3.16: Señal recibida para una distancia de 1m con el montaje 2. 75
Figura 3.17: Enlace de 2.5 m usando lámpara de 12 V. 76
Figura 3.18: Circuito receptor de salida digital para el montaje 2. 78
Figura 3.19: Señal de salida del módulo digital para el montaje 2. 79
Figura 3.20: Circuito receptor para el montaje 3. 81
Figura 3.21: Señal digital y señal analógica para valor de potenciómetro de 1.5 KΩ. 82
Figura 3.22: Señal digital y señal analógica para valor de potenciómetro de 2 KΩ. 83
Figura 3.23: Señal digital y señal analógica para valor de potenciómetro de 1 KΩ. 83
Figura 3.24: LoPy de Pycom. 84
Figura 3.25: Diagrama de bloques del dispositivo LoPy. 86
Figura 3.26: Pinout del dispositivo LoPy. 87
Figura 3.27: Expansion Board. 88
Figura 3.28: Diagrama de bloques. 89
Figura 3.29: Utilidades de la Expansion Board. 89
Figura 3.30: Interfaz principal del entorno Atom. 91
Figura 3.31: Pinout del dispositivo Bi-directional Logic Level Converter. 92
Figura 3.32: Circuito interno de cada canal de conversión. 93
Figura 3.33: Conexionado LoPy-Level Converter-circuito transmisor VLC. 94
Figura 3.34: Leyenda asociada a los pines del conexionado LoPy-Level Converter-circuito transmisor VLC. .. 94
Figura 3.35: Primera versión del circuito transmisor VLC para la integración con LoPy. 94
Figura 3.36: Circuito transmisor VLC utilizado para la integración con LoPy. 96
Figura 3.37: Circuito receptor VLC utilizado para la integración con LoPy. 96
Figura 3.38: Codificación de la funcionalidad del transmisor VLC. 98

Figura 3.39: Codificación de la funcionalidad del receptor VLC. *98*
Figura 3.40: Resultado de la correcta comunicación entre transmisor y receptor VLC. *99*
Figura 3.41: Montaje final del enlace 1. ..... *100*
Figura 3.42: Señal a la salida del dispositivo Bi-directional Logic Level Converter. *100*
Figura 3.43: Señal a la salida del comparador en el circuito receptor VLC. *101*
Figura 3.44: Circuito transmisor inicial para el enlace 2. ..... *102*
Figura 3.45: Utilización de varios diodos LED para la transmisión del enlace 2. ..... *103*
Figura 3.46: Circuito receptor inicial para el enlace 2. ..... *104*
Figura 3.47: Montaje 1 del enlace 2. ..... *105*
Figura 3.48: Señal recibida en el circuito receptor inicial del enlace 2. ..... *106*
Figura 3.49: Circuito transmisor utilizado para el enlace 2. ..... *107*
Figura 3.50: Circuito receptor utilizado en el enlace 2. ..... *108*
Figura 3.51: Codificación de la funcionalidad del transmisor en el enlace 2. ..... *109*
Figura 3.52: Codificación de la funcionalidad del receptor en el enlace 2. ..... *109*
Figura 3.53: Respuesta obtenida en el receptor del enlace 2. ..... *110*
Figura 3.54: Circuito transmisor VLC para el enlace 1. ..... *110*
Figura 3.55: Circuito receptor VLC para el enlace 1. ..... *111*
Figura 3.56: Montaje completo del enlace 1. ..... *111*
Figura 3.57: Circuito transmisor VLC para el enlace 2. ..... *112*
Figura 3.58: Montaje completo del enlace 2. ..... *112*
Figura 4.1: Número de dispositivos Bluetooth. *115*
Figura 4.2: Modulación FHSS. *117*
Figura 4.3: Espectro utilizado por BLE. *117*
Figura 4.4: Protocolo de establecimiento de comunicación BLE. *118*
Figura 4.5: Protocolo de comunicación de perfil GATT. *120*
Figura 4.6: Máquina de estado de un dispositivo BLE Central. *120*
Figura 4.7: Máquina de estado de un dispositivo BLE Peripheral. *121*
Figura 4.8: Arquitectura de perfil GATT. *121*
Figura 4.9: Topología broadcast. *123*
Figura 4.10: Codificación de la funcionalidad BLE del transmisor. *124*
Figura 4.11: Codificación de la funcionalidad BLE del receptor. *126*
Figura 4.12: Resultado de la comunicación BLE. *127*
Figura 5.1: Comparativa por alcance y consumo de potencia. *132*
Figura 5.2: Modulación de espectro ensanchado. *134*
Figura 5.3: Topología: "Estrella de estrellas". *136*
Figura 5.4: Estudio temporal de un nodo final de clase A. *137*
Figura 5.5: Estudio temporal de un nodo final de clase B. *138*
Figura 5.6: Estudio temporal de un nodo final de clase C. *138*

Figura 5.7: Arquitectura LoRa/LoRaWAN. *139*
Figura 5.8: Chirp en el espectro y en el eje temporal. *140*
Figura 5.9: Pulsos Upchirp y Downchirp. *140*
Figura 5.10: Ejemplo para el estudio de los parámetros de transmisión. *143*
Figura 5.11: Realación SF-BitRate-Alcance-Tiempo de transmisión. *144*
Figura 5.12: Formato de paquete LoRa. *144*
Figura 5.13: Formato de paquete LoRaWAN. *148*
Figura 5.14: Encapsulado de las tramas del paquete LoRaWAN. *150*
Figura 5.15: Relaciones entre los elementos típicos de una red IoT basada en LoRa. *155*
Figura 5.16: Modo de funcionamiento LoRa/LoRaWAN y claves de activación OTAA. *157*
Figura 5.17: Join-request, eliminación de canales no prestablecidos y configuración del socket. *158*
Figura 5.18: Bloqueo y desbloqueo del socket. *158*
Figura 5.19: Codificación de la funcionalidad de la estación base. *159*
Figura 5.20: Resultado de la prueba de comunicación LoRa/LoRaWAN. *159*
Figura 6.1: Arquitectura de la plataforma HW/SW propuesta: Elemento Gateway. *162*
Figura 6.2: Dispositivo LoPy utilizado como nanogateway. *163*
Figura 6.3: Librerías utilizadas en nanogateway.py. *164*
Figura 6.4: Variables utilizadas en nanogateway.py. *165*
Figura 6.5: Estructura STAT_PK. *166*
Figura 6.6: Estructuras RX_PK y TX_ACK_PK. *166*
Figura 6.7: Creación de la clase NanoGateway y de la función __init__(). *167*
Figura 6.8: Función __init__()_2. ..... *168*
Figura 6.9: Conversiones Datarate-SF/BW. *169*
Figura 6.10: Función _log(). *169*
Figura 6.11: Función start(). *171*
Figura 6.12: Función start()_2. ..... *171*
Figura 6.13: Función _connect_to_WiFi(). *172*
Figura 6.14: Función _make_stat_packet(). *172*
Figura 6.15: Función _push_data() y _pull_data(). *172*
Figura 6.16: Función _udp_thread(). *173*
Figura 6.17: Función _udp_thread()_2. ..... *174*
Figura 6.18: Función _udp_thread()_3. ..... *175*
Figura 6.19: Función _lora_cb(). *175*
Figura 6.20: Función _freq_to_float(). *176*
Figura 6.21: Función _make_node_packet(). *176*
Figura 6.22: Función _send_down_link(). *177*
Figura 6.23: Función _ack_pull_rsp(). *177*
Figura 6.24: Función stop(). *178*

Figura 6.25: config.py. *179*
Figura 6.26: main.py. *180*
Figura 7.1: Arquitectura de la plataforma HW/SW propuesta: Elemento Estación Base. *181*
Figura 7.2: Librerías utilizadas en el nodo estación base. *182*
Figura 7.3: Configuración de LoRa/LoRaWAN en el nodo estación base. *183*
Figura 7.4: Configuración de socket LoRa/LoRaWAN en el nodo estación base. *183*
Figura 7.5: Configuración de comunicación VLC y BLE en el nodo estación base. *184*
Figura 7.6: Declaración de bytearrays y de funciones clearframe() y clear_bytearr(). *185*
Figura 7.7: Funciones send(), send_VLC() y send_BLE(). *186*
Figura 7.8: Funciones receive() y receive_VLC(). *187*
Figura 7.9: Función receive_BLE(). *187*
Figura 7.10: Funciones send_LoRa() y receive_LoRa(). *188*
Figura 7.11: Bucle infinito del nodo estación base. *188*
Figura 8.1: Arquitectura de la plataforma HW/SW propuesta: Elemento Nodo Final. *191*
Figura 8.2: Librerías utilizadas en el nodo final. *192*
Figura 8.3: Configuración VLC y BLE en el nodo final. *193*
Figura 8.4: Declaración de bytearrays y de funciones clearframe() y clear_bytearr(). *194*
Figura 8.5: Funciones send(), send_VLC() y send_BLE(). *195*
Figura 8.6: Funciones receive() y receive_VLC(). *196*
Figura 8.7: Función receive_BLE(). *196*
Figura 8.8: Bucle infinito del nodo final. *197*
Figura 9.1: Arquitectura de la plataforma HW/SW propuesta: Elemento The Things Network. *199*
Figura 9.2: Portal web The Things Network. *200*
Figura 9.3: Consola de TTN. *201*
Figura 9.4: Gateway Overview. *201*
Figura 9.5: Application Overview. *202*
Figura 9.6: Application Overview_2. ..... *203*
Figura 9.7: Device Overview. *204*
Figura 9.8: Device Overview_2. ..... *204*
Figura 10.1: Adaptación de la plataforma HW/SW para la solución Smart Parking. *207*
Figura 10.2: Montaje completo del Nodo Final. *209*
Figura 10.3: Sensor HC-SR04. ..... *209*
Figura 10.4: Declaración de la clase 'Ultrasonic'. *211*
Figura 10.5: Función distance_in_cm(). *212*
Figura 10.6: Batería LiPo BAT573. ..... *213*
Figura 10.7: Diagrama de flujo del funcionamiento del nodo final. *216*
Figura 10.8: Selección de las tecnologías de comunicación y variables declaradas. *223*
Figura 10.9: Variables configurables. *223*

Figura 10.10: Timers, RTC y objeto sensor utilizados. 224
Figura 10.11: Función consulta_sensor(). 224
Figura 10.12: Funciones increase_frameCounter() y actualizar_config(). 225
Figura 10.13: Funciones define_mode(), clearframe(), clearbytearr() y estadoPlaza(). 226
Figura 10.14: Función formarTrama(). 227
Figura 10.15: Función formarTrama()_2. 228
Figura 10.16: Función trama(). 228
Figura 10.17: Codificación del funcionamiento del nodo final. 229
Figura 10.18: Codificación del funcionamiento del nodo final. 230
Figura 10.19: Selección de la tecnología de comunicación para el nodo estación base. 231
Figura 10.20: Adaptación del firmware del nodo estación base. 231
Figura 10.21: Función Decoder. 233
Figura 10.22: Función Decoder_2. 234
Figura 10.23: Función Encoder. 235
Figura 10.24: Sentencia JSON. 235
Figura 10.25: Montaje del nodo final y del nodo estación base. 236
Figura 10.26: Nodo final. 236
Figura 10.27: Nodo estación base. 237
Figura 10.28: Nodo gateway. 237
Figura 10.29: Datos recibidos en TTN sin obstáculo en el sensor. 239
Figura 10.30: Simulación de ocupación de la plaza de estacionamiento. 240
Figura 10.31: Recepción de nueva trama indicando ocupación de la plaza de estacionamiento. 240
Figura 10.32: Simulación de liberación de la plaza de estacionamiento. 241
Figura 10.33: Recepción de nueva trama indicando liberación de la plaza de estacionamiento. 241
Figura 10.34: Tramas enviadas por el nodo final original. 242
Figura 10.35: Tramas enviadas por el nodo final tras ser configurado. 242

Índice de Tablas

Tabla 2.1: Visibilidad en comunicaciones VLC para diferentes condiciones climáticas. 25
Tabla 3.1: Comparación de diferentes tecnologías de iluminación. 55
Tabla 3.2: Comparación de diferentes tecnologías de sensorización lumínica 56
Tabla 3.3: Modos de operación PHY I. 64
Tabla 3.4: Modos de operación PHY II. 64
Tabla 3.5: Modos de operación PHY III. 65
Tabla 3.6: Ventajas y desventajas de la tecnología VLC. 67
Tabla 3.7: Condiciones de experimento para el montaje 1. 71
Tabla 3.8: Resultados del experimento del montaje 1. 72
Tabla 3.9: Condiciones de experimento para el montaje 2. 74
Tabla 3.10: Resultados del experimento del montaje 2. 74
Tabla 3.11: Comparativa de prestaciones entre lámparas. 76
Tabla 3.12: Variación de la señal analógica en función de la frecuencia de la señal. 77
Tabla 3.13: Resultados con el receptor de salida digital para el montaje 2. 78
Tabla 3.14: Variación de la señal digital en función de la frecuencia de la señal. 80
Tabla 3.15: Condiciones de experimento para el montaje 3. 81
Tabla 3.16: Funciones asociadas al estado de los Jumper en la Expansion Board. 90
Tabla 3.17: Influencia del valor de R1 en la comunicación del montaje 4. 95
Tabla 3.18: Relación LED – consumo de potencia. 103
Tabla 3.19: Estudio comparativo entre tipos de LED y su consumo en diferentes configuraciones. 108
Tabla 10.1: Características del sensor HC-SR04. 210
Tabla 10.2: Contenido del Byte 0 de las tramas. 218
Tabla 10.3: Contenido de las tramas de tipo Start Frame. 219
Tabla 10.4: Contenido de las tramas de tipo Info Frame. 219
Tabla 10.5: Contenido de las tramas de tipo Keep-Alive Frame. 220
Tabla 10.6: Contenido de las tramas de tipo Configuration Frame. 221
Tabla 10.7: Contenido del Byte 0 de las tramas de tipo Configuration Frame. 222

Pliego de condiciones

Tabla PC. 1: Condiciones Hardware. 260
Tabla PC. 2: Condiciones Software. 261
Tabla PC. 3: Condiciones Firmware. 261

Tabla P. 1: Coeficientes reductores para trabajo tarifado según el COIT. *264*
Tabla P. 2: Amortización del material hardware. *266*
Tabla P. 3: Amortización del software. *267*
Tabla P. 4: Amortización del inmovilizado material. *267*
Tabla P. 5: Presupuesto, incluyendo trabajo tarifado y amortización del inmovilizado material. *268*
Tabla P. 6: Presupuesto, incluyendo trabajo tarifado, amortización y redacción del trabajo. *269*
Tabla P. 7: Coste de material fungible. *270*
Tabla P. 8: Presupuesto total del Trabajo Fin de Máster. *271*

Tabla de Acrónimos

ABP	*Activation By Personalization*
ACK	*Acknowledgement*
ADC	*Analogue to Digital Converter*
ADR	*Adaptative Data Rate*
AES	*Advanced Encryption Standard*
API	*Application Programming Interface*
ASCII	*American Standard Code for Information Interchange*
ASIC	*Application-Specific Integrated Circuit*
BIBO	*Be-In/Be-Out*
BLE	*Bluetooth Low Energy*
CO	*Monóxido de Carbono*
CO_2	*Dióxido de Carbono*
COIT	*Colegio Oficial de Ingenieros de Telecomunicación*
CPU	*Central Processing Unit*
CRC	*Cyclic Redundancy Check*
CSBM	*Color-Space-Based Modulation*
CSK	*Color Shift Keying*
CSS	*Chirp Spread Spectrum*
D-PPM	*Differencial Pulse Position Modulation*
DUT	*Device-Under-Test*
EITE	*Escuela de Ingeniería de Telecomunicación y Electrónica*
ETSI	*European Telecommunications Standards Institute*

FEC	*Forward Error Correction*
FHSS	*Frequency Hopping Spread Spectrum*
FPGA	*Field-Programmable Gate Array*
FSK	*Frequency Shift Keying*
GAP	*Generic Access Profile*
GATT	*Generic Attribute Profile*
GCM	*Generalized Color Modulation*
GND	*Ground*
GNSS	*Global Navigation Satellite System*
GPIO	*General-Purpose Input-Output*
GPS	*Global Positioning System*
HW	*Hardware*
I^2C	*Inter-Integrated Circuit*
I^2S	*Inter-IC Sound*
ID	*Identificador*
IDE	*Integrated Development Environment*
IEEE	*Institute of Electrical and Electronics Engineers*
IGIC	*Impuesto General Indirecto Canario*
IoT	*Internet of Things*
IP	*Internet Protocol*
I-PPM	*Inverse Pulse Position Modulation*
IR	*Infrared Radiation*
ISM	*Industrial, Scientific and Medical bands*
ISO	*International Organization for Standardization*
ITU	*International Telecommunication Union*

IVWSN *Intra-Vehicular Wireless Sensor Network*

JSON *JavaScript Object Notation*

LED *Light-Emitting Diode*

LiPo *Polímero de Litio*

LoRa *Long Range*

LoRaWAN *Long Range Wide Area Network*

LoS *Line of Sight*

LPWAN *Low Power Wide Area Network*

LTE *Long Term Evolution*

M2M *Machine-to-Machine*

MAC *Medium Access Control*

MCM *Multi Carrier Modulation*

MIMO *Multiple-Input Multiple-Output*

NB-IoT *Narrow Band-Internet of Things*

NFC *Near Field Communication*

NO_2 *Dióxido de Nitrógeno*

O_3 *Ozono*

OFCOM *Office of Communications*

OFDM *Orthogonal Frecuency-Division Modulation*

OOK *On-Off Keying*

OTAA *Over The Air Activation*

ONU *Organización de Naciones Unidas*

OSI *Open Systems Interconnection*

P2P *Peer to Peer*

PAN *Personal Area Network*

PC	*Personal Computer*
PDF	*Portable Document Format*
PHY	*Physical Layer*
PIB	*Producto Interior Bruto*
PM	*Particulate Matter*
PPM	*Pulse Position Modulation*
PWM	*Pulse-Width Modulation*
RAM	*Random Access Memory*
REPL	*Read-Eval-Print-Loop*
RF	*Radio Frequency*
RFID	*Radio Frequency Identification*
RGB	*Red-Green-Blue*
RSSI	*Received Signal Strength Indicator*
RTC	*Real Time Clock*
SCM	*Single Carrier Modulation*
SD	*Secure Digital*
SF	*Spreading Factor*
SISO	*Single-Input Single-Output*
SO_2	*Dióxido de Azufre*
SPI	*Serial Peripheral Interface*
SSID	*Service Set Identifier*
SSM	*Spread Spectrum Modulation*
SW	*Software*
TCP	*Transmission Control Protocol*
TDoA	*Time Difference of Arrival*

TFM	*Trabajo Fin de Máster*
TIC	*Tecnología de la Información y las Comunicaciones*
TTN	*The Things Network*
TV	*Television*
UART	*Universal Asynchronous Receiver/Transmitter*
UDP	*User Datagram Protocol*
ULPGC	*Universidad de Las Palmas de Gran Canaria*
USB	*Universal Serial Bus*
UUID	*Universally Unique IDentifier*
V2V	*Vehicle-to-Vehicle*
VLC	*Visible Light Communication*
V-PPM	*Variable Pulse Position Modulation*
WiFi	*Wireless Fidelity*
WLAN	*Wireless Local Area Network*
WNAM	*Wireless Neighborhood Area Network*
WPAN	*Wireless Personal Area Network*
WUP	*World Urbanization Prospects*

Memoria

Capítulo 1. Introducción

Varios años atrás, se comenzó a gestar lo que hoy se conoce como Internet de las Cosas o *Internet of Things* (IoT), cuando este concepto no era más que una idea futurista. En aquel entonces, aumentar exponencialmente el número de comunicaciones, debido a este nuevo concepto, haciendo uso del espectro radioeléctrico ya representaba un reto. El espectro radioeléctrico es un bien compartido y limitado, por lo que su uso posee muchas restricciones. Cada vez más se buscan soluciones que no afecten a la congestión del espectro radioeléctrico o que incluso ayuden a descongestionarlo, traspasando un buen número de comunicaciones inalámbricas de una banda concreta y saturada, a otra banda menos explotada.

Antes de adentrarse en estas nuevas soluciones o tecnologías de comunicación, se debe describir el concepto de IoT. IoT se basa fundamentalmente en la interconexión de cualquier producto con otro, haciendo uso de Internet, y creando de esta manera un entorno inteligente y versátil. La interconexión de productos, máquinas, y objetos, permite crear una red de dispositivos conectados que debe ser gestionada y controlada. Al mantener los objetos conectados, se puede conocer su estado, y en consecuencia interactuar con ellos.

El número de dispositivos conectados está aumentando considerablemente en los últimos años, y se prevé que siga con esa progresión. A esto hay que sumar que la mayor parte de los dispositivos electrónicos que salen al mercado disponen de alguna opción que permite conectarlos a Internet, favoreciendo aún más el desarrollo de soluciones IoT.

En definitiva, los dispositivos conectados a Internet no hacen más que incrementar su número, tal y como se indica en la Figura 1.1, en la que se adjunta una gráfica de Statista [1], portal de estadísticas *online*. En esta Figura 1.1 se muestra que cada vez hay más dispositivos conectados a Internet, por lo que el número de dispositivos que se pueden usar para aplicaciones IoT se incrementa también. En el año 2012 se alcanzó la cifra de los 8.700 millones de dispositivos conectados, y para el año 2020 se estima que se alcancen 50.100 millones. Cada año el número de dispositivos conectados se incrementa, y en las estimaciones a corto plazo el crecimiento se espera que sea aún mayor.

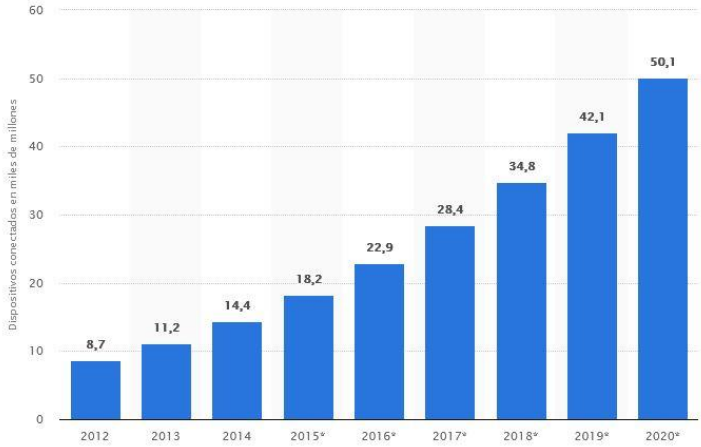


Figura 1.1: Número de dispositivos conectados a Internet.

El aumento de dispositivos conectados repercute en el uso de las bandas de frecuencia más habituales para la comunicación inalámbrica, por lo que estas se ven cada vez con más carga, y como se comentó anteriormente, ante la saturación del espectro de radiofrecuencia y la necesidad de liberar y descongestionar las bandas más recurridas, van apareciendo y consolidándose nuevas tecnologías de comunicación.

Como una buena solución a este problema de la saturación radioeléctrica surge *Visible Light Communication* (*VLC*). Una tecnología de comunicación óptica inalámbrica que hace uso del espectro visible (400 THz – 789 THz) para sus comunicaciones [2]. Esta tecnología de comunicación, al trabajar en el espectro visible, permite mantener comunicaciones inalámbricas sin la utilización de bandas tan recurridas como las bandas ISM (*Industrial, Scientific and Medical*) o las bandas empleadas en *WiFi* y *Bluetooth* (2.4 GHz y 5 GHz), tal y como se explica en [3]. Además, esta tecnología, debido a su naturaleza y a la reutilización de la infraestructura ya existente, permite obtener soluciones de altas prestaciones en

términos de tasas de datos, de consumo energético, y de convivencia con otras señales de radiofrecuencia, como es el caso de [4].

Por otra parte, ante el reto de la comunicación en dispositivos IoT, que mayoritariamente requieren un bajo consumo de potencia, ya que se suelen alimentar por una pila de botón o por una batería, surgen también tecnologías de comunicación como *Bluetooth Low Energy* (*BLE*) [5] o *LoRa/LoRaWAN* [6].

En función de la aplicación, se opta por una tecnología de comunicación o por otra. Por ejemplo, se puede seleccionar una tecnología de bajo consumo según el alcance que requiera la comunicación de dicha aplicación. En la referencia [7] se realiza un estudio comparativo de las principales tecnologías de comunicación que permite visualizar qué tecnología encaja con según qué aplicación. Si la comunicación es de área personal, se puede utilizar *BLE*, mientras que, si la aplicación demanda una comunicación de largo alcance, se puede utilizar *LoRa/LoRaWAN*. Para comunicaciones de área local se puede optar por tecnologías como *WiFi*, en la que el consumo energético, en este caso, no es tan bajo como en las tecnologías anteriormente citadas.

En este sentido, en la referencia [8] se realiza una primera aproximación hacia la utilización de tecnología *VLC* en la gestión de sensores. En este trabajo se realizó una pasarela entre *VLC* y *LoRa/LoRaWAN* basada en recursos de elevadas prestaciones en el ámbito de IoT que permite recibir información por luz visible (*VLC*) y transmitirla a un nodo *gateway* a través de una comunicación de largo alcance y bajo consumo de potencia como *LoRa/LoRaWAN*. Se trataba de una primera aproximación que perseguía el objetivo de estudiar la viabilidad de integrar la comunicación *VLC* y *LoRaWAN*. Con respecto a este trabajo, en el presente Trabajo Fin de Máster se pretende, además de mantener la funcionalidad de pasarela de la plataforma, integrar diferentes tecnologías como *BLE*, *VLC* o *LoRa/LoRaWAN* que permitan una comunicación bidireccional incluso de forma simultánea en un dispositivo de IoT. Es definitiva, se plantea una plataforma de carácter genérica que permita la comunicación, con las tecnologías integradas (*BLE*, *VLC* y *LoRa/LoRaWAN*), en cualquier sentido, de tal forma que se pueda aplicar a cualquier caso de uso dentro del ámbito de IoT.

1.1 Objetivos

El objetivo de este TFM consiste en el desarrollo de una plataforma *Hardware/Software* (HW/SW) genérica basada en dispositivos de IoT, que permita comunicarse bidireccionalmente con las tecnologías de comunicación *BLE*, *VLC* y *LoRa/LoRaWAN*. Se pretende realizar una plataforma que integre la funcionalidad completa de cada módulo de comunicación con el objetivo de que sea aplicable a cualquier caso de uso dentro del ámbito IoT. Se plantea que sea una plataforma *open source* con el fin de seleccionar en cada caso de uso, las comunicaciones y el sentido de comunicación, que requiera cada solución IoT.

Como caso de uso de la plataforma propuesta, se plantea en este TFM el ámbito de las *Smart Cities,* y en concreto el campo de *Smart Parking* como solución IoT que puede beneficiarse de una plataforma con las características descritas. El objetivo social que se persigue en el presente Trabajo Fin de Máster al plantear este caso de uso es el de agilizar el tráfico, garantizar la sostenibilidad y el bienestar medioambiental, así como economizar el tiempo de los conductores en la difícil tarea de la búsqueda de estacionamiento.

Dentro de los objetivos de este TFM se incluye el estudio y aplicación de varias tecnologías de comunicación, entre las que se encuentran *VLC*, *BLE*, *LoRa/LoRaWAN* y *WiFi*. Se estudian diversas tecnologías de telecomunicación con el fin de comparar sus propiedades en diferentes escenarios y valorar qué tecnologías de comunicación encajan mejor con los requerimientos de comunicación de la plataforma HW/SW propuesta. Este estudio comprende la caracterización de los casos de aplicación y la valoración cuantitativa del desempeño mostrado por cada una de las tecnologías, de tal forma que se pueda realizar una comparación real entre estas.

Para cumplir con las exigencias que plantea una solución de carácter IoT, se utilizará un dispositivo que ha sido ideado, diseñado y comercializado para soluciones de este tipo. Este es el dispositivo LoPy del fabricante Pycom [9]. La caracterización de este dispositivo es un objetivo del presente TFM.

La arquitectura que sigue la solución propuesta consta de tres tipos de nodos. El primer tipo de nodo es aquel que se considera como estación móvil, el segundo tipo de nodo se

considera estación base, y el tercer tipo de nodo se considera un nodo *gateway*. Se tienen conexiones individuales entre el nodo móvil y su nodo estación base asignado, y a su vez, cada estación base se comunica con el nodo *gateway*. Para conformar esta configuración, se considera como objetivo de este TFM la programación de los nodos móviles, de los nodos estación base y del nodo *gateway*. Además, en relación con la configuración y programación de los nodos *gateway*, se tendrá una aplicación que interactúe con la información presente en dichos nodos, integrada inicialmente en *The Things Network* (TTN).

Otro objetivo adicional es la caracterización de los sensores que proveen al servicio de información. En función del caso de uso, se debe utilizar un sensor u otro, por lo que para la plataforma genérica inicial que se plantea no se especifica un sensor determinado. Los sensores deben contar con una interfaz compatible con el dispositivo de comunicación.

A continuación, se resumen los objetivos de forma estructurada:

- Estudio, aplicación y comparación de las tecnologías de comunicación *VLC*, *BLE*, *LoRa/LoRaWAN* y *WiFi*.
- Caracterización del sensor o sensores.
- Caracterización de los dispositivos LoPy.
- Programación de la funcionalidad de los nodos móviles.
- Programación de la funcionalidad de los nodos estación base.
- Programación de la funcionalidad del nodo *gateway*.
- Integración de la aplicación en TTN.

1.2 Peticionario

Actúa como peticionario del presente Trabajo Fin de Máster (TFM) la Escuela de Ingeniería de Telecomunicación y Electrónica (EITE) de la Universidad de Las Palmas de Gran Canaria (ULPGC) como requisito indispensable para la obtención del título de Ingeniero de Telecomunicación, tras haber superado con éxito las asignaturas especificadas en el Plan de Estudios.

1.3 Estructura del documento

El presente documento está dividido en cuatro partes diferenciadas: Memoria, Pliego de condiciones, Presupuesto y Anexo. A su vez, la Memoria se ha estructurado en once capítulos, además de las referencias bibliográficas, tal como se describe a continuación. La memoria está estructurada de la siguiente forma:

- **Capítulo 1.** ***Introducción.*** En este capítulo se presentan los motivos que han dado lugar al planteamiento de este Trabajo Fin de Máster (TFM), se explican los conceptos básicos que se tratarán en este y se presentan, además, los objetivos, el peticionario y la estructura del documento.

- **Capítulo 2.** ***Antecedentes.*** En este capítulo se realiza un estudio completo sobre la temática del presente TFM. Se comienza presentando el concepto de IoT y estudiando las tecnologías de comunicación *VLC*, *BLE* y *LoRa/LoRaWAN*, para posteriormente estudiar y comentar soluciones de tipo *Smart Cities,* y en concreto de *Smart Parking*. Finalmente se expone la arquitectura de la plataforma HW/SW propuesta.

- **Capítulo 3. VLC.** En este capítulo se expone el estudio realizado sobre la tecnología de comunicación *VLC*. También se realiza el desarrollo *hardware* y *firmware* necesario para implementar esta tecnología en la plataforma HW/SW propuesta. En este mismo capítulo se caracteriza y documenta el dispositivo LoPy.

- **Capítulo 4. BLE.** En este capítulo se expone el estudio realizado sobre la tecnología de comunicación *BLE*. También se realiza el desarrollo *firmware* necesario para implementar esta tecnología en la plataforma HW/SW propuesta.

- **Capítulo 5. *LoRa/LoRaWAN.*** En este capítulo se expone el estudio realizado sobre la tecnología de comunicación *LoRa/LoRaWAN*. También se realiza el desarrollo *firmware* necesario para implementar esta tecnología en la plataforma HW/SW propuesta.

- **Capítulo 6. *Nanogateway*.** En este capítulo se presenta el desarrollo del elemento nodo *Nanogateway* de la arquitectura propuesta. Se analiza en profundidad el tipo de *gateway* usado, incluyendo su caracterización y estudio del *firmware* utilizado.

- **Capítulo 7. *Estación base.*** En este capítulo se presenta el desarrollo del elemento *Nodo Estación Base* de la arquitectura propuesta. Se analiza en profundidad el diseño y el desarrollo HW/SW de este elemento.

- **Capítulo 8. *Nodo final.*** En este capítulo se presenta el desarrollo del bloque *Nodo final* de la arquitectura propuesta. Se analiza en profundidad el diseño y el desarrollo HW/SW de este elemento.

- **Capítulo 9. *The Things Network.*** En este capítulo se presenta la configuración y el desarrollo realizado en la plataforma *The Things Network*.

- **Capítulo 10. *Use of Case: Smart Parking.*** En este capítulo se desarrolla un caso de uso de la plataforma HW/SW genérica presentada en los capítulos anteriores. En concreto se adapta la plataforma genérica para una solución *Smart Parking*.

- **Capítulo 11. *Conclusiones y líneas futuras.*** En este capítulo se resumen las conclusiones y líneas futuras que se valoran a partir de la experiencia obtenida a partir de la realización del TFM.

La segunda parte del documento consiste en el Pliego de Condiciones, mientras que la tercera parte se corresponde con el Presupuesto, y por último un Anexo que contiene información adicional sobre la documentación aportada.

Capítulo 2. Antecedentes

2.1 *Internet of Things*

El crecimiento del número de dispositivos que cuentan con tecnologías de comunicación integradas, así como el abaratamiento de los costes del *hardware* asociado, son dos de los pilares responsables del auge que vive el concepto de *Internet of Things* (IoT). La dinámica que sigue el mercado de los transceptores inteligentes integrados es de disminución de costes y reducción del consumo de potencia. A todo esto, se deben sumar las mejoras en sistemas de almacenamiento de energía y de datos. En definitiva, el desarrollo continuo de la tecnología permite que se vayan implementando, cada vez más, soluciones basadas en IoT.

2.1.1 Concepto

El concepto de IoT se fundamenta en la interconexión de cualquier producto con otro, haciendo uso de Internet, y creando de esta manera un entorno inteligente y versátil. La interconexión de productos, máquinas, y objetos, permite crear una red de dispositivos conectados que debe ser gestionada y controlada. Al mantener los objetos conectados, se puede conocer su estado, y en consecuencia interactuar con ellos.[10]

Para hablar de IoT, se debe estudiar el nivel de inteligencia. El nivel de inteligencia de los objetos es el grado de conocimiento y control sobre el dispositivo conectado. Así, existen diferentes niveles, los más bajos, implican un menor grado de interactividad con el dispositivo y los más altos, permiten un mayor número de funcionalidades al tener mayor control y conocimiento del dispositivo conectado.

- Nivel 1: Identificación. Nivel mínimo e imprescindible. Se puede identificar el objeto de forma única.

- Nivel 2: Ubicación. Se conoce la ubicación del dispositivo, tanto actual como pasada.

- Nivel 3: Estado. Se conoce la información, el estado y las características del dispositivo. Asociado al uso de sensores.

- Nivel 4: Contexto. El dispositivo es capaz de conocer su entorno y transmitir información de este.

- Nivel 5: Criterio propio. El dispositivo es capaz de ejecutar acciones según la información, la ubicación, el estado, el entorno, y otras circunstancias.

En función del nivel de inteligencia del dispositivo IoT se utilizan unas tecnologías u otras. Como referencia, para el nivel de identificación resulta muy útil hacer uso de tecnologías como *Radio Frecuency Identification* (RFID) o *Near Field Communications* (NFC), mientras que para el nivel de ubicación se suele utilizar la tecnología *Global Positioning System* (GPS), con el propósito de medir el estado y en contexto se incluyen sensores, y para el nivel de criterio propio del dispositivo se pueden añadir directamente actuadores. Además, un dispositivo IoT puede integrar varias de las tecnologías propias de cada nivel.

De cada nivel de inteligencia se pueden extraer capacidades que diferencian a unos dispositivos de otros. Estas capacidades son:

- Comunicación y cooperación: Los objetos se conectan a los servicios de Internet, y también entre ellos, favoreciendo la cooperación.

- Capacidad de direccionamiento: Localizable en la red IoT.

- Identificación: Se logra identificar al objeto mediante diferentes tecnologías.

- Localización: Asociado al nivel de inteligencia de ubicación.

- Actuación: El objeto posee actuadores y sabe cómo gestionar su uso.

2.1.2 Impacto y crecimiento de IoT

Con el objetivo de conocer el impacto de IoT en el ámbito de la conectividad, en la Figura 2.1 se muestra una gráfica extraída de *IoT Analytics* [11], *e*n la que se aprecia el número de conexiones activas entre dispositivos en miles de millones, a nivel mundial. En la Figura 2.1 se pueden visualizar dos cifras, en rojo la cifra de conexiones activas generadas por dispositivos IoT, y en blanco la del resto de dispositivos. Se entiende por resto de dispositivos a los *smartphones*, ordenadores de sobremesa y portátiles, y demás conexiones que no se consideren de carácter IoT. Los datos, desde el año 2015 hasta el año 2018 son reales, mientras que los datos de 2019 a 2025 son estimaciones según el crecimiento esperado en el mercado de la conectividad. Actualmente, el número de conexiones activas generadas por IoT es de 7 mil millones, frente a los 10,8 mil millones de conexiones activas generadas por otros dispositivos.

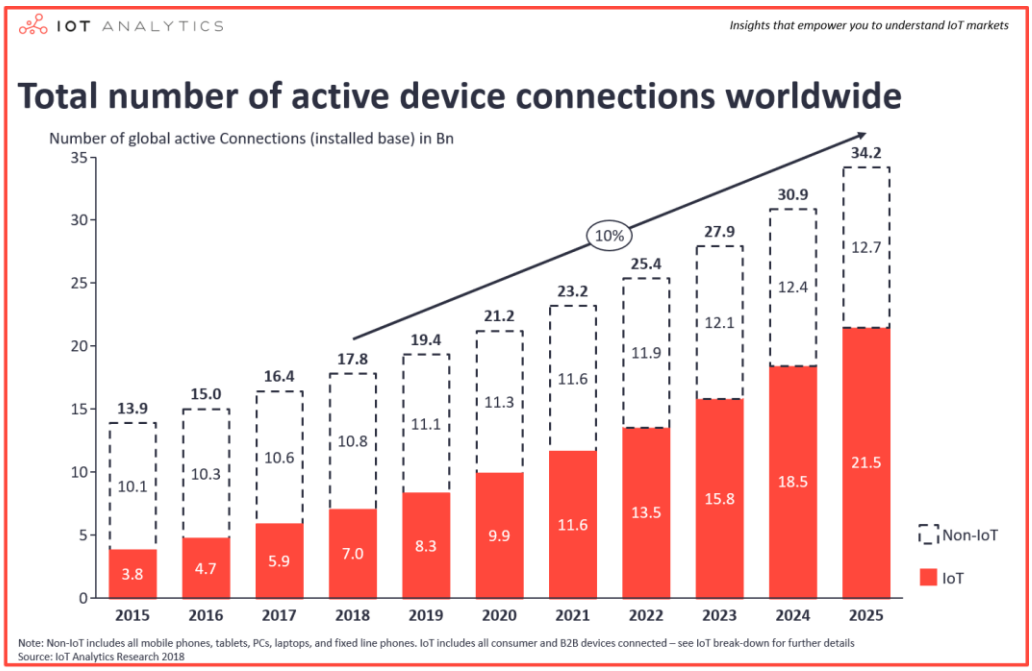


Figura 2.1: Número de conexiones activas a nivel mundial desde 2015 a 2025.

Se puede ver cómo el número de conexiones va a ir en aumento, tanto en los dispositivos IoT como en el resto de los dispositivos. Pero se aprecia la diferencia en este crecimiento, siendo el crecimiento de dispositivos IoT más notorio. Tanto es así, que para el año 2021 se espera que el número de conexiones generadas por IoT sea igual que el de conexiones generadas por otros dispositivos. Para el año 2025, se espera una cifra de 21.5 mil millones de conexiones para dispositivos IoT, mientras que para los demás dispositivo la cifra se mantendría sobre los 12 mil millones de conexiones. La cifra total de conexiones rondaría los 35 mil millones de conexiones.

Por otra parte, cuando se habla de IoT, se debe tener en consideración cuál es su arquitectura genérica, esta se representa en la Figura 2.2 [12]. Como si de un modelo OSI se tratara, se puede analizar la arquitectura IoT desde la capa física hasta la capa de aplicación, pasando por capas de red. En la capa más baja de la arquitectura IoT, se encuentran los dispositivos que se encargan de la valoración del entorno y de transmitir esa información a capas superiores. En esta capa se encuentra el nivel de sensorización, en la que dependiendo de la solución que se desee implementar se encuentran diferentes tipos de sensores.

En esta capa inferior de la arquitectura se encuentran los denominados nodos finales de las soluciones. En la capa inmediatamente superior a esta se encuentra el nivel dónde se ubican los nodos concentradores o nodos *gateways*. Estos nodos se suelen encargar de recibir la información desde los dispositivos finales y comunicarse o bien con las capas superiores, o bien con otros nodos *gateways* del mismo nivel.

En caso de que se comuniquen con capas superiores, entraría en escena la capa de red, en la que se realizan los procesos de encaminamiento y direccionamiento necesarios para transferir los datos al receptor deseado. Tras la capa de red, se encuentra la capa de almacenamiento o plataforma de recepción de datos. En este nivel se están desarrollando herramientas continuamente para mejorar el servicio que prestan a la aplicación, que sería la última capa de esta arquitectura.

Por supuesto, como en cualquier infraestructura de telecomunicación, existen dos factores que son trasversales a todas las capas, ya que se aplican, o deberían aplicarse en todos los

niveles de la arquitectura. Estos dos factores son la gestión y mantenimiento, y la seguridad.

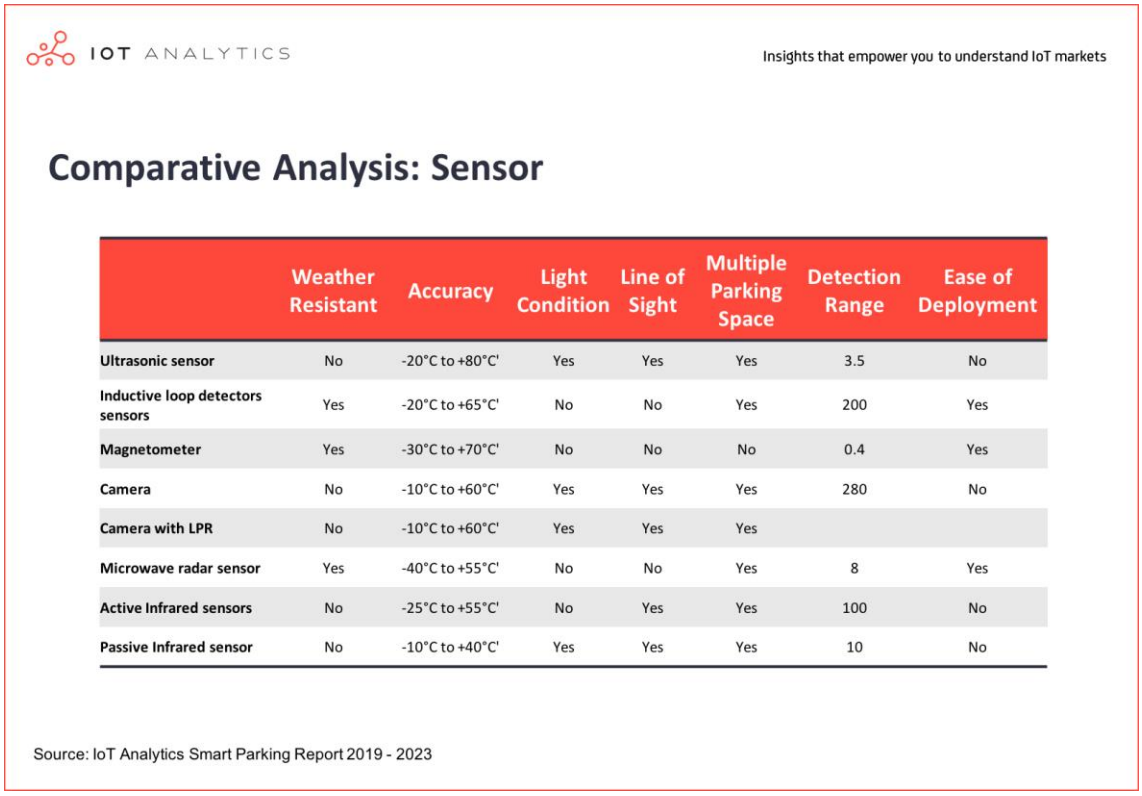
Figura 2.2: Arquitectura genérica de IoT.

Según se va recorriendo la arquitectura de forma ascendente, se va encontrando más memoria disponible, más capacidad de cómputo, más accesibilidad remota a esta, y, en definitiva, más centralización. Mientras que, si se va recorriendo la arquitectura hacia las capas más inferiores, se encuentra una topología más distribuida, con capacidad de medir el entorno y con más ahorro de energía.

En cuanto a la tecnología utilizada en IoT, para el nivel más bajo de la arquitectura se debe hablar de los elementos que miden parámetros del entorno, los sensores. Existen multitud de sensores que recogen información del medio y la transmiten a los dispositivos o nodos finales, habitualmente de forma cableada. En función de la tecnología en la que se basan, la resistencia a elementos externos, la zona de temperatura en la que pueden trabajar, el rango de actuación e incluso su dificultad de implementación, se escoge un tipo de sensor para una determinada aplicación u otro. Un ejemplo de esto se puede visualizar en la tabla de la Figura 2.3 de *IoT Analytics* [13]. En este ejemplo concreto sobre *Smart Parking* se valoran diferentes alternativas para detectar el estado de una plaza de aparcamiento.

Se realiza una clasificación sobre las características de cada sensor que pueden influir en el correcto desempeño de la solución concreta que se aborda. Como se puede ver en la

Figura 2.3, se utilizan sensores de ultrasonido, magnetómetros, radares, infrarrojos e incluso cámaras.



IOT ANALYTICS

Insights that empower you to understand IoT markets

Comparative Analysis: Sensor

	Weather Resistant	Accuracy	Light Condition	Line of Sight	Multiple Parking Space	Detection Range	Ease of Deployment
Ultrasonic sensor	No	-20°C to +80°C'	Yes	Yes	Yes	3.5	No
Inductive loop detectors sensors	Yes	-20°C to +65°C'	No	No	Yes	200	Yes
Magnetometer	Yes	-30°C to +70°C'	No	No	No	0.4	Yes
Camera	No	-10°C to +60°C'	Yes	Yes	Yes	280	No
Camera with LPR	No	-10°C to +60°C'	Yes	Yes	Yes		
Microwave radar sensor	Yes	-40°C to +55°C'	No	No	Yes	8	Yes
Active Infrared sensors	No	-25°C to +55°C'	No	Yes	Yes	100	No
Passive Infrared sensor	No	-10°C to +40°C'	Yes	Yes	Yes	10	No

Source: IoT Analytics Smart Parking Report 2019 - 2023

Figura 2.3: Análisis de diferentes sensores destinados a Smart Parking.

Los sensores se comunican directamente con los nodos finales, que suelen implementarse en plataformas *hardware* que permiten mantener dicha comunicación vía interfaces I^2C, SPI o comunicación serie en la mayoría de los casos. Estos mismos dispositivos permiten programar la funcionalidad necesaria para realizar una correcta transmisión de los datos recibidos desde el sensor hacia el nodo concentrador o *gateway*. La comunicación entre el nodo final y el nodo concentrador puede llevarse a cabo con diferentes tecnologías de comunicación que se diferencian en el alcance, la tasa de datos, el consumo de potencia y otros parámetros que son fundamentales a la hora de escoger una de ellas para cada solución. En la Figura 2.4 extraída de *IoT Analytics* [14] se muestra el nicho del número total de conexiones IoT que ocupa cada tipo de tecnología, sin entrar a valorar estándares o tecnologías concretas.

Las tecnologías de comunicación predominantes en la actualidad son las inalámbricas de área personal (WPAN) y las de área local (WLAN). Las comunicaciones cableadas y *Machine-to-Machine*(M2M), aunque en menor medida, también ocupan una parte dentro del

número total de conexiones IoT. Sin embargo, en la búsqueda de comunicaciones de largo alcance, pero con bajo consumo de potencia, surgen las tecnologías *Low Power Wide Area Network* (LPWAN), que para el año 2025 se espera que ocupen un rango significativo en el número de conexiones IoT, sólo por detrás de WPAN y WLAN. Para 2025, también se espera que la tecnología que actualmente se está empezando a implantar, 5G, cubra un nicho de comunicaciones IoT considerable. Para ese año, ya existirán también un buen número de aplicaciones basadas en un nuevo de tipo de red, denominadas *Wireless Neighborhood Area Network* (WNAM) que comunican dispositivos en un rango que cubra aproximadamente un barrio.

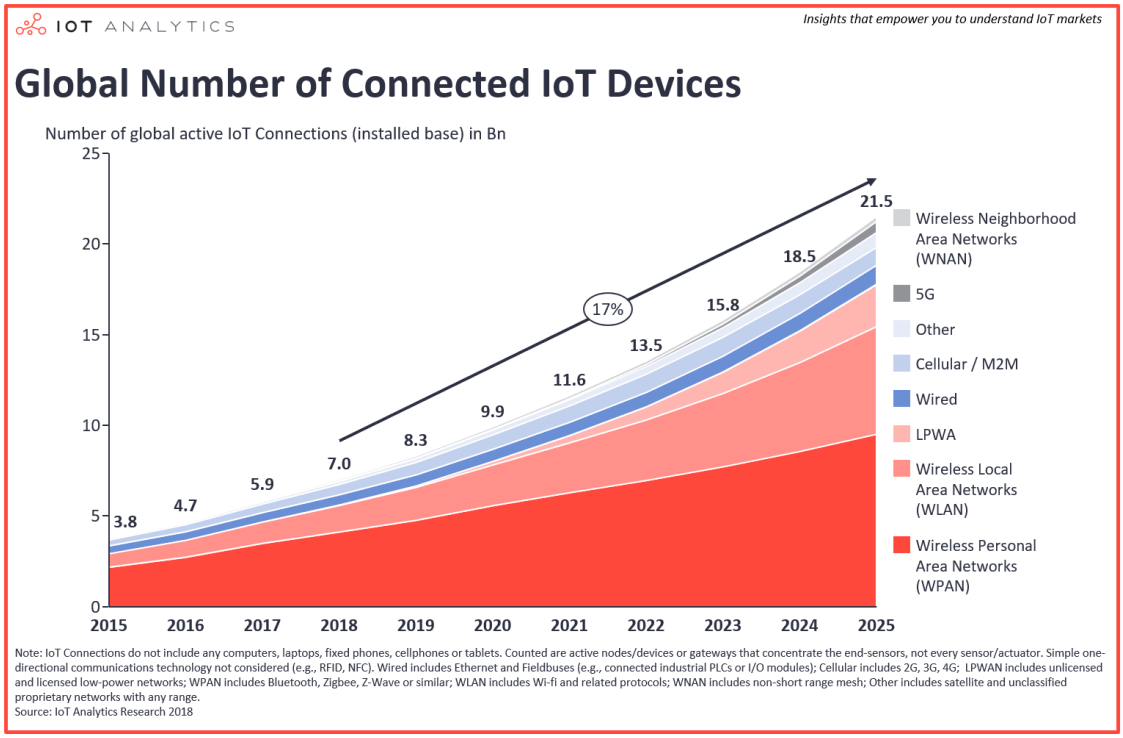


Figura 2.4: Desglose de tecnologías utilizadas en las conexiones IoT.

Cuando se trata de tecnología para las capas más elevadas de la arquitectura IoT, se habla de *software*. Desde la plataforma, base de datos o nube en la que se recogen y almacenan los datos, hasta la aplicación de usuario, se puede trabajar con multitud de tecnologías *software*. Existen plataformas específicas para aplicaciones IoT que permiten realizar una gestión eficiente de los datos recibidos desde las capas inferiores. Se siguen desarrollando tecnologías que faciliten la interpretación de los datos y que de forma autónoma sean capaces de presentar de manera inteligente los datos que requiere la aplicación, y por ende el usuario final.

Si se realiza un resumen de todas las tecnologías que tienen cabida en IoT, se puede obtener una imagen como la que se muestra en la Figura 2.5 [14]. Se trata de un radar, en el que las tecnologías más cercanas al centro son las que están más consolidadas en IoT, mientras que las más alejadas son aquellas que se están empezando a gestar. El radar descrito por *IoT Analytics* se divide en 3 fragmentos: IoT *hardware*, IoT *software* y IoT *connectivity*. Se consideran los tres bloques tecnológicos que sustentan el concepto de IoT, y en todos ellos se encuentran tecnologías maduras y tecnologías en fase de investigación.

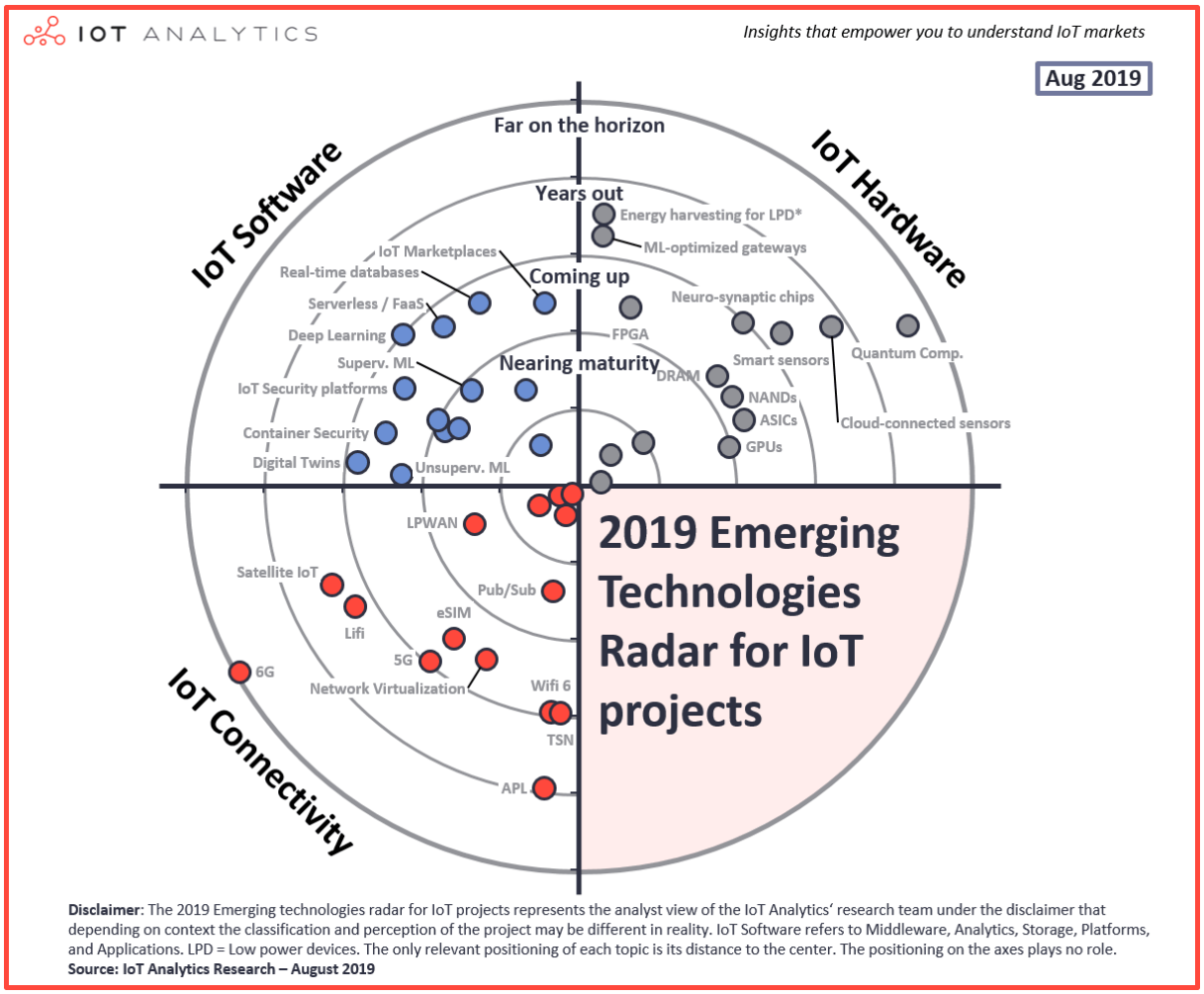


Figura 2.5: Radar de tecnologías empleadas en IoT.

Por ejemplo, dentro del bloque *hardware*, las *Central Processing Unit* (CPU) o los módulos de comunicación de bajo consumo de potencia son tecnologías ya consolidadas en IoT. Por su parte, las tecnologías *Field-Programmable Gate Array* (FPGA) o *Application-Specific Integrated Circuit* (ASIC) se van aproximando a esa fase de madurez, mientras que los

sensores inteligentes o la computación cuántica se encuentran todavía lejos de participar en soluciones comerciales.

Por otra parte, en el bloque *software*, tecnologías como la computación en la nube o las plataformas específicas para IoT son muy comunes en las soluciones IoT, mientras que *Machine Learning* o bases de datos en tiempo real son ejemplos de tecnologías que se van aproximando cada vez más a convertirse en una realidad dentro de IoT.

En cuanto al bloque de *connectivity*, como se explicó anteriormente, las redes inalámbricas de área personal y local son las más asentadas en el mercado, mientras que las redes LPWAN o WNAM están cada vez más cerca de la fase de madurez. Como tecnología de comunicación para el futuro se espera, todavía de forma lejana, 6G.

2.2 Estado del arte de las tecnologías *BLE*, *VLC* y *LoRa/LoRaWAN*

En este TFM se estudian diferentes tecnologías de comunicación como *BLE*, *VLC* y *LoRa/LoRaWAN*. Se trabaja con tecnologías de distintas con el fin de complementar sus propiedades en diferentes escenarios, valorando qué tecnologías de comunicación encajan mejor con los requerimientos de comunicación de la plataforma HW/SW propuesta. Antes de pasar al estudio técnico y detallado de cada tecnología, se aborda en este mismo capítulo el estado del arte de dichas tecnologías. *BLE*, *VLC* y *LoRa/LoRaWAN* han tenido diferentes impactos en el mercado, propiciadas a partir de las características de cada tecnología, como la fase de madurez, el coste, la seguridad o el alcance.

Se presentan a continuación diferentes aplicaciones y soluciones desarrolladas que se basan en estas tecnologías.

2.2.1 *Bluetooth Low Energy (BLE)*

Las aplicaciones o soluciones que implementan *BLE* como tecnología de comunicación son muy variadas, haciendo uso todas ellas de esta tecnología en comunicaciones de corto alcance. Entre otras aplicaciones existentes en la bibliografía, se encuentran aplicaciones,

como la expuesta en [15], que hace uso de *BLE* para el desarrollo de un sistema de gestión de asistencia de estudiantes.

El sistema consiste en la comunicación *BLE* entre el *smartphone* de cada estudiante y su tarjeta identificadora. Una de las ventajas que aporta esta aplicación es la velocidad a la hora de registrar la asistencia, al contar cada estudiante con un dispositivo propio y no depender de un dispositivo que debe pasar por toda la clase. Además, el hecho de no depender de una red inalámbrica de área local como una conexión *WiFi* y su posible estado, dota de independencia a este sistema de registro de asistencia. Esta aplicación aprovecha la propia limitación de la tecnología (ser de área personal) para evitar que los estudiantes puedan registrarse desde fuera del aula. En la Figura 2.6 [15] se muestra el diagrama que recoge el funcionamiento del sistema. Inicialmente, el profesor fija el ID y el nombre de la clase y activa la comunicación *BLE*. Cada alumno se registra desde su dispositivo usando *bluetooth* y el servidor recoge la asistencia.

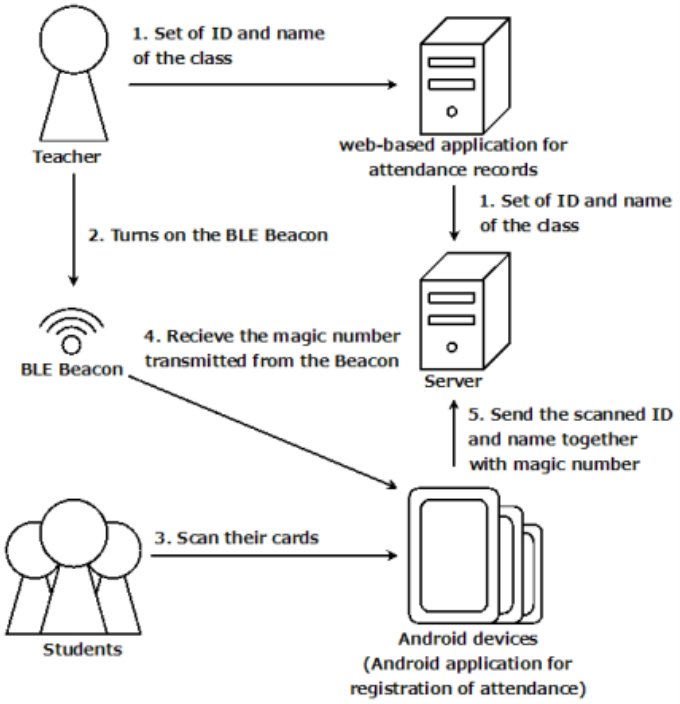


Figura 2.6: Diagrama de funcionamiento del sistema de control de asistencia por BLE.

También se hace uso de *BLE* en entornos médicos como es el caso del hospital *Nickaus Children's Hospital* [16]. En este centro sanitario se hace uso de *BLE* para la gestión de recursos médicos. Existía la necesidad de reducir el tiempo que emplean los enfermeros en la gestión de material como las carros de parada que incluyen toda la medicación del

paciente entre otras cosas, y con este fin se desarrolló un sistema basado en *BLE* para asociar cada material a los pacientes, y ubicar además las sillas de ruedas o bombas clínicas.

También se han desarrollado ideas basadas en *BLE* que aportan seguridad a los ciudadanos, especialmente en casos de violencia callejera, dónde la persona que comienza su ruta va pasando por una serie de *checkpoints* en los que se realiza una transacción *BLE* y si en algún punto se la ruta no se llega al *checkpoint* en un período de tiempo determinado, se envía un aviso a los números de contacto prestablecidos [17].

Con la misma filosofía de seguimiento que se tiene para garantizar la seguridad de las personas, también se puede hacer uso de la tecnología *BLE* para localizar o guiar a los turistas o clientes de una actividad. Por ejemplo, dentro de un museo se puede realizar una guía a partir de una ruta de puntos *BLE*.

También en el ámbito de la seguridad en carretera existen soluciones que integran *BLE*, como en el caso de [18], en el que haciendo uso de redes de sensores y de comunicación *bluetooth Vehicle-to-Vehicle (*V2V) se pretenden evitar peligros en el tráfico de las ciudades.

En esta línea, el artículo [19] presenta la idea de utilizar las *Intra-Vehicular Wireless Sensor Network* (IVWSN) de los vehículos modernos para, a través de *BLE*, mejorar y crear aplicaciones con mejores prestaciones en términos de consumo de potencia, complejidad, bajo coste y alta fiabilidad. En la Figura 2.7 [19] se muestra el porcentaje de comunicaciones correctas y fallidas en diferentes escenarios. Se puede comprobar que la comunicación intra-vehicular entre sensores inalámbricos a través de *BLE* es posible. Se aprecia que las mayores pérdidas de paquetes se encuentran en las comunicaciones de distintos compartimentos. Pero en general, la comunicación vía *BLE* se puede llevar a cabo, obteniendo un sistema de bajo consumo de potencia y fiable.

Las mejoras en la conducción vienen en diferentes soluciones. Además de las aplicaciones anteriores, *BLE* se puede usar también para la comunicación entre vehículos y señales de tráfico. Las señales de tráfico envían comunicación en formato *broadcast* y los vehículos la reciben y nutren con la información al usuario a través de la propia interfaz táctil, o bien de forma sonora. En [20] se realiza un estudio sobre el algoritmo de triangulación de las posiciones de las señales de tráfico a partir de cuatro radios *BLE* ubicadas en cuatro zonas distintas del coche, tal y como se muestra en la Figura 2.8 [20].

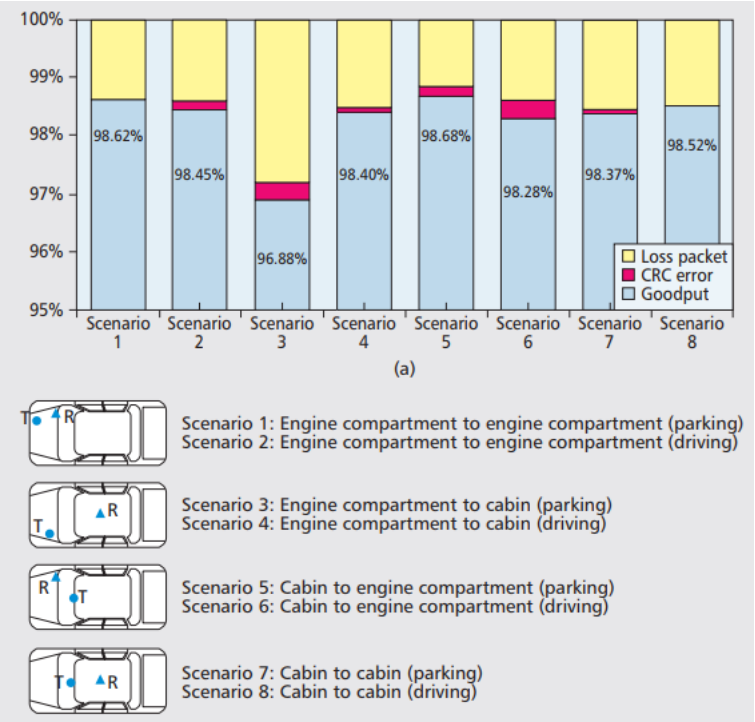


Figura 2.7: Porcentaje de comunicaciones BLE con acierto y error en un vehículo.

En este mismo artículo se realiza una relación interesante entre el parámetro *Received Signal Strength Indicator* (RSSI) y la distancia. Según [20] al transmitir la señal de tráfico en modo broadcast y ser recibida por cada radio *BLE*, si se recoge el indicador RSSI y se introduce en la Ecuación 2.1, se logra despejar la distancia entre el transmisor *broadcast* y cada uno de los receptores.

$$RSSI(d) = P0 - 20\log_{10}\frac{4\pi d}{\lambda}\ dBm \qquad (2.1),$$

donde P0 es una constante empírica.

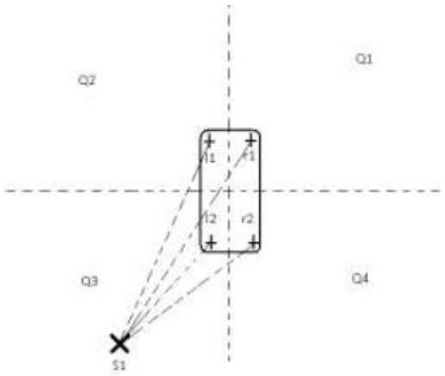


Figura 2.8: Triangulación de un vehículo a partir de cuatro radios BLE.

En la referencia [21] se estudia otra aplicación de *BLE* en el campo de *Smart Grid* y *Smart Home*. En este caso concreto se estudia además la interoperabilidad entre diferentes tecnologías inalámbricas dentro de un sistema *Smart Grid* y *Smart Home*. Las tecnologías con las que coexiste *BLE* son *ZigBee*, KNX-RF Multi y EnOcean. En la Figura 2.9 [21] se muestra un resumen de la infraestructura tratada y de los dispositivos conectados por cada protocolo.

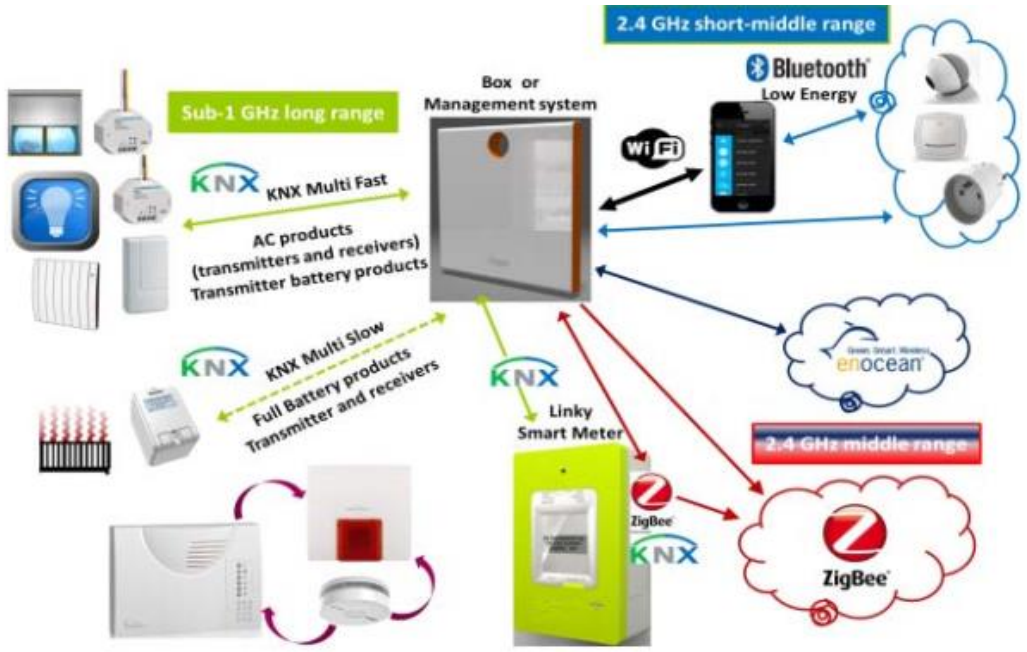


Figura 2.9: Coexistencia de ZigBee, KNX-RF, EnOcean y BLE en aplicación Smart Home.

Finalmente, se concluye que la coincidencia entre las tecnologías de telecomunicación utilizadas en esta aplicación no creaba colisiones de RF, dotando al sistema de una robustez importante en términos de interferencia.

Haciendo uso de *BLE* se desarrollan soluciones cada vez más autónomas, como es el caso del estudio realizado en [22] . En este desarrollo se estudia un sistema *Be-In/Be-Out* (BIBO) en el que mediante transacciones *BLE* se realiza un sistema de *ticketing* automático al estar dentro del vehículo. Existen multitud de sistemas de este tipo basados *en Radio Frequency Identification* (RFID), pero este artículo pretende mostrar *BLE* como una alternativa factible. En las conclusiones del artículo los autores aclaran que, para las pruebas realizadas, sí es posible utilizar *BLE* como alternativa, aunque faltaría completar las pruebas con más investigación y desarrollo.

2.2.2 *Visible Light Communication (VLC)*

En lo que respecta a la tecnología *VLC*, una de las aplicaciones más comunes en la bibliografía es la reutilización de una infraestructura de iluminación para la transmisión de datos. Esta idea se recoge entre otros artículos, en [23], en el que se explica el funcionamiento de un sistema de conectividad IoT con iluminación de interiores basándose en la tecnología *VLC*. Cada luminaria posee su propia dirección IP con el fin de poder controlar su estado de forma remota. En el plan propuesto también se incluyen sensores que permitan mejorar la eficiencia del sistema mientras sirve como una fuente de datos para nuevos sistemas y servicios en edificios.

El sistema se ha ideado para el entorno de una oficina, en el que los equipos obtienen conectividad vía *VLC* y el sistema de iluminación es inteligente. La inteligencia del sistema de iluminación se basa en el uso de sensores que captan el nivel de luminosidad, y en la comunicación que mantienen con las fuentes de luz artificial. En la Figura 2.10 [23] se muestra el diseño de la infraestructura propuesta.

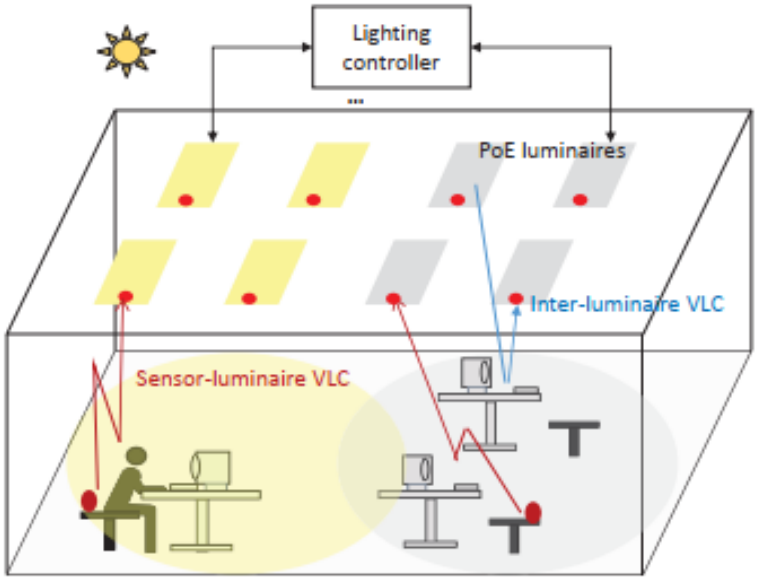


Figura 2.10: Sistema de iluminación y conectividad en entorno de oficina.

Tanto en entornos de trabajo como en los hogares, *VLC* permite reutilizar la infraestructura de iluminación para la transmisión de datos. En el artículo [24] se realiza un estudio sobre la relación señal-a-ruido y la tasa de error de bit en las comunicaciones *VLC* para diferentes posiciones de dispositivos dentro de una casa inteligente. En la Figura 2.11 [24] se muestra

la posición escogida para tres dispositivos que van a ser sometidos a una simulación de comunicación *VLC*.

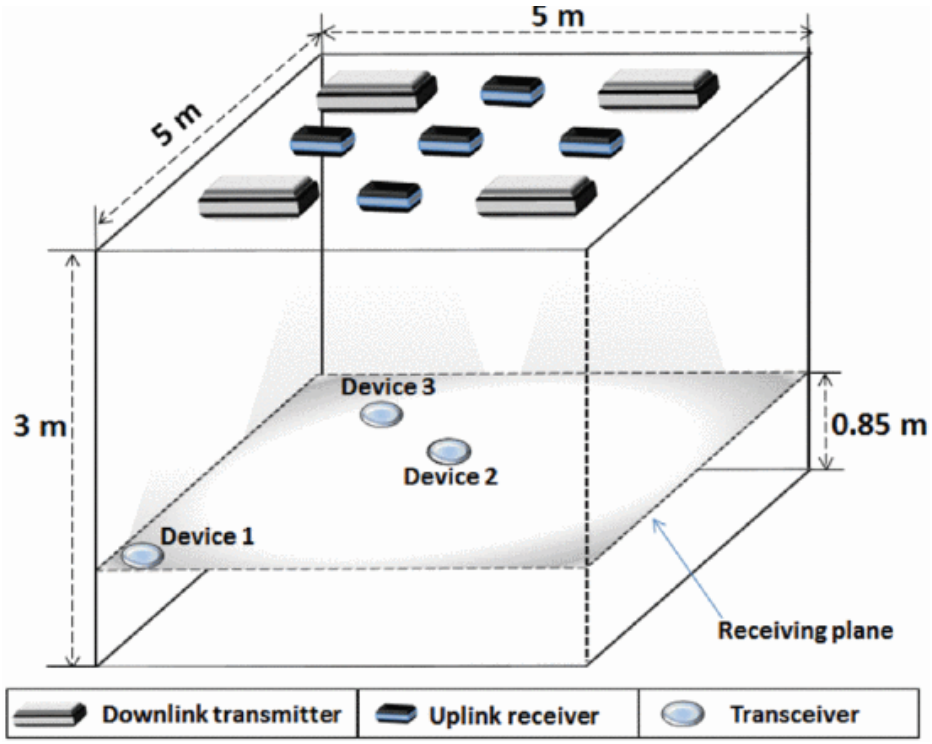


Figura 2.11: Estudio de comunicación VLC en un habitáculo.

El resultado de la simulación sobre los tres *Device-Under-Test* (DUT) realizada por [24], se adjunta en la Figura 2.12, en la que se puede apreciar cómo el dispositivo 1, que se encuentra en una esquina del habitáculo, posee una tasa de error muy alta, así como una relación señal-a-ruido baja en comparación con los dispositivos que se encuentran justo debajo del sistema de iluminación.

En el ámbito de la conducción también tiene cabida la comunicación por luz visible, tanto en comunicaciones entre señales de tráfico y vehículos, como entre dos vehículos (V2V). Estas ideas de comunicación traen consigo multitud de interrogantes, como por ejemplo el uso de la luz visible para comunicaciones en condiciones climatológicas adversas. Este es el caso de estudio del artículo [25], en el que se trabaja sobre comunicaciones *Vehicle-to-Vehicle* (V2V) basadas en *VLC* para condiciones de niebla. La niebla es una de las condiciones climatológicas que más condicionan el correcto desempeño de las comunicaciones de luz visible en el exterior. Se realiza un análisis sobre el color de luz o longitud de onda que resulta más adecuado para estas condiciones.

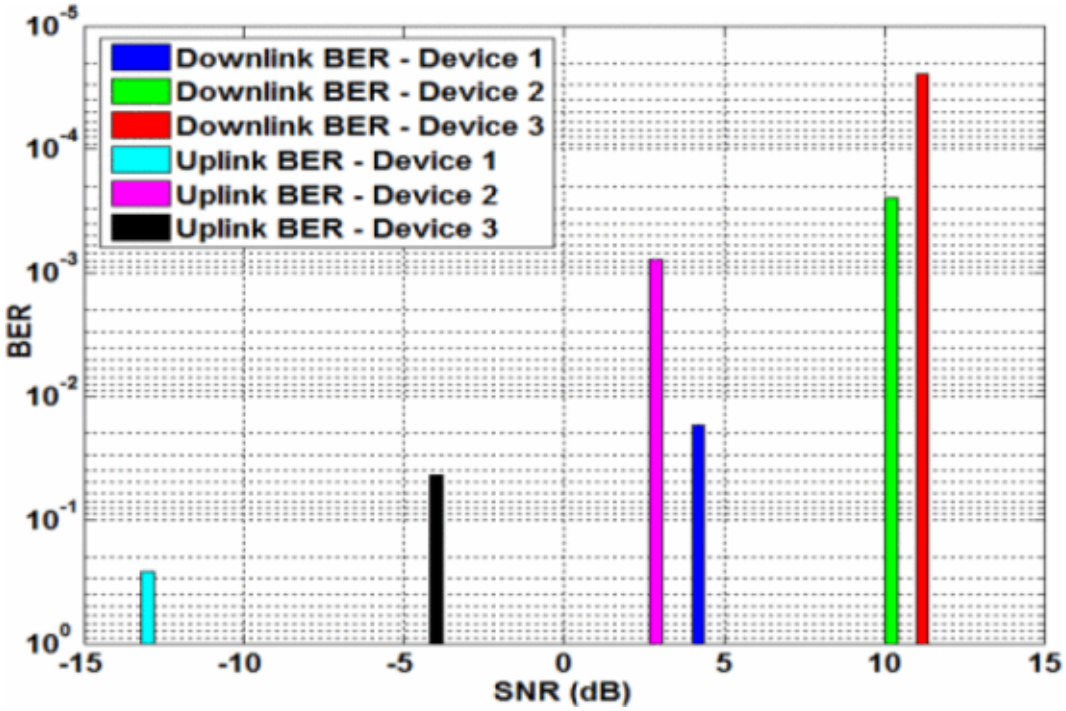


Figura 2.12: Resultados de la simulación. SNR y BER para los tres DUT.

En las comunicaciones V2V, el ángulo de incidencia sobre el receptor óptico es determinante, y con el fin de conocer esta restricción para diferentes casos de circulación, se presenta en la Figura 2.13 [25] una relación entre el ángulo de comunicación, la velocidad del vehículo, y la distancia de seguridad entre ellos.

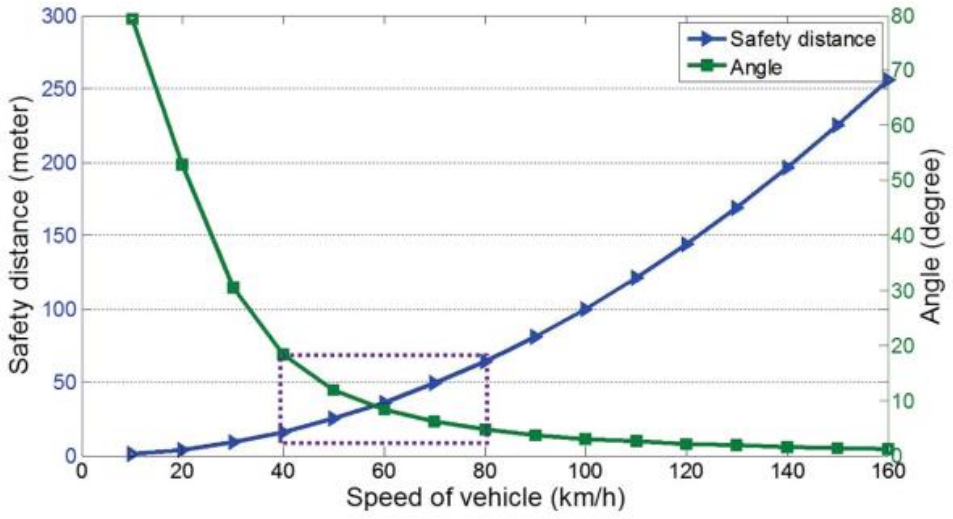


Figura 2.13: Relación velocidad del vehículo-distancia de seguridad-ángulo de comunicación VLC.

De esta gráfica se logra deducir que, por ejemplo, para una velocidad de 80 km/h y una distancia de seguridad asociada de 70 metros, el ángulo de incidencia no debe ser mayor

de 20º. Con respecto al color que menos atenuación sufre en la comunicación entre vehículos, este es el rojo (longitud de onda de 630 nm). En la Tabla 2.1 [25] se muestra la relación entre las condiciones climáticas, la visibilidad y la atenuación, en la que queda como evidencia la disminución del rango de visibilidad en condiciones adversas.

Condición climática	Visibilidad (Km)	Atenuación (dB/Km)
Despejado	20.0	0.73
Neblina	2.0	7.93
Niebla fina	1.50	10.69
Niebla leve	1.00	15.15
Niebla densa	0.50	30.29

Tabla 2.1: Visibilidad en comunicaciones VLC para diferentes condiciones climáticas.

En esta aplicación concreta se hace uso de una lente *Fresnel* con el fin de concentrar la luz en los múltiples fotodiodos y así obtener la intensidad más alta posible.

Existen aplicaciones de *VLC* cuyo objetivo no es la propia comunicación, sino la medida de atenuación. En el caso específico de [26] se aprovecha un enlace *VLC* para medir la atenuación del agua y de ahí extraer el nivel de polución existente. En la Figura 2.14 [26] se muestra la comparativa de la tensión de salida en el receptor para diferentes comunicaciones *VLC* (uso de diferentes longitudes de onda).

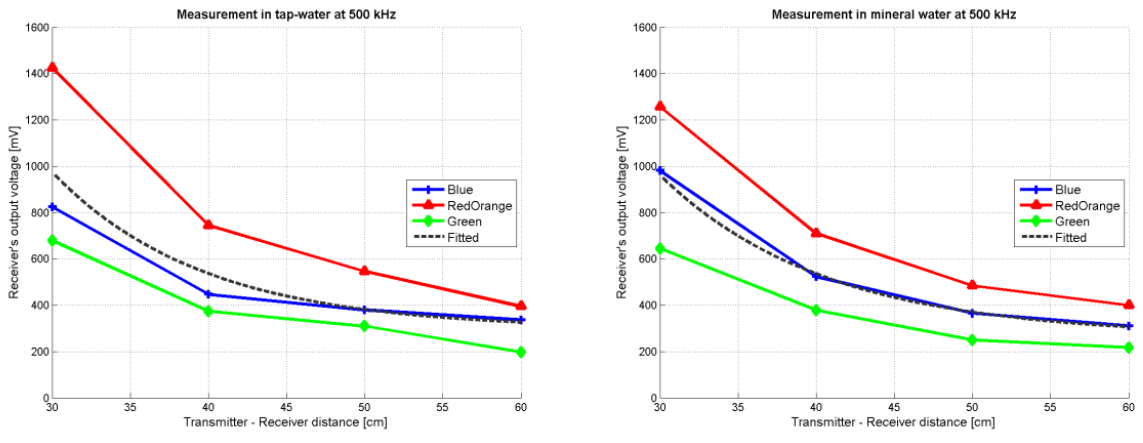


Figura 2.14: Comparativa de tensión recibida en agua del grifo y en agua mineral.

En la gráfica de la izquierda se muestra la tensión recibida en agua del grifo, mientras que la gráfica de la derecha se corresponde con la tensión medida del agua mineral.

2.2.3 *LoRa/LoRaWAN*

Con respecto al uso de la tecnología de comunicación *LoRa/LoRaWAN*, entre las aplicaciones de referencia existentes, se analizan algunas de las recogidas en la referencia [10].

LoRa/LoRaWAN es la tecnología en la que se basa una aplicación instalada en una de las comunidades más concienciadas con el medio ambiente del Líbano. A veinte minutos de la capital, Beirut, se encuentra una población llamada *BeitMisk*. Este entorno está ideado para ser ecológico, autosuficiente y lo más limpio posible, medioambientalmente hablando. Uno de los aspectos que más cuidan en esta comunidad es la calidad del aire, motivo por el que han implantado una solución basada en un sistema de sensores que envían la información a una plataforma digital, lo que permite conocer el estado del aire en la zona. A esta información tienen acceso, tanto la administración de la urbanización, como los habitantes de esta comunidad. Este sistema también cuenta con la capacidad de monitorización en tiempo real.

La solución desarrollada por la empresa libanesa *Data Consult*, junto con la española *Libelium* [27], ha llegado al despliegue de sensores por toda el área de la comunidad para lograr una mayor interacción con el entorno. Un ejemplo de esto puede ser la notificación del mejor momento del día, en términos de calidad del aire, para dar un paseo, hacer ejercicio en el exterior, o que los niños jueguen en el parque. Además del despliegue de sensores, se ha procedido a la digitalización de los elementos físicos de la ciudad, añadiendo una capa de aprendizaje automático e inteligencia artificial, ofreciendo así información sobre los datos y análisis predictivos.

Los nodos finales, a través de los sensores, miden los parámetros de temperatura, humedad y presión atmosférica, monóxido de carbono (CO), dióxido de nitrógeno (NO_2), dióxido de azufre (SO_2), ozono (O_3) y las partículas contaminantes o *Particulate Matter* PM1, PM2.5, PM10 [27].

Para la comunicación entre estos nodos finales y los *gateways,4* se utiliza el protocolo de comunicación *LoRa/LoRaWAN*. Se debe destacar que los *gateways* utilizados forman parte del despliegue de *Ogero Telecom* para la red nacional, con el fin de extender las soluciones IoT en el país. Los *gateways* están conectados con el sistema de almacenamiento online que proporciona *Google*, y desde aquí, se envía la información a la plataforma y a los servidores desarrollados por *Data Consult*. Una vez que se dispone de estos datos en los servidores, la información se procesa y refleja de manera eficaz. En este proceso también se añade una capa de inteligencia artificial que realiza análisis predictivos que asisten a los procesos de toma de decisiones. En la Figura 2.15 [27] se representa el esquema de comunicaciones que se utiliza en esta aplicación.

Otra utilidad que se puede obtener de esta aplicación, aparte del beneficio diario de los ciudadanos, se da cuando *BeitMisk* acoge conciertos, festivales de cine, u otro tipo de actividades sociales. En estos casos, el sistema permite a los responsables de los eventos y a las autoridades de la comunidad planificar mejor las actividades, proyectos o construcciones sin incrementar los niveles de contaminación.

Por lo tanto, este despliegue permite, tanto a los habitantes como a la administración de la comunidad, conectarse con su ciudad a través de herramientas de comunicación intuitivas, ahorrando recursos y respetando en todo momento el medioambiente. Los distintos usuarios pueden acceder a la información de diferentes formas, dependiendo del consumidor.

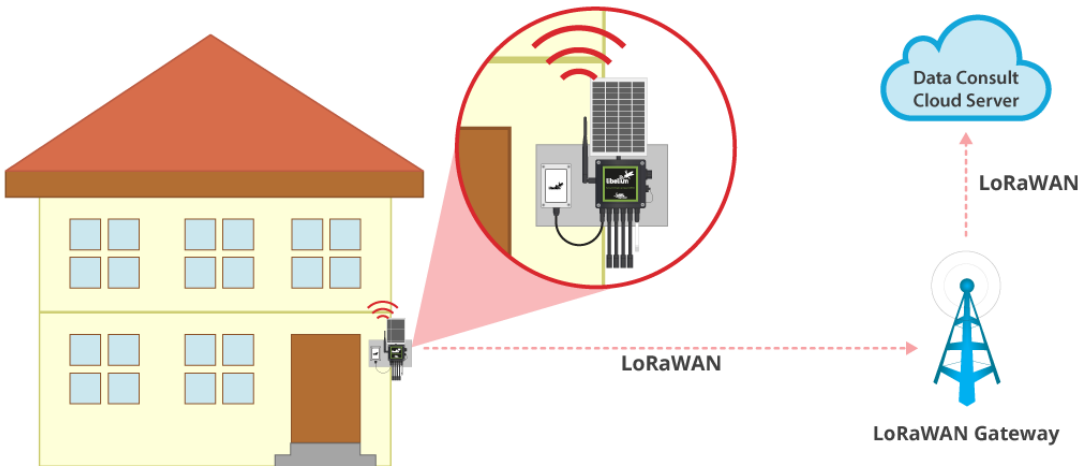


Figura 2.15: Estructura de comunicaciones en BeitMisk.

Los responsables del sistema en la comunidad pueden acceder a la información a través de *dashboard web*, una interfaz web como la que se muestra en la Figura 2.16 [27]. En cambio, los vecinos no requieren de la información completa, y por eso cuentan con una aplicación móvil que contiene directamente la información útil para ellos.

Otra aplicación que hace uso de la tecnología de comunicación *LoRa/LoRaWAN* es *InteliLIGHT* [28]. *InteliLIGHT* es una solución de gestión remota de alumbrado público y una plataforma de ciudad inteligente desarrollada por *Flashnet*. La gestión y el control de la red en profundidad permite recibir una retroalimentación en tiempo real de cualquier cambio que se produzca en la red. Contar con esta posibilidad reduce la pérdida de energía y ofrece herramientas avanzadas de optimización para el mantenimiento. Concretamente, en la referencia [28] se especifica que el uso de esta solución supone hasta un 80% de ahorro de energía y un 42% de ahorro en costes de mantenimiento. Actualmente, *InteliLIGHT* cuenta con más de 80.000 reguladores de iluminación en 10 ciudades de todo el mundo, consiguiendo así reducir el consumo de energía de iluminación y las correspondientes emisiones de CO_2.

La arquitectura que se utiliza en esta solución sigue el patrón de la aplicación anterior. Es decir, se tienen nodos finales, que en este caso serían los elementos de iluminación de la vía pública, que envían y reciben información vía *LoRa/LoRaWAN* a los *gateways* concentradores, y desde los *gateways* se envía la información a través de comunicaciones *Transmission Control Protocol/Internet Protocol* (TCP/IP) hacia los servidores, en los que se hace uso de los datos recogidos y se toman decisiones que se enviarán a los nodos finales. En la Figura 2.17 se puede ver una representación de la arquitectura descrita.

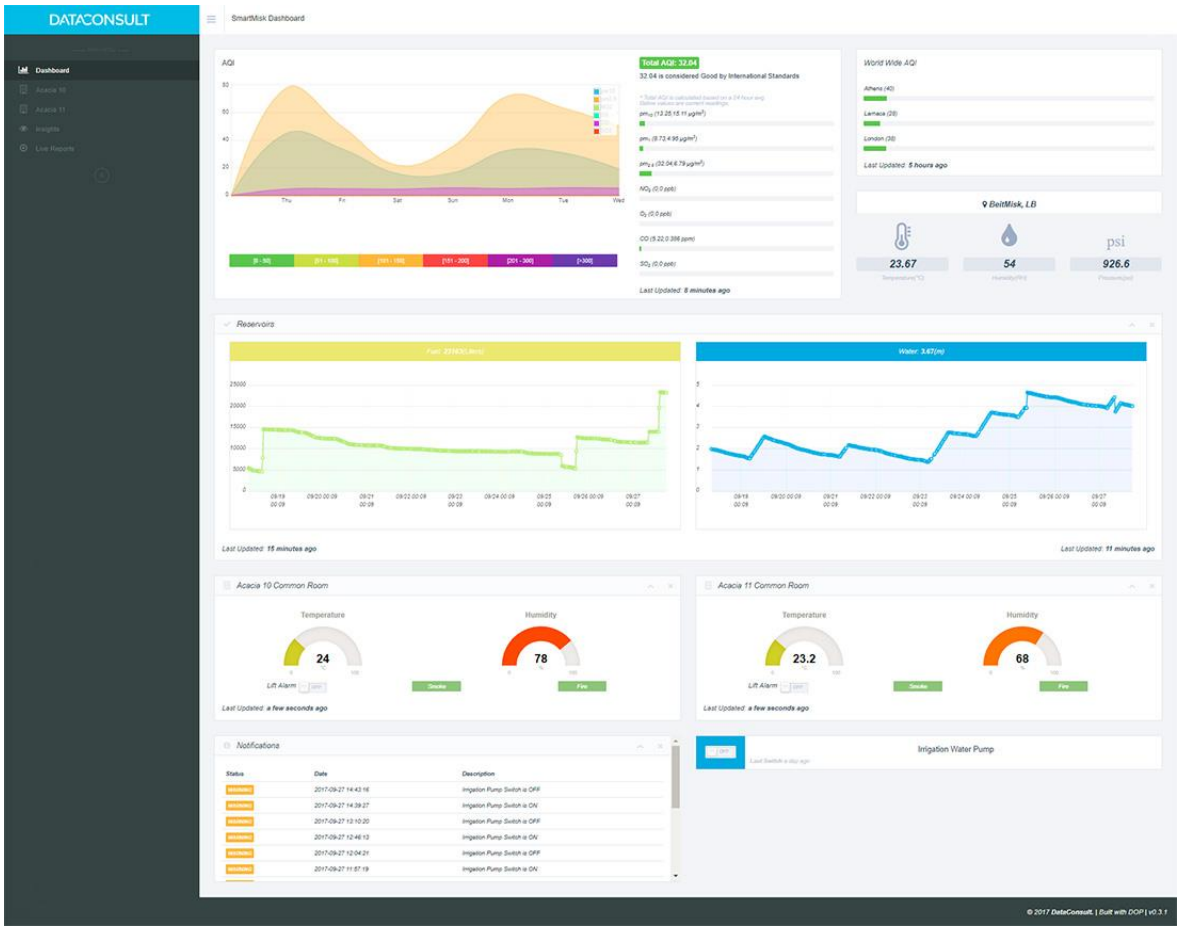


Figura 2.16: Dashboard web de la solución Smart Environment de BeitMisk.

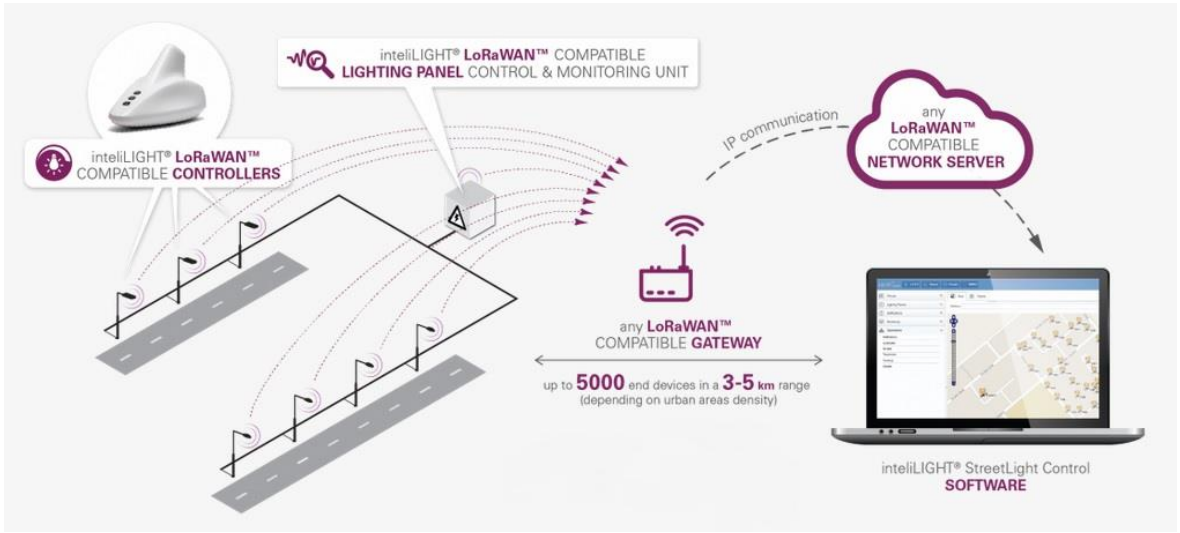


Figura 2.17: Arquitectura utilizada por la solución InteliLIGHT.

2.2.4 Integración de *Visible Light Communication* y *LoRa/LoRaWAN*

Como referencia de aplicaciones que integran las tecnologías *VLC* y *LoRaWAN*, en [8] se desarrolla una interfaz capaz de proporcionar información del entorno a través de una *Application Programming Interface* (API), y de mostrar los datos obtenidos del entorno y volcarlos en una página web. En el propio estudio se comenta que se trata de una primera aproximación hacia la integración de tecnologías de comunicación en dispositivos de tamaño reducido, y que pueden comportarse como nodos de una red. El diagrama de bloques presentado en [8] sobre la solución desarrollada, se muestra en la Figura 2.18.

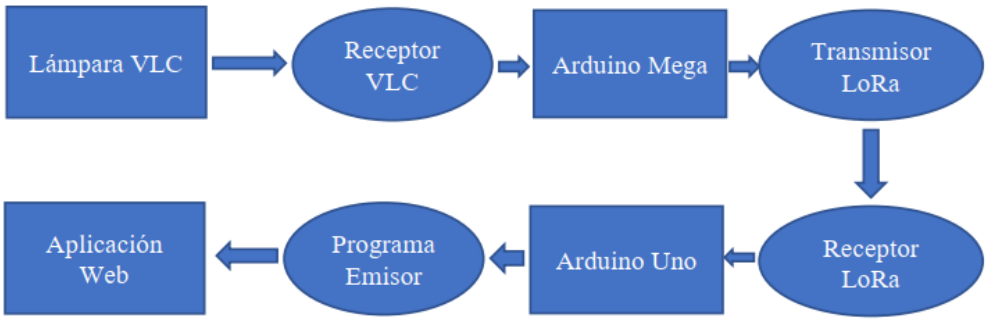


Figura 2.18: Arquitectura de interfaz VLC-LoRa.

En este diagrama de bloques se resume el sistema de comunicación propuesto. Inicialmente la lámpara *VLC* transmite información que percibe el receptor *VLC* que se integra en un dispositivo *Arduino Mega*, al igual que el transmisor *LoRa*. Desde este dispositivo se envía la información en formato *LoRa* a otro dispositivo *Arduino*, y de ahí mediante comunicación TCP/IP se vuelcan los datos en la aplicación Web. La idea de usar *LoRa* como tecnología de comunicación inalámbrica de largo alcance se fundamenta en la limitación tecnológica de *VLC* más allá de entornos de interior, dónde se reducen sus prestaciones. Usando *LoRa* se puede mantener una comunicación de largo alcance entre nodos que se encuentran en ambientes de interior y que hacen uso de las altas velocidades, la nula interferencia radioeléctrica y la eficiencia energética que ofrece *VLC*. Es decir, se realiza una pasarela entre *VLC* y LoRa/LoRaWAN que permite recibir información por luz visible (*VLC*) y transmitirla a un nodo *gateway* a través de una comunicación de largo alcance y bajo consumo de potencia, como *LoRa/LoRaWAN*.

Con respecto a esta propuesta, en el presente Trabajo Fin de Máster se pretende, además de mantener la funcionalidad de pasarela de la plataforma, integrar diferentes tecnologías como *BLE*, *VLC* o *LoRa/LoRaWAN* que permitan una comunicación bidireccional incluso de forma simultánea. Es decir, se plantea una plataforma de carácter genérica que permita la comunicación con las tecnologías integradas (*BLE*, *VLC* y *LoRa/LoRaWAN*), en cualquier sentido, de tal forma que se pueda aplicar a cualquier caso de uso dentro del ámbito de IoT. En este sentido, la solución propuesta, a diferencia de la referencia [8], se basa en una plataforma de desarrollo IoT. En el caso de la referencia [8] se utiliza una plataforma de desarrollo ineficiente para incluirla en una aplicación de IoT debido a sus múltiples prestaciones y elevado consumo de potencia, mientras que en esta propuesta de TFM se va a utilizar un dispositivo IoT como plataforma de desarrollo. Es decir, la plataforma propuesta no queda en una prueba de concepto, sino que haciendo uso de un dispositivo específico para IoT, se pretende cumplir con las restricciones de consumo de potencia que exigen las aplicaciones IoT, así como ajustarse a la funcionalidad y prestaciones requeridas.

2.3 *Smart Cities*

2.3.1 Origen

En los últimos años, tal y como se indica en [10], el éxodo de los habitantes rurales hacia las grandes ciudades está propiciando su masificación. En el año 2015 las estadísticas indicaban que más del 50% de la población mundial residía en ciudades, y estos datos no han hecho más que aumentar, previendo la Organización de las Naciones Unidas (ONU) que un 68% de población mundial estará ubicada en zonas urbanas para el año 2050 [29], lo que representa un crecimiento muy significativo teniendo en cuenta que en el año 1950 este porcentaje era aproximadamente la mitad, un 30%. Precisamente, la Organización de las Naciones Unidas publica periódicamente un estudio sobre el estado de la población a nivel mundial denominado *World Urbanization Prospects* (WUP) [29]. En este informe se recogen datos y estadísticas de interés, relativas al estado de la población mundial. Así, en el informe WUP de 2018 se indica por ejemplo cuáles son las regiones o continentes más urbanizados. Como se muestra en la Figura 2.19 [29], en primer lugar, se tiene a Norte

América (82.2% de población urbana), seguido del resto del continente americano y el Caribe (80.7%), mientras que, en tercer lugar se sitúa Europa (74.5%). En contraste con estas tres regiones, se tiene que Asia (49.9%) y África (42.5%) se encuentran por debajo de la mitad, lo que significa que tienen más población rural que urbana. No obstante, estas regiones se están urbanizando a mayor velocidad que otras, por lo que se estima que en el año 2050 el porcentaje de urbanismo en Asia y África se incremente hasta alcanzar un 64% en el caso de Asia, y un 56% en el continente africano. En lo que respecta al continente oceánico, desde 1970 hasta la actualidad no se ha detectado un cambio tan significativo como en los otros continentes, y tampoco se espera un gran crecimiento para el año 2050. Sin embargo, el porcentaje de población urbana supera la mitad, y está en un 68.2%, casi al nivel de Europa.

En general, a nivel mundial, las cifras oficiales establecen que en el año 1950 la población urbana era de 751 millones de personas, y actualmente es de 4.2 mil millones, y para el año 2050 se espera que esta cifra aumente en 2.5 mil millones de personas. Este crecimiento se espera que se produzca en un 90% en los continentes africano y asiático.

A nivel de países, el estudio también muestra datos de interés, como por ejemplo el porcentaje de población urbana y el nivel de población mundial, representado en la Figura 2.20 [29]. Así, la lista de países con mayor población urbana está encabezada por Bélgica, con un 98% de población urbana sobre la población total. Otros países con porcentajes muy elevados son Japón (92%), Argentina (92%) y Holanda (92%). A partir de estos datos se entiende que existen varios países que acumulan la mayor parte de su población en ciudades.

Con respecto a las ciudades, en 1990 había 10 ciudades con más de 10 millones de habitantes, o lo que es lo mismo, 10 megaciudades. En estas diez grandes ciudades residían un total de 153 millones de personas, menos del 7% de la población urbana mundial. Hoy en día, el número de megaciudades se ha triplicado, contabilizándose actualmente 33 megaciudades, habitadas por un total de 529 millones de personas, lo que ha incrementado el porcentaje del 7% de la población urbana mundial hasta un 13%.

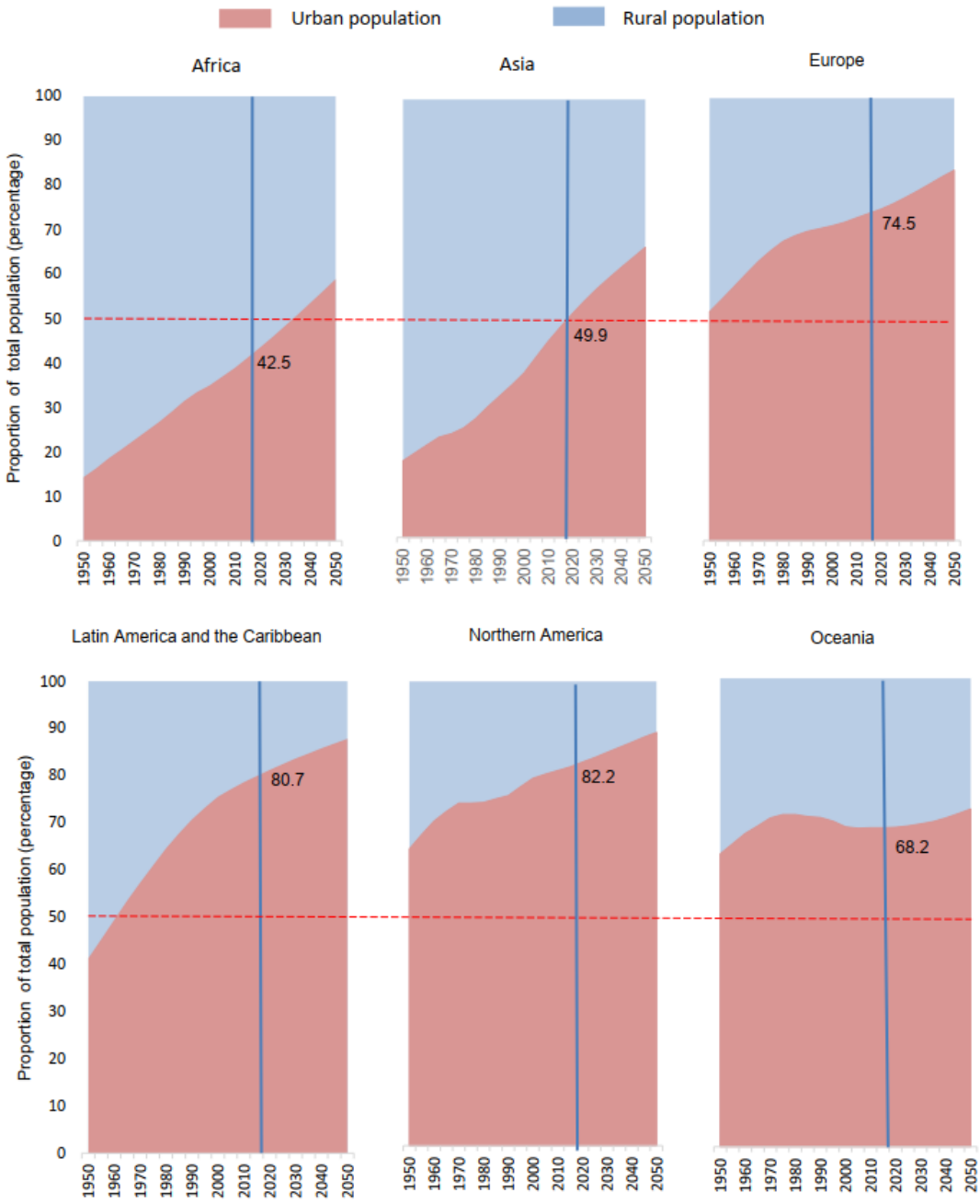


Figura 2.19: Estudio de población rural y urbana en las grandes regiones.

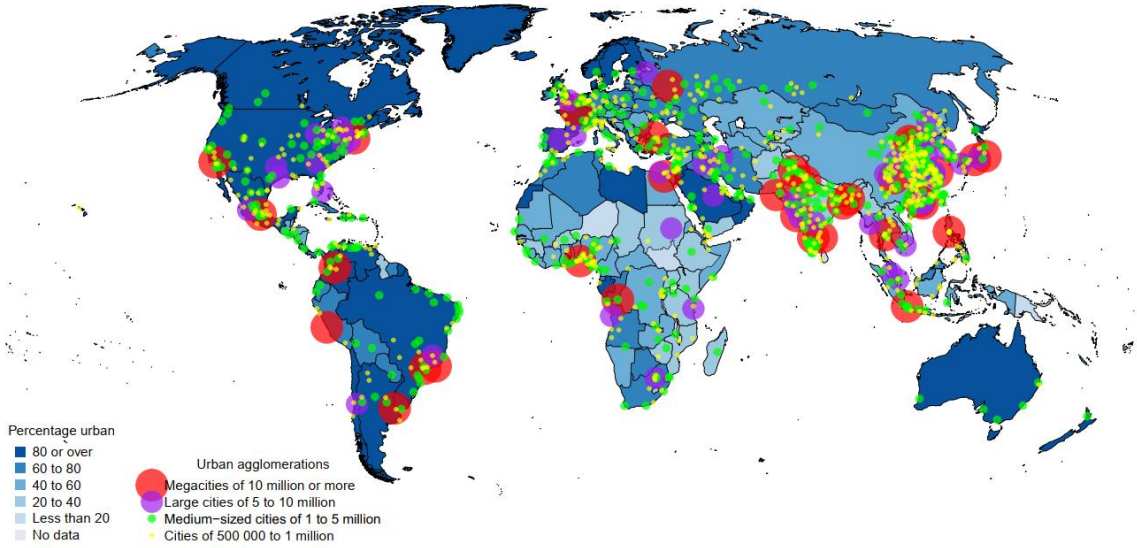


Figura 2.20: Porcentaje de urbanización y nivel de población de las ciudades a nivel mundial.

Dentro de la lista de las 33 megaciudades, se encuentran en los primeros puestos ciudades como Tokio (Japón) con 37 millones de habitantes, Delhi (India) con 29 millones, Shanghái (China) con 26 millones, y Ciudad de México (México) y São Pablo (Brasil) con 22 millones de habitantes cada una. Son las mayores ciudades en la actualidad, y se estima que lo sigan siendo en el futuro, algunas de ellas incrementando aún más su número de habitantes.

A partir de todos estos datos se puede deducir que el crecimiento de las ciudades es notable, y no solo en su número de habitantes, sino que cada vez surgen más megaciudades. Esta información se refleja en la gráfica de la Figura 2.21, extraída del informe WUP 2018 [29], en la que muestra claramente la aparición de nuevas megaciudades, y cómo han crecido en su número de habitantes. Además, el número de habitantes no se ha incrementado solamente en las megaciudades, sino que, en otros tipos de espacios urbanos, como en el caso de pequeñas áreas urbanas (menos de 500.000 habitantes), ciudades pequeñas (500.000 a 1 millón de habitantes), medianas (1 - 5 millones de habitantes) y grandes (5 y 10 millones de habitantes), también se ha visto cómo su población ha aumentado considerablemente.

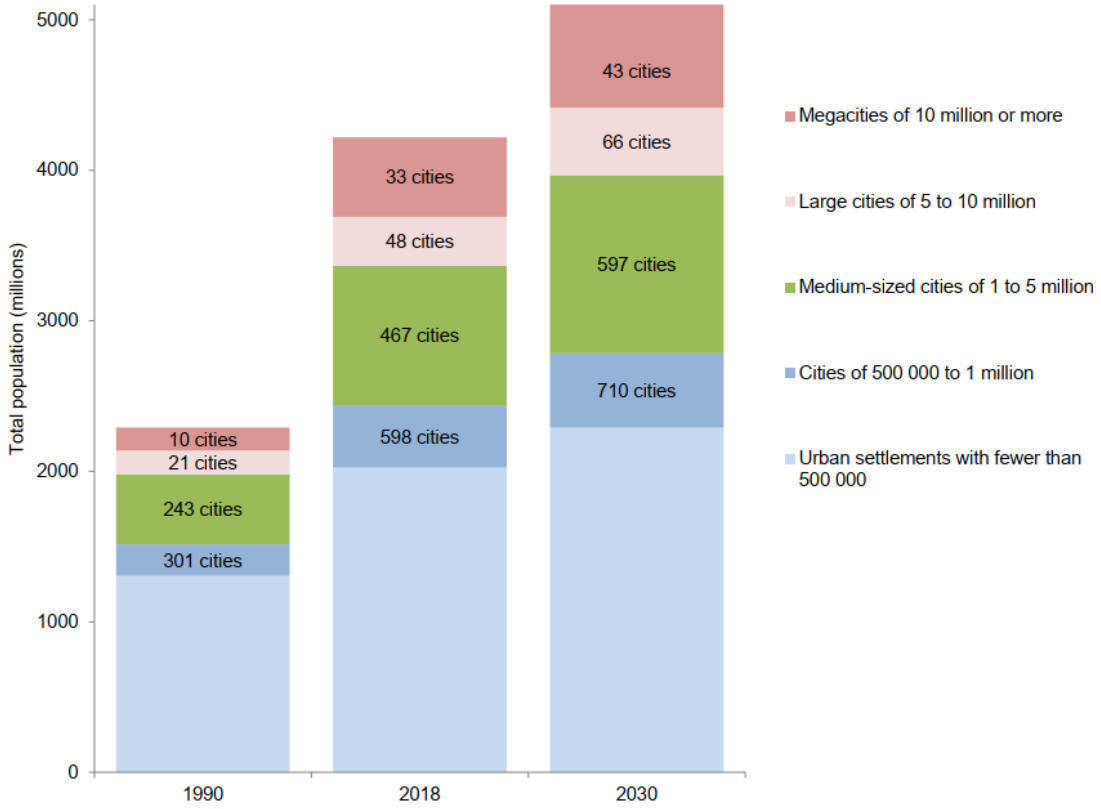


Figura 2.21: Nivel de población en los diferentes tipos de ciudades.

Ante este escenario, se puede afirmar que en estos núcleos urbanos se encuentran serios problemas de saturación, consumo de energía y movilidad. En consecuencia, una de las conclusiones que se extraen del informe WUP 2018 es que a medida que el mundo continúa urbanizándose, los desafíos asociados al desarrollo sostenible son cada vez más relevantes, sobre todo en los países de ingresos medios-bajos donde la velocidad de urbanización es mayor.

Está claro que la disponibilidad de infraestructuras, de educación, de rápido acceso al trabajo y de seguridad, aporta calidad de vida a las personas, y la ausencia de estos recursos, generan grupos humanos con baja o mala calidad de vida. Además, el crecimiento desordenado de las ciudades genera un proceso de desigualdad entre diferentes sectores sociales, pues la saturación de las zonas urbanas beneficia a unos y perjudica a otros. Todos estos problemas pueden derivar en riesgos para la salud, ya que se incrementan los niveles de estrés en las personas y provocan enfermedades, tanto físicas como mentales, que por supuesto, influyen también en la comunidad social. [30]

Esta es la razón por la que se reciben con gran aceptación soluciones tecnológicas que favorezcan la agilidad, la comodidad y el bienestar de los habitantes en grandes metrópolis.

Bajo estas circunstancias surge el concepto de *Smart City*, o Ciudad Inteligente que se presenta como una solución o alternativa al modo de gestión y control de diferentes sistemas determinantes dentro de las ciudades. Por lo tanto, es necesario transformar las ciudades tradicionales en *Smart Cities*, y con la evolución de la tecnología que se experimenta en la actualidad, esta transformación es cada vez más viable.

2.3.2 Definición de *Smart City*

Las grandes ciudades y las áreas metropolitanas se pueden entender como sistemas complejos con conexiones entre sus diferentes sectores e individuos. Este concepto viene definido en [10], referencia citada textualmente en este apartado. Por esto la planificación urbana y el desarrollo de mecanismos de decisión dinámicos resultan cada vez más importantes. Para realizar una buena planificación y desarrollar los mecanismos de decisión correctos es necesario conocer lo que sucede en las ciudades y tener capacidad para actuar sobre ellas. [31]

Las ciudades son sistemas complejos que se gestionan en tiempo real y que generan una gran cantidad de datos. A partir de estos datos se debe hacer un uso inteligente de las Tecnologías de la Información y las Comunicaciones (TIC) con el fin de afrontar los retos que presentan las ciudades del presente y del futuro.

La idea que subyace en el concepto de *Smart Cities* consiste en aprovechar el potencial de los avances tecnológicos para conseguir mejores resultados en diferentes ámbitos de la gestión de una ciudad. Con esta idea, lo que se pretende es ahorrar costes, un objetivo que solo se puede conseguir priorizando la eficiencia en todos los sistemas que componen las *Smart Cities*.

Si bien no existe una definición común de *Smart City*, ya que puede resultar un término un poco ambiguo, como definición simple se puede decir que se trata de: “Una ciudad que usa las Tecnologías de la Información y las Comunicaciones (TIC) para proporcionar una mejor calidad de vida a sus ciudadanos.” [31]

Otra definición más amplia y exacta, realizada por el experto en *Smart Cities*, Boyd Cohense [32], acota aún más el concepto de ciudad inteligente: "Las *Smart Cities* utilizan las TIC para ser más inteligentes y eficientes en el uso de recursos, reduciendo costes y ahorrando energía, mejorando los servicios proporcionados y la calidad de vida, y reduciendo la huella medioambiental, todo ello con la ayuda de la innovación y una economía baja en carbono."

Así, se puede establecer que una *Smart City* es un sistema compuesto por muchos subsistemas, ya que solo el desarrollo tecnológico en múltiples ámbitos de la ciudad consigue que ésta se considere inteligente. Por cada ámbito de la ciudad que requiera de gestión, control y seguimiento se puede encontrar una buena oportunidad para desarrollar un subsistema inteligente. La agrupación de todos los subsistemas inteligentes en la misma ciudad es lo que se conoce como el sistema inteligente completo de la ciudad, la *Smart City*. A partir de esta definición se entiende que existe un gran número de ámbitos en los que se pueden desarrollar soluciones inteligentes, es más, es imposible enumerar esta cantidad, pues surgen oportunidades de desarrollo urbano en cada pequeña necesidad social. Por esto, se engloban muchos de los ámbitos de desarrollo en términos más generales que permiten diferenciar bien los diferentes subsistemas que componen una *Smart City*. Los principales subsistemas que se pueden encontrar en una *Smart City* son los que se muestran en la Figura 2.22. En esta imagen se visualiza cómo el sistema completo o *Smart City* está compuesto por diferentes subsistemas como son *Smart Mobility*, *Smart Enviroment*, *Smart Grid*, *Smart Building and Home* o *Smart Government*. Existen tantos subsistemas como necesidades tienen los ciudadanos y las instituciones, pero estas cinco dimensiones copan la mayor parte de aplicaciones desarrolladas. En el presente TFM se desarrolla una plataforma genérica que bien puede emplearse para muchos ámbitos de las *Smart Cities* con mínimas modificaciones, pero con el objetivo de centrar su desempeño en un caso de uso concreto el subsistema escogido es *Smart Mobility*, y en concreto en el concepto de *Smart Parking*.

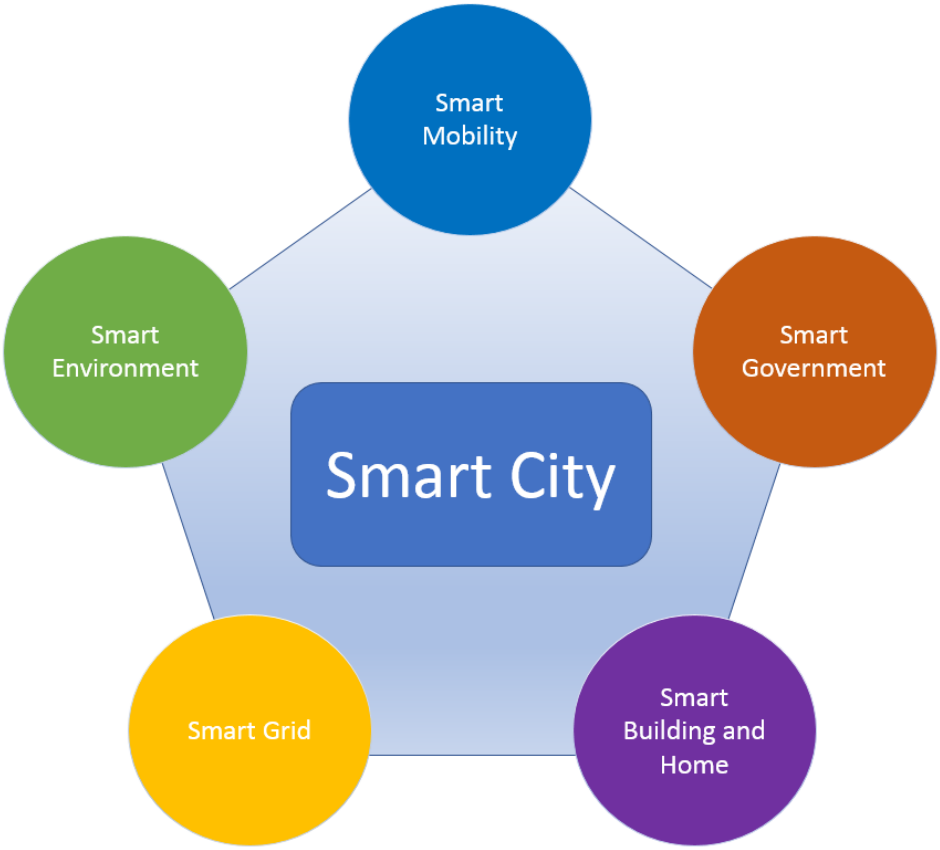


Figura 2.22: Smart City como sistema de subsistemas.

El auge de IoT repercute directamente en el concepto de ciudades inteligentes *o Smart Cities*. En *Smart Cities*, la comunicación es un elemento fundamental, y haciendo uso de las ideas y aplicaciones IoT, se pueden completar las comunicaciones de una forma robusta y eficiente. Además, para las diferentes aplicaciones se puede optar por una solución IoT u otra, dependiendo de las características que demande en cada caso. La idea de IoT se ajusta perfectamente al modelo de *Smart City*. La interconexión de los objetos en una ciudad posibilita la comunicación entre ellos, y, por tanto, se permite aumentar el nivel de automatismo e independencia de los sistemas. Además, haciendo uso de soluciones IoT, se logra establecer comunicaciones entre los objetos y los usuarios, facilitando así muchos procesos. [10]

A continuación, se muestran ejemplos de *Smart Cities*. En muchas dimensiones dentro de las *Smart Cities* se encuentran proyectos de IoT que potencian la idea de la conectividad en la mayoría de los elementos existentes en las urbes, así como del uso eficiente de energía.

- Londres. Según el informe anual [33], Londres ocupa el primer puesto de ciudad inteligente a nivel mundial. Este puesto en el ranking se debe a diferentes ámbitos en los que la ciudad inglesa destaca. Por ejemplo, en movilidad y transporte. Contando con uno de los espacios aéreos más transitados del mundo, y, por lo tanto, también con un aeropuerto muy recurrido, las proximidades a este suelen encontrarse congestionadas. Ante esta situación se desarrolló un sistema de transporte eléctrico denominado *Heathrow pod* que transporta a los usuarios desde la zona de aparcamiento de su vehículo particular hasta la terminal 5, o viceversa. De esta forma se descongestiona el tráfico en las inmediaciones del aeropuerto, con todas las ventajas que eso supone para las personas y para el medio ambiente. En la Figura 2.23 se muestra el vehículo eléctrico utilizado.

  Aparte de desarrollar innovaciones en materia de transporte y movilidad, también se encuentra en los primeros puestos del ranking debido a su factor social o capital humano. Esta dimensión, viene logrando un buen desempeño en mayor medida gracias a sus universidades (que figuran entre las mejores del mundo) y centros culturales. Además, es de las ciudades de las que más *startups* alberga y que cuenta con más programadores, además de poseer una plataforma de datos abiertos (*London Datastore*) que utilizan mensualmente más de 50.000 ciudadanos, compañías, investigadores y desarrolladores [33].

Figura 2.23: Heathrow pod.

- Nueva York. Esta ciudad norteamericana se encuentra en el segundo puesto del ranking realizado por [33]. Aparte de ser la urbe con mayor Producto Interior Bruto (PIB), es además una de las más "inteligentes". Entre sus ideas innovadoras se encuentran sistemas de iluminación inteligente, sistema de gestión de agua en todos los edificios públicos o adaptación de servicios a las personas mayores. Una aplicación que cuenta con el uso de sensores conectados es el del posicionamiento de los disparos con armas de fuego. Los sensores permiten realizar un proceso de triangulación con el objetivo de ubicar el punto dónde se realizó el disparo. Esta solución es muy importante para esta ciudad en concreto, ya que el 75% de los disparos no son notificados.

  Por otra parte, la movilidad está regulada. Los semáforos poseen un tiempo de transición adaptativo en función del tráfico de la zona. Dicho tráfico se detecta a través de cámaras que funcionan en tiempo real.

  Otra idea ha sido sustituir las cabinas telefónicas en tótems de carga móvil, con conectividad *WiFi*, botón de emergencias y pantallas táctiles con información de interés ciudadana [33].

- Como tercer ejemplo para resaltar la importancia del binomio de las tecnologías IoT con el concepto de *Smart City*, se presenta una solución *Smart Environment*. Se trata de la gestión inteligente de los residuos, una idea que está siendo explotada por *eCube* [34], una empresa surcoreana que está recibiendo muy buena aceptación en todo el mundo.

  El tratamiento de residuos que la compañía propone se basa en la utilización de contenedores de basura inteligentes, como los mostrados en la Figura 2.24 [34]. Estos contenedores detectan si están llenos, y en ese caso avisan a los encargados de su recogida. De esta manera, se puede trazar una ruta óptima para la recogida de basura diaria y no mantener una ruta fija e ineficiente en muchas ocasiones. Además, la solución hace uso de datos históricos para ofrecer análisis predictivos que permitan una mejor planificación. Bajo esta idea innovadora, los costes operativos se pueden reducir hasta en un 80%. Una limpieza bien planificada y

precisa, favorece el entorno de la ciudad, al no dejar las calles abarrotadas de basura, como ocurre en muchos casos. Los contenedores además se alimentan con energía solar.

En este ejemplo la solución es claramente del tipo IoT, y se respetan los pilares fundamentales de *Smart Cities*. [10]

Figura 2.24: Contenedores de basura inteligentes.

- A nivel nacional, viene destacando desde hace tiempo la ciudad de Santander. Desde 2011 se empezó a trabajar en un proyecto denominado *SmartSantander* [35] en el que se recoge información del entorno a partir de multitud de sensores. El proyecto consiste en dotar a la ciudad de múltiples sensores para que recojan información que posteriormente será utilizada. Estos sensores se colocaron en fachadas de edificios, marquesinas y otros puntos estratégicos de la ciudad. El número de sensores implantados se aproxima a la cifra de 15.000, y toda la información que recogen, se envía a una base de datos, donde herramientas de *Big Data* ordenan y clasifican la información. Esta información está destinaba a mostrar datos sobre la calidad del aire, estado de los contenedores, tráfico y circulación, y otros aspectos de interés en el ámbito de la ciudad cántabra. Con toda esta información se pueden tomar mejores decisiones por parte de las autoridades responsables de la ciudad, y facilitar información eficaz al ciudadano.

En la Figura 2.25 se muestra la arquitectura planteada en el proyecto *SmartSantander*, en la que se pueden diferenciar tres niveles:

- Nodos finales: Incluyen los sensores y si fuera el caso, también los actuadores.
- Nodo concentrador o *gateway*: Encargado de comunicarse con los nodos finales y hacer de enlace con los servicios de Internet.
- Plataforma: Donde se gestionan los datos. Sobre este nivel suelen trabajar las estrategias de *Big Data*. Desde aquí se ofrece la información.

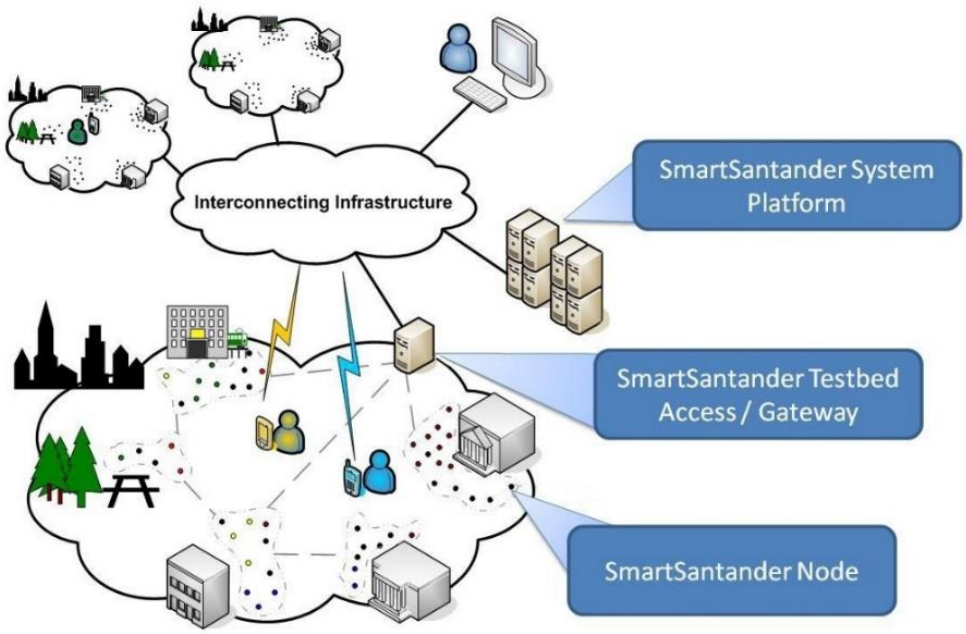


Figura 2.25: Arquitectura de SmartSantander.

2.4 *Smart Mobility y Smart Parking*

En el presente TFM se desarrolla una plataforma genérica que bien puede emplearse para muchos ámbitos de las *Smart Cities* con mínimas modificaciones, pero con el objetivo de centrar su desempeño en un caso de uso concreto, el subsistema escogido es *Smart Mobility*, y en concreto en el concepto de *Smart Parking*.

2.4.1 Estado de la movilidad

En la referencia [10] citada textualmente se estudia el estado actual de la movilidad. La movilidad y el tráfico representan un gran desafío para las ciudades del futuro, ya que la mayoría de las ciudades han ido construyéndose según la necesidad de acoger a personas que deciden cambiar los pueblos o zonas rurales por grandes metrópolis. En esta expansión, se construyen muchos hogares, pero se descuidan aspectos como la movilidad. Así, se incrementa el número de habitantes, y el número de viviendas, pero no se mejoran ni ensanchan las carreteras o vías urbanas, causando así saturaciones que dificultan el transporte.

En las grandes ciudades de España, como en el caso de Madrid, el aumento de población y la falta de adaptación de las ciudades a este gran cambio está generando enormes problemas de movilidad y por tanto de contaminación. La estadística dice que cada conductor que reside en la capital española sufre aproximadamente 42 horas de atascos al año. Este es un gran indicativo de la cantidad de tráfico existente en las grandes ciudades. Por supuesto, el número de desplazamientos ya genera elevados niveles de contaminación, pero si a eso se le suma la falta de agilidad de los vehículos por el colapso de las carreteras, las emisiones de gases nocivos para el medioambiente se incrementan considerablemente. De hecho, el 41% de las emisiones de gases de efecto invernadero en el municipio de Madrid están relacionadas con el transporte, lo cual equivale a 7 toneladas métricas de emisiones de dióxido de carbono [36].

Otro apunte importante es que, en la mayoría de las grandes ciudades, el transporte más utilizado es el vehículo privado. Volviendo al caso de la ciudad de Madrid, el 80% de la contaminación generada por transporte se debe al uso de vehículos privados. Si se considera el número de kilómetros que se recorren aproximadamente en una ciudad como la madrileña, esta cifra alcanza los 23.000 millones de Km al año, que se reparten como se muestra en la Figura 2.26. Como se puede ver en esta gráfica los vehículos privados son el transporte que más Km recorren en comparación con el resto.

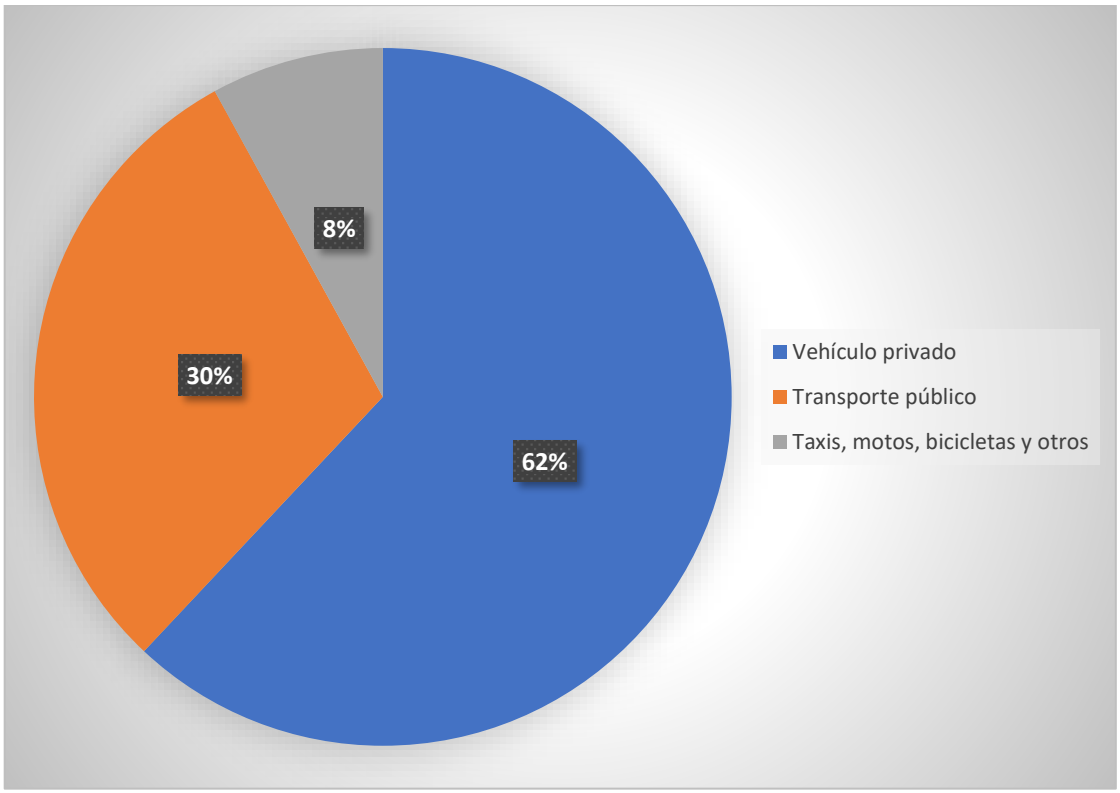


Figura 2.26: Porcentaje de Km recorridos al año por cada tipo de medio de transporte en Madrid.

Para mejorar el sistema actual de movilidad y transporte en las grandes ciudades, se plantean cuatro grandes cambios. Estos grandes cambios, recogidos en la siguiente lista, son al fin y al cabo los objetivos principales que concentran la idea de *Smart Mobility*:

- Mejorar el servicio de transporte público.
- Diseñar y fomentar el uso de vehículos sostenibles.
- Facilitar el estacionamiento de los vehículos.
- Agilizar la circulación.

2.4.2 *Smart Parking*

El presente Trabajo Fin de Máster tiene como caso de uso de la plataforma genérica desarrollada el campo de *Smart Mobility*. Como se indica en la referencia [10], citada textualmente, las soluciones *Smart Parking* son cada vez más demandadas, ya que

actualmente un 30% del volumen del tráfico en el centro de las ciudades está relacionado con la búsqueda de aparcamiento [37], dejando a los conductores durante una media de 20 minutos en la ardua tarea de encontrar un espacio disponible para estacionar su vehículo.

Con soluciones de *Smart Parking* se pretende simplificar esta tarea mediante diferentes procesos tecnológicos garantizando la sostenibilidad, el bienestar medioambiental, la eficiencia, y por supuesto, el ahorro de tiempo de las personas, evitando de esta manera malestar en ellas, que puedan desencadenar conductas negativas y conflictivas en la conducción. Estas conductas negativas no se quedan en el momento del estacionamiento, sino que toda la tensión acumulada repercute directamente en la salud de los conductores, tanto física como mentalmente. Muchas veces, además, la búsqueda de aparcamiento se complica en hora punta, provocando que los trabajadores lleguen tarde a sus trabajos u obligaciones, y se reduzca la productividad.

Existen diferentes soluciones desarrolladas en este ámbito de las *Smart Cities*. Muchas de ellas se encuentran en fases de prueba, o tratando de optimizarse, por lo que queda mucho camino por recorrer en este ámbito para tratar de alcanzar un sistema de aparcamiento cómodo, sencillo, rápido y accesible. Actualmente las principales herramientas que se utilizan en *Smart Parking* son [38]:

- *Software* de análisis de fotos y vídeos: Comprueban la ocupación de las celdas y detectan vehículos estacionados de forma ilegal.

- Sensores de proximidad: Sensores que detectan el estado de la plaza, suelen ser sensores infrarrojos.

- Paneles *Light-Emitting Diode* (LED): Soporte informativo que muestra el estado de las plazas de estacionamiento.

- Parquímetros inteligentes: Recogen la información sobre las plazas que están ocupadas, además de cumplir con su función habitual.

- *Tags RFID*: Contienen información asociada al dueño del vehículo, aspecto útil para promociones o agilidad en el pago. La información se transmite mediante campos electromagnéticos, con la ayuda de lectores que normalmente utiliza el personal de la zona de aparcamiento.

- Aplicaciones web y móvil: Con capacidad para realizar reservas, pagos y renovaciones desde cualquier dispositivo con conexión a Internet.

Con todas estas soluciones, que son algunas de todas las que están desarrollándose, y de las que están por venir, se logran unos beneficios importantes. Según [38] se reduce en un 43% el tiempo empleado en buscar aparcamiento y un 30% menos de Km recorridos en esta tarea. Además, con estas ideas de *Smart Parking* se consigue también una base de datos con la información de los diferentes comportamientos, horarios de uso más habituales, zonas más frecuentadas, etc. En resumen, información sobre el usuario que permita generar soluciones más aproximadas a las necesidades reales del conductor.

En la Figura 2.27, extraída también de [38] se muestran en cifras los beneficios de hacer uso de las soluciones de *Smart Parking*. En esta gráfica se refleja la reducción de emisiones, la reducción de aparcamiento circunstancial en zonas de circulación, como en un segundo carril y la descongestión del tráfico general.

A continuación, se muestran ejemplos de soluciones ya desarrolladas sobre el ámbito de *Smart Parking*.

La compañía de Reino Unido *SmartParking* [39] ha desarrollado una plataforma similar en algunos aspectos a la que se presenta como caso de uso en este TFM. Los aspectos que mantienen en común son principalmente los beneficios que se desean aportar al conductor.

En este caso se hace uso de una aplicación móvil, que se recomienda consultar antes de iniciar la conducción para localizar las plazas de aparcamiento que se encuentran disponibles. Esta misma aplicación, una vez comenzado el trayecto, va dando indicaciones para llegar a la plaza haciendo uso de tecnologías de localización, como GPS. Una vez que el vehículo haya estacionado, el sensor reúne información de este para el pago, el

cumplimiento y la gestión del espacio. El sensor utilizado es de carácter magnético, aparte del uso de infrarrojos.

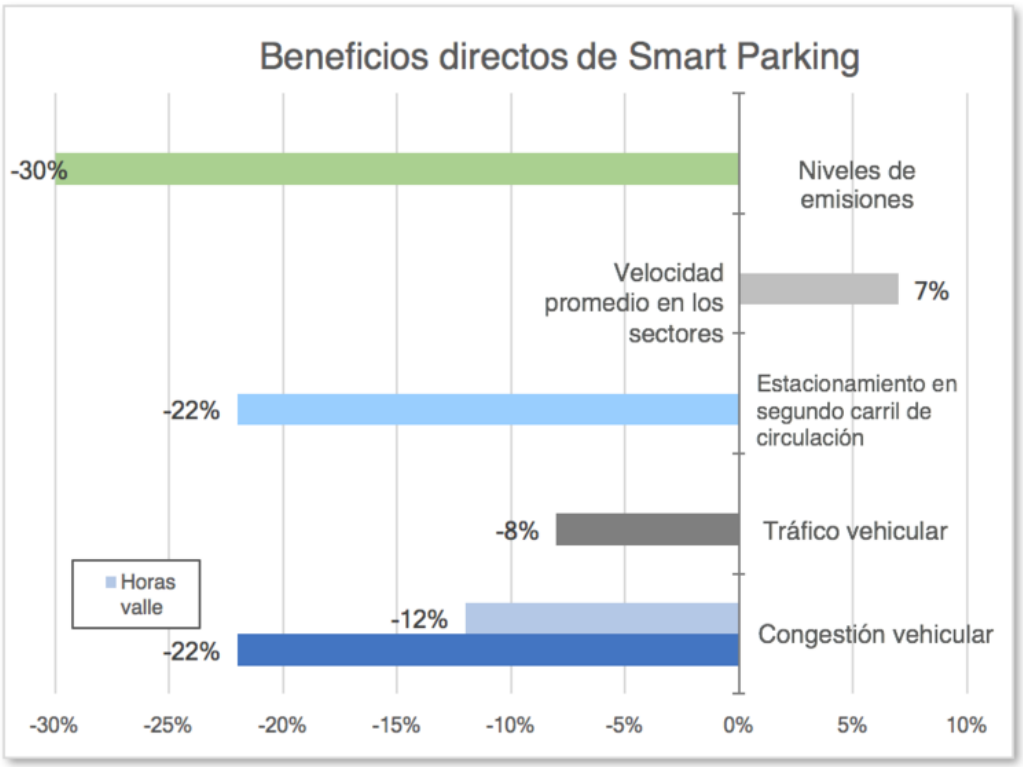


Figura 2.27: Beneficios directos de Smart Parking.

Además de la aplicación y del sensor ubicado en la plaza de aparcamiento, se cuenta con un *software* para la gestión y administración de las plazas que permite realizar planificaciones diariamente y, además, a largo plazo. Para este estudio a largo plazo, es necesario la recopilación de una cantidad significativa de información que se recoge en cada estacionamiento, almacenando datos, como la hora o la duración del estacionamiento. En la Figura 2.28 [39] se muestra como referencia una captura de este *software*, denominado *SmartRep*.

Otro servicio parecido es el proporcionado por la empresa I+3D. La gran diferencia con la solución anterior es que el epicentro de los servicios ofrecidos es la gestión y control de técnicas de *Big Data* sobre toda la información recogida. El proyecto 3SCPARK [40] se presenta como "un *Big Data* que permite a los sistemas conectados, compartir datos y vincular servicios para facilitar la movilidad de los ciudadanos". Debido a la arquitectura que utiliza esta solución IoT, se dispone de un sistema centralizado que integra todos los

servicios que un aparcamiento inteligente necesita. En la Figura 2.29 se muestra la arquitectura utilizada en el proyecto 3SCPARK.

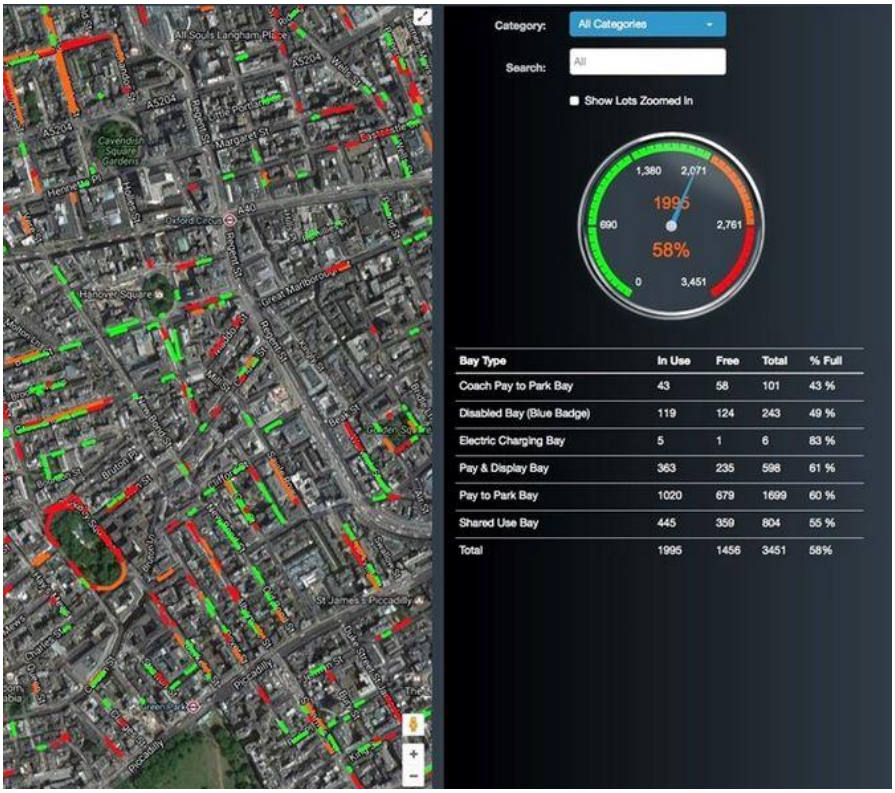


Figura 2.28: SmartRep. Software de aparcamiento inteligente.

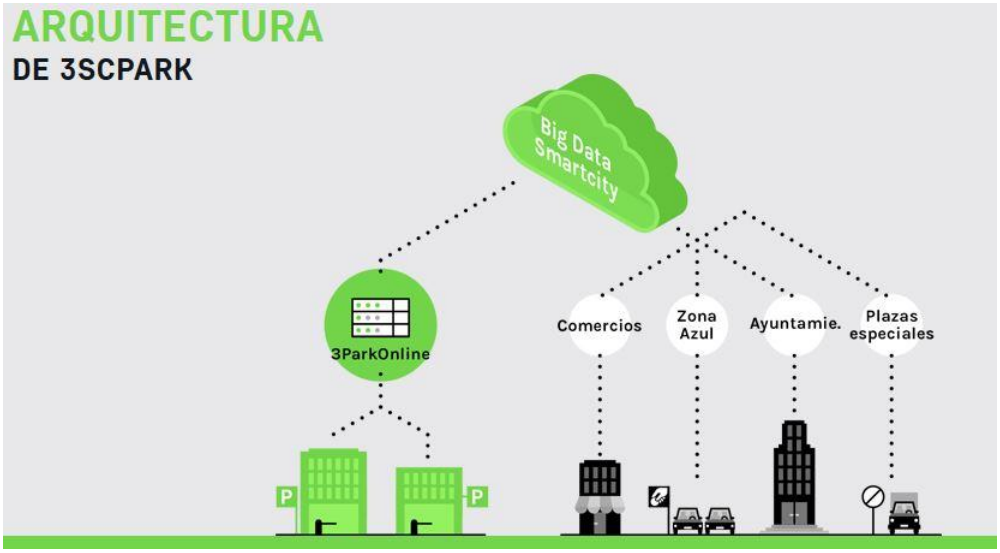


Figura 2.29: Arquitectura de 3SCPARK.

Al tener todos los elementos conectados a *Big Data* 3SCPARK, incluidas las zonas azules o comercios, se pueden vincular diferentes servicios, facilitando siempre el proceso al

usuario. En resumen, se tienen cuatro elementos conectados constantemente al sistema 3SCPARK:

- Aplicación móvil: A través de la aplicación los clientes pueden buscar aparcamiento, abrir la barrera y pagar el estacionamiento.

- Portal web: Se puede buscar toda la información relacionada y realizar reservas.

- Centro de control: Para controlar y gestionar toda la infraestructura de los aparcamientos desde un único lugar.

- Parking: El propio parking está conectado, y al tratarse de un sistema centralizado, permite integrar más de un aparcamiento, pudiendo gestionarlos y controlarlos de manera conjunta.

Por último, se presenta una aplicación *Smart Parking* de interior. En este caso la solución propuesta en [41] trata dos problemas habituales de este tipo de sistema: La detección del estado de la plaza y la navegación. Para el primer problema se plantea el uso de sensores geomagnéticos, y para la navegación se utiliza la tecnología *WiFi*.

Se divide el sistema en dos módulos, uno para la detección y otro para la navegación. Para la detección los sensores geomagnéticos deciden el estado de la plaza de estacionamiento a partir de su medida del entorno y la ejecución de un algoritmo de detección.

En el segundo módulo se utiliza la tecnología *WiFi* como tecnología de posicionamiento en interior. Para ello se hace uso de una red de sensores inalámbricos que permite averiguar la posición de los vehículos que se encuentran en movimiento y transmitir esa información al servidor.

Una vez que los datos llegan al servidor, estos se procesan y se envía información efectiva a los usuarios del sistema a través de terminales móviles inteligentes. En la Figura 2.30 se muestra la arquitectura del sistema.

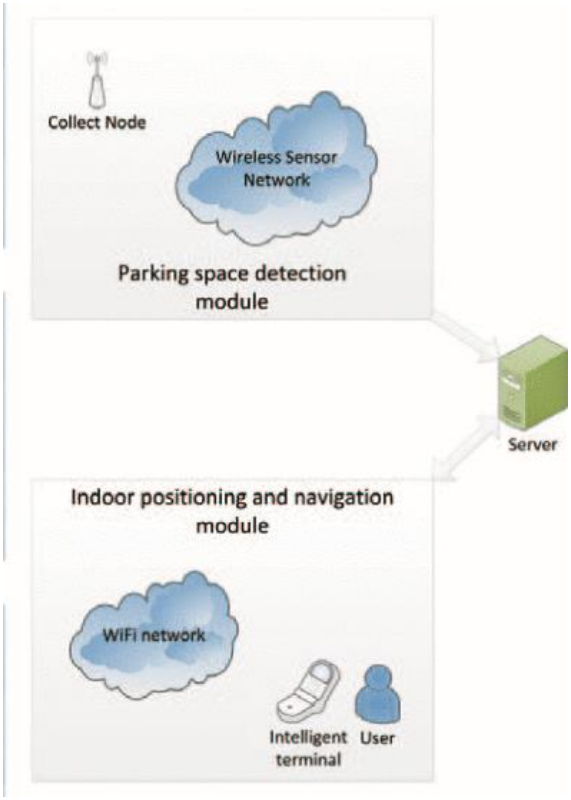


Figura 2.30: Arquitectura de un sistema Smart Parking Indoor.

Esta solución permite encontrar aparcamiento a través de información en tiempo real, ahorrando de este modo tiempo a los conductores y reduciendo las emisiones contaminantes de los vehículos. El problema puede estar en que para llevar a cabo una solución de este tipo haría falta implementar una infraestructura *WiFi*, así como contar con conectividad en los terminales móviles dentro de la zona de estacionamiento.

2.5 Descripción de la plataforma HW/SW propuesta

En la Figura 2.31 se presenta la arquitectura de la plataforma HW/SW planteada en el desarrollo del presente TFM.

Como se comentó en el capítulo de introducción, la plataforma propuesta integra las tecnologías de comunicación *VLC*, *BLE* y *LoRa/LoRaWAN* con el fin de proporcionar una solución IoT versátil y configurable para múltiples aplicaciones. Las tecnologías de comunicación consideradas permiten establecer la comunicación bidireccional entre los diferentes tipos de nodos de la arquitectura.

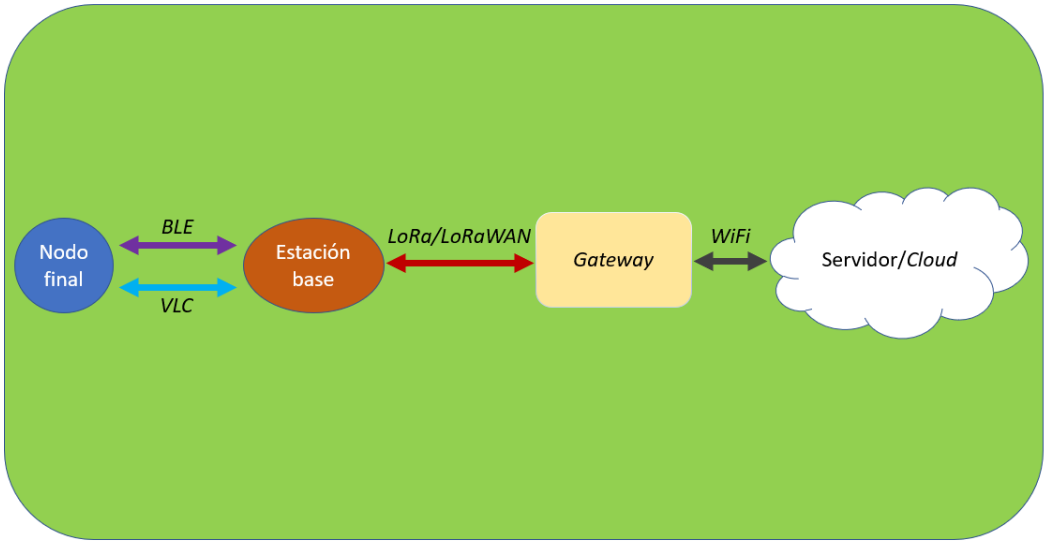


Figura 2.31: Arquitectura de la plataforma HW/SW propuesta.

Los diferentes tipos de nodos que se han desarrollado en el presente TFM son:

- Nodo final: Este nodo, mediante el uso de un sensor integrado recoge información del entorno, y haciendo uso de *BLE* o *VLC* la transfiere al nodo estación base. Del mismo modo, pero en sentido contrario de la comunicación, este nodo puede recibir información desde la estación base mediante *BLE* o *VLC*. Este nodo es alimentado por batería, por lo que su funcionalidad debe acotarse a restricciones de consumo de potencia.

- Nodo estación base: Este nodo, realiza la función de intermediario entre el nodo final y el nodo *gateway* o concentrador. Es decir, la información que reciba desde el nodo final mediante *BLE* o *VLC* se retransmite hacia el nodo *gateway* usando la tecnología de comunicación *LoRa/LoRaWAN*. Del mismo modo, pero en sentido contrario de la comunicación, este nodo puede recibir información desde el nodo *gateway* mediante *LoRa/LoRaWAN* y retransmitirla hacia el nodo final utilizando *BLE* o *VLC*. En este caso, el nodo puede ser alimentado de forma ininterrumpida. No obstante, se utilizan tecnologías de bajo consumo de potencia con el fin de hacer un uso energéticamente eficiente del nodo.

- Nodo *gateway*: Este nodo, recoge los mensajes recibidos desde los nodos estación base mediante *LoRa/LoRaWAN* y los envía vía *WiFi* hacia el servidor que almacena y procesa la información. Del mismo modo, si desde este servidor se desea enviar un mensaje hacia un nodo final, el nodo *gateway* recibe el mensaje vía *WiFi* y lo traslada hacia el nodo estación base mediante *LoRa/LoRaWAN*.

Además de los tres tipos de nodos, en esta arquitectura existe un cuarto elemento, el servidor o plataforma. En este caso este servidor es *The Things Network* (TTN), una plataforma que permite recoger los mensajes recibidos, decodificarlos y exportarlos hacia unas aplicaciones externas. Desde TTN, también se puede codificar mensajes y enviarlos hacía el *gateway* correspondiente. Es decir, esta plataforma se encuentra en continua comunicación con el nodo *gateway*. Esta comunicación se realiza vía *WiFi*.

El propósito de este TFM es presentar una plataforma de IoT genérica que posea la arquitectura planteada. Se pretende diseñar esta plataforma de tal manera que, seleccionando un determinado sensor y adecuando la configuración de cada uno de los nodos, se pueda obtener una solución concreta para una aplicación específica a partir de la plataforma genérica inicial.

En los siguientes capítulos se abordará el estudio de cada una de las tecnologías de comunicación integradas en la arquitectura propuesta, así como la codificación de cada tipo de nodo y la configuración de la plataforma *The Things Network*. Finalmente se plantea un *use of case* en el que se configura la plataforma genérica para una aplicación concreta con el fin de validar su adaptabilidad.

Capítulo 3. VLC

3.1 *Visible Light Communication*: Descripción

Ante la saturación del espectro de radiofrecuencia, y la necesidad de liberar y descongestionar las bandas más recurridas, van apareciendo y consolidándose nuevas tecnologías de comunicación como la que se trata en este punto del documento.

Durante el desarrollo de los Juegos Olímpicos de Londres 2012, se produjo por primera vez la posibilidad real de que la red de comunicaciones de la capital británica colapsara por un exceso de demanda. Con el fin de garantizar el correcto funcionamiento de la comunicación de audio, vídeo, y hasta de los sensores utilizados en el desarrollo de la competición, OFCOM (regulador de las comunicaciones británicas) tuvo que recurrir a parte del espectro destinado para uso militar para salvar la situación.

Por casos como este, la aparición de tecnologías de comunicación que no saturen más el espectro de radiofrecuencia son una buena opción para las nuevas comunicaciones requeridas, o para renovar las antiguas. Como una buena solución a este problema de la saturación radioeléctrica surge la denominada *Visible Light Communication* (*VLC*).

3.1.1 Definición

Visible Light Communication (*VLC*) es una tecnología de comunicación óptica inalámbrica que hace uso del espectro visible (400 THz – 789 THz) para sus comunicaciones. En

comunicaciones ópticas se suele trabajar habitualmente en términos de longitud de onda, quedando el espectro visible acotado por las longitudes de onda de 380 nm (violeta) y de 750 nm (rojo). En la Figura 3.1 [42] se puede ver la ubicación del espectro visible dentro del espectro electromagnético completo.

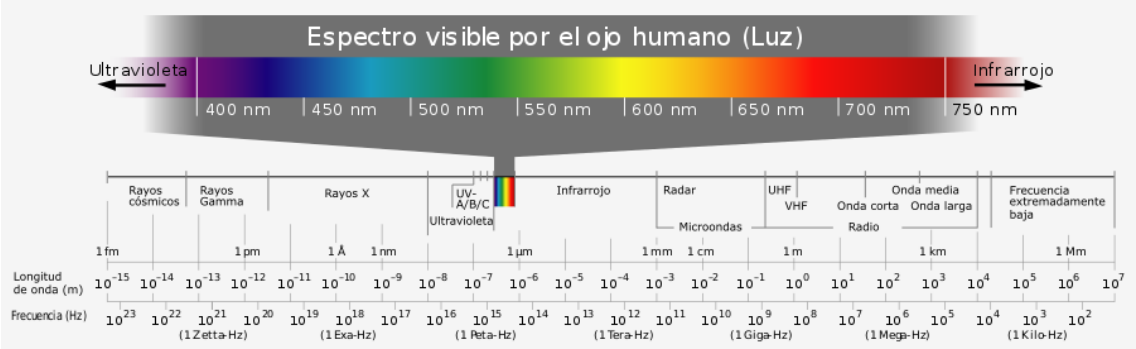


Figura 3.1: Ubicación del espectro visible en el espectro electromagnético.

En la Figura 3.2 se muestra un diagrama de bloques genérico de un sistema de comunicación *VLC*. En el transmisor (bloques azules de la Figura 3.2) se modulan los datos y se transmiten normalmente a través de diodos LED. Por su parte, en la recepción (bloques verdes de la Figura 3.2) se utilizan fotodetectores para recibir la señal de luz visible y posteriormente se demodulan los datos.

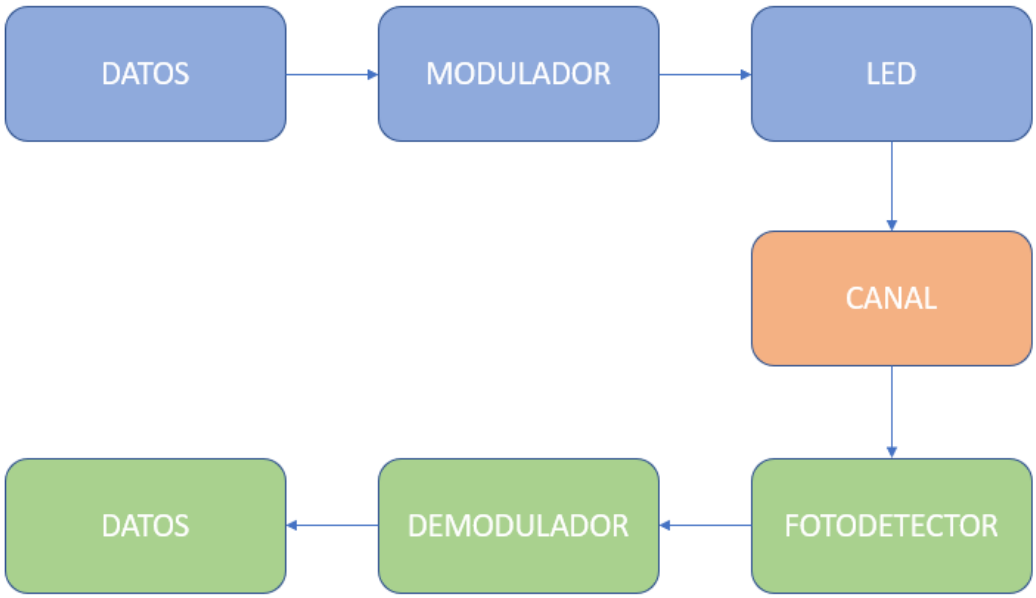


Figura 3.2: Diagrama de bloques genérico de una comunicación VLC.

Generalmente en el transmisor se utiliza tecnología LED, debido fundamentalmente a la alta frecuencia de conmutación que posee, así como al bajo consumo de potencia frente a la luminosidad que ofrece. Además, se trata de una tecnología con una vida útil mucho mayor que la de otros sistemas de iluminación. El consumo de potencia es un parámetro importante, ya que estos sistemas suelen cumplir una doble funcionalidad: iluminación y transmisión de datos. Con respecto a los receptores, en *VLC* se suele hacer uso de fotodiodos o fototransistores de bajo coste. En la Tabla 3.1 [43], [44], se muestra una tabla comparativa entre diferentes sistemas de iluminación. Se puede comprobar cómo el consumo de potencia para el mismo número de lúmenes (unidad del sistema internacional para el flujo luminoso) es muy inferior en la tecnología LED. Y, por lo tanto, si se plantea este parámetro a la inversa, la iluminación LED cuenta con más lúmenes por vatio que el resto de las tecnologías de iluminación. Como se comentó anteriormente, la vida útil de la iluminación LED es notablemente superior a la del resto de tecnología de iluminación. En cuanto a la capacidad de transmisión de datos, los sistemas LED son los únicos capaces de conmutar lo suficientemente rápido como para no ser detectados por el ojo humano y hacer la comunicación imperceptible, evitando efectos molestos y no deseados para los usuarios.

Parámetro	Incandescente	Halógeno	Fluorescente	LED	
Potencia (W) para el mismo número de lúmenes (lm)	25 W	20 W	5-7 W	2-3 W	200-280 lm
	40 W	30 W	7-12 W	3-5 W	450-520 lm
	60 W	50 W	12-15 W	6-9 W	500-620 lm
	90 W	75 W	18-23 W	9-11 W	900-1000 lm
Lumen/W	13-15	20-25	50-70	90-130	
Vida útil	1000 h	3000 h	7000 h	50000 h	

Tabla 3.1: Comparación de diferentes tecnologías de iluminación.

Con respecto a la tecnología utilizada en los receptores de los enlaces *VLC*, existen diferentes opciones que se estudian y analizan en la Tabla 3.2 [45]. Dependiendo de la

aplicación y de la respuesta que se requiera en cada caso de uso se puede optar por un tipo de fotorreceptor u otro.

Parámetro	Fotomultiplicador	Fotodiodo	Fototransistor	Célula fotovoltaica	Fotorresistor	Fotosensor MSM
Velocidades	Muy alta	Muy alta	Alta	Baja	Baja	Muy alta
Longitud de onda	0.2-0.4	0.2-2.0	0.4-1.1	0.4-0.7	0.4-0.7	0.4-0.8
Sensibilidad	Excelente	Muy Alta	Alta	Alta	Baja	Alta
Rango dinámico	Bueno	Excelente	Muy bueno	Bueno	Pobre	Muy bueno
Estabilidad	Muy buena	Muy buena	Buena	Pobre	Buena	Muy buena
Coste	Alto	Bajo	Muy bajo	Muy bajo	Bajo	Medio
Robustez	Pobre	Alta	Excelente	Excelente	Excelente	Excelente
Tamaño	Grande	Pequeño	Pequeño	Pequeño	Pequeño	Medio

Tabla 3.2: Comparación de diferentes tecnologías de sensorización lumínica.

Una vez presentadas las diferentes opciones tecnológicas para el transmisor o el receptor, se puede comenzar con el análisis de los diferentes modos de transmisión que permite realizar *VLC*. Dependiendo de diferentes parámetros como las posiciones y las orientaciones del emisor y del receptor, así como de los obstáculos que existan entre ellos, se pueden clasificar los modos de transmisión en cuatro tipos.

Cuando el emisor y el receptor están orientados y alineados, y, además, no hay obstáculos entre ellos, se consigue una línea de visión directa (*Line of Sight, LOS*). Este tipo de modo de transmisión se da en comunicaciones punto a punto en distancias largas, con diodos láser y al tratarse de largas distancias pues se da en exteriores. Este modo, al utilizar un haz de luz concentrado permite obtener altas tasas de transmisión. Otra de las ventajas que ofrece este modo de transmisión es que se trata del modo con menos pérdidas debido a su naturaleza. Esta misma naturaleza es la que hace que el sistema sea inmóvil y limite su

uso para aplicaciones concretas con emisores y receptores muy limitados en movimiento. Es un modo de transmisión punto a punto. En la Figura 3.3 [8], en la parte izquierda, se visualiza un ejemplo de este modo de transmisión dirigido con línea de visión.

Cuando el emisor y el receptor tienen una línea de visión, pero no están orientados entre sí, se cataloga como un modo no dirigido con línea de visión. El emisor emite su haz de luz por todo el área de cobertura y el receptor debe tener un ángulo de visión amplio [45]. Es un modo de transmisión que suele usarse en comunicaciones punto a multipunto, o *broadcast*. Este modo de transmisión dota de una mayor movilidad al sistema de comunicación *VLC*, ya que, si se fija el emisor en un punto determinado, múltiples receptores pueden moverse dentro del área de cobertura de este y obtener conectividad. Este modo de transmisión puede conllevar problemas de propagación multicamino, y comparado con el modo de transmisión dirigido con línea de visión, es menos eficiente y la velocidad de transmisión también es inferior. Este modo suele emplearse en los sistemas *VLC* que reutilizan el sistema de iluminación. En la parte central de la Figura 3.3, se muestra un ejemplo de este modo de transmisión no dirigido con línea de visión.

Cuando no existe línea de visión directa entre el emisor y el receptor, se tiene un modo de transmisión difuso. En este modo se hace uso de la reflexión del haz de luz emitido por el transmisor con el fin de evitar interrupciones de la línea de visión y obtener mayor movilidad desde el receptor o receptores. El inconveniente de la propagación multicamino se agrava en este modo de transmisión, así como la atenuación, la eficiencia o la velocidad de transmisión. Es un modo de transmisión que requiere de una mayor potencia en la emisión. Este es un modo que no se suele emplear para comunicaciones *VLC* sino más bien para sistemas de comunicación por infrarrojos. En la parte derecha de la Figura 3.3, se muestra un ejemplo de este modo de transmisión difuso.

Existe una cuarta opción, que más bien es una modificación, del modo de transmisión dirigido con línea de visión. Esta modificación se basa en permitir la movilidad del receptor realizando un seguimiento lumínico desde el emisor. De esta forma, el emisor y el receptor se mantienen alineados y orientados, que era la condición necesaria para tratar el modo de transmisión como dirigido. Las ventajas de este nuevo modo son iguales que las que tenía el primer modo explicado, pero añadiendo el hecho de lograr una mayor movilidad

en el receptor. El inconveniente ahora aparece a la hora de justificar si el coste y la dificultad del diseño son asumibles para las aplicaciones que se deseen desarrollar.

Figura 3.3: Modos de transmisión.

3.1.2 Capa física: Modulación

En cuanto a la modulación y a la demodulación, existen modulaciones predominantes en los sistemas *VLC*. Las modulaciones más usadas se agrupan en tres bloques: *Single Carrier Modulation* (SCM), *Multi Carrier Modulation* (MCM) y *Color Shift Keying* (CSK).

En las modulaciones de una única portadora se incluyen las modulaciones *On-Off Keying* (OOK), *Pulse Position Modulation* (PPM), *Differencial Pulse Position Modulation* (D-PPM), *Inverse Pulse Position Modulation* (I-PPM), *Variable Pulse Position Modulation* (V-PPM) y *Pulse Width Modulation* (PWM). En modulaciones de múltiples portadoras destaca la modulación *Orthogonal Frecuency-Division Modulation* (OFDM). Por último, se tiene la alternativa a estos dos bloques anteriores con un tipo de modulación diferente basado en el color, como *Color Shift Keying* (CSK).

OOK es posiblemente la modulación más simple en cuanto a la facilidad para su implementación. Se basa en el encendido y apagado de la fuente lumínica. Generalmente, el “0” representa su estado de apagado y el “1” representa su estado de encendido. En el estado de encendido, se transmite un pulso de luz y en el estado de apagado disminuye la intensidad de la luz, pero sin llegar a apagarlo completamente. Se suele complementar esta modulación con códigos como el código Manchester con el fin de evitar situaciones de parpadeo ante la aparición de muchos “0” o “1” seguidos.

La principal ventaja de esta modulación es su bajo coste y simplicidad, pero como desventaja presenta una alta sensibilidad a interferencias. En la Figura 3.4 [46] se muestra un ejemplo de la señal usando esta modulación, así como el código Manchester.

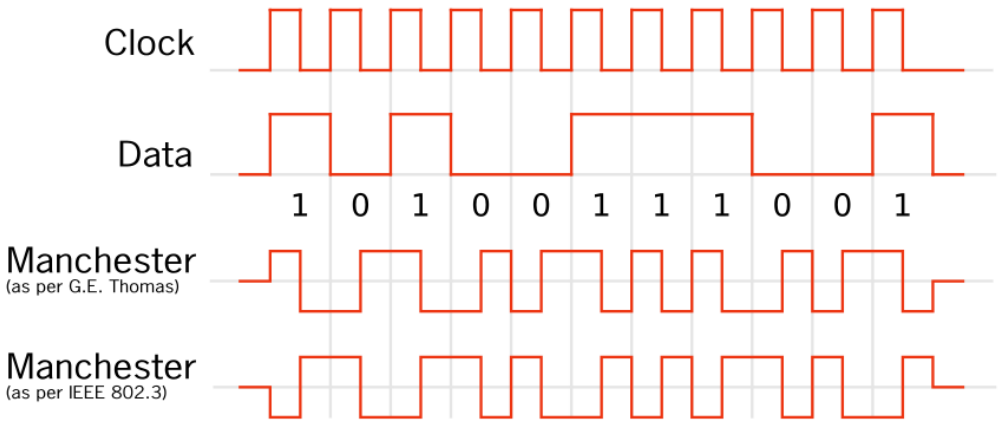


Figura 3.4: Modulación OOK con código Manchester.

PPM o modulación por posición de pulso codifica M bits con el fin de transmitir un pulso de duración $T/2^M$ en una de las 2^M posibles posiciones dentro del intervalo de T segundos, siendo T la duración del símbolo. En función del valor de M, se pueden encontrar diferentes modulaciones del tipo 2^M-PPM. En la Figura 3.5 [45] se muestra un ejemplo de la modulación 4-PPM. En este ejemplo se puede ver cómo al usar 2 bits se obtienen 4 posibles posiciones, y cómo cada T segundos se muestra un símbolo.

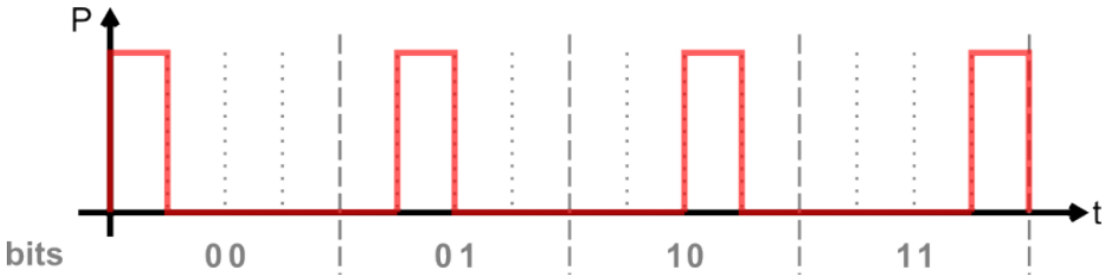


Figura 3.5: Modulación 4-PPM.

Existen diferentes variaciones de la modulación PPM, como D-PPM, I-PPM y V-PPM. En PPM se determina el símbolo en función de la posición del pulso respecto al reloj, por lo que requiere de una correcta sincronización. Cuando no es así, se producen errores que pueden evitarse midiendo el tiempo existente entre pulsos, codificando cada pulso en

relación con el anterior [45]. Esta variación se conoce como modulación por posición de pulso diferencial o D-PPM. En la Figura 3.6 [45] se muestra un ejemplo de esta modulación D-PPM. Otra alternativa que pretende mejorar la luminosidad que se logra con esta modulación es I-PPM o modulación por posición de pulso inversa. En esta modulación se invierte el estado bajo por el estado alto y viceversa. En la Figura 3.7 [45] se puede visualizar un ejemplo de esta modulación inversa.

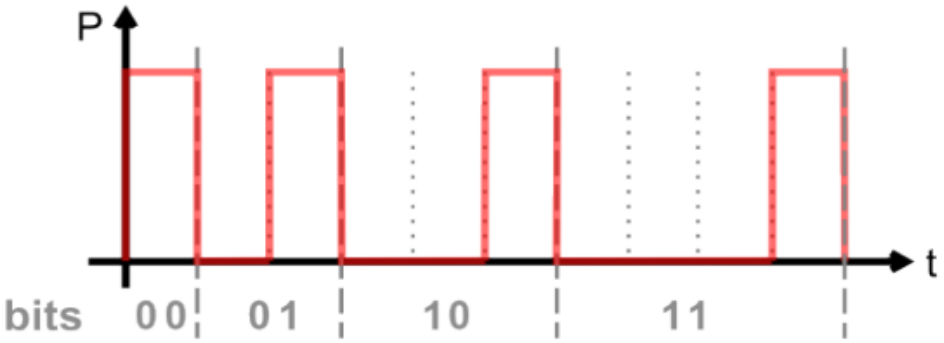


Figura 3.6: Modulación D-PPM.

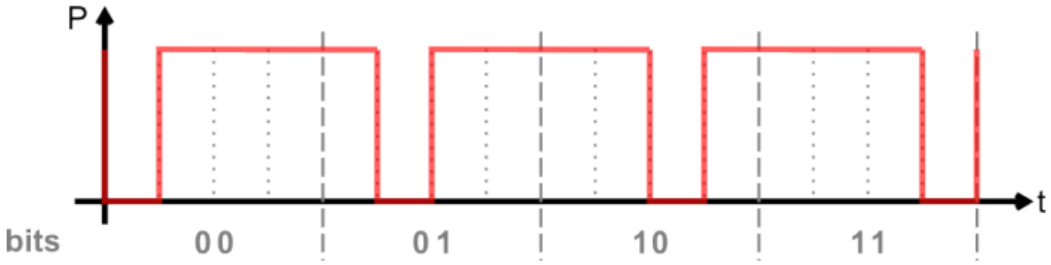


Figura 3.7: Modulación I-PPM.

Antes de abordar la modulación V-PPM se debe comentar la modulación de ancho de pulso o PWM. En esta modulación, la posición y la amplitud de los pulsos son constantes, pero lo que varía y contiene la información es el ancho de cada pulso. Es decir, el ancho del pulso es proporcional a la amplitud de la señal analógica. La ventaja que ofrece PWM es que el LED del transmisor puede parpadear a una frecuencia indetectable para el ojo humano, además de escalonar estos parpadeos en función del nivel de luminosidad deseado [8]. Pero, por otra parte, PWM es difícil de implementar ya que requiere de microprocesadores y de osciladores para su implementación. Por estos inconvenientes junto con los que ya presentaba la modulación PPM, se desarrolló V-PPM, que es el resultado de una combinación entre PPM y PWM.

En V-PPM el mensaje se codifica mediante PPM y el nivel de iluminación se controla mediante el ciclo de trabajo de la modulación PWM. De esta forma, se logra mejorar el control de la atenuación, la potencia media por bit es constante y no se producen parpadeos o *flicker*. En la Figura 3.8 [45] se muestra un ejemplo de esta modulación, en la que se codifica el mensaje a través de 2-PPM (1 bit), y PWM controla el nivel de iluminación.

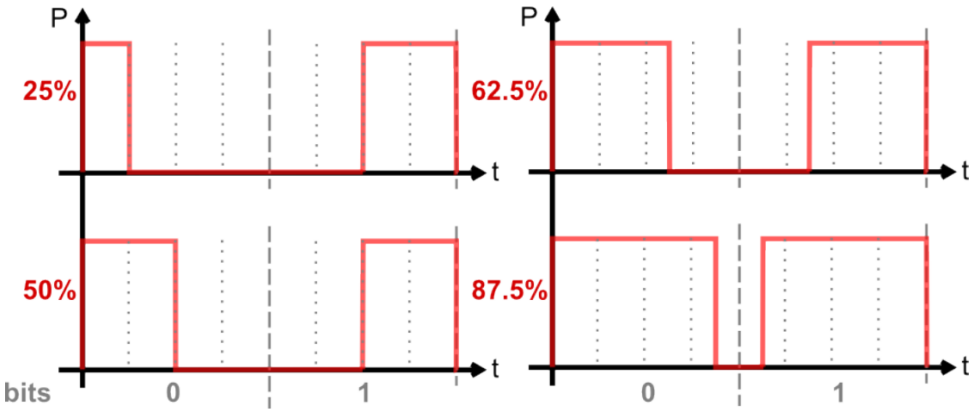


Figura 3.8: Modulación V-PPM.

En cuanto a las modulaciones de múltiple portadora, la más empleada es OFDM (*Orthogonal Frecuency Division Multiplexing*). OFDM es una modulación robusta ante la interferencia entre símbolos y que cuenta con una eficiencia espectral elevada. En definitiva, es una modulación ideada para sistemas *VLC* que requieran de grandes prestaciones en términos de velocidad, robustez y eficiencia espectral.

Una modulación diferente a las anteriores y que no se puede catalogar según el número de portadoras usadas, es *Color Shift Keying* (CSK). Esta modulación, como su nombre indica, se basa en el desplazamiento de color, y es una modulación ideada para usarla con LED de tipo RGB, ya que pueden iluminar con cualquier color que se pueda obtener a partir de los colores rojo, verde y azul. Según el estándar, se usan constelaciones de 4, 8 o 16 símbolos, donde cada uno representa un valor cromático distinto. El color percibido se compara con las coordenadas de color que se encuentran en el espacio de color CIE 1931. En la Figura 3.9 se adjunta este espacio de color CIE 1931.

CSK tiene la restricción de que la combinación de colores correspondientes a los diferentes puntos de la constelación debe producir un color objetivo que será el color de la lámpara que percibe el usuario. Para mejorar este inconveniente se introdujeron otro tipo de

modulaciones *Color-Space-Based Modulation* (CSBM), como son las modulaciones de color generalizadas o *Generalized Color Modulation* (GCM), que independizan la estructura de los puntos de la constelación de la modulación de las variaciones en el color y la intensidad de la luz de la lámpara. Por lo tanto, se puede lograr una comunicación VLC manteniendo el color y el brillo originales [47][48].

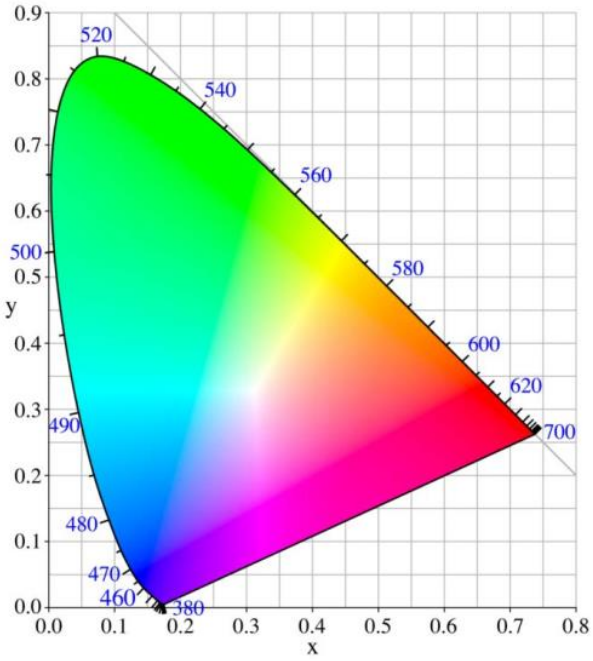


Figura 3.9: Espacio de color CIE 1931.

3.1.3 Características técnicas y estándar

En función del área o entorno de utilización del sistema *VLC*, su diseño e implementación debe regirse por el estándar o no. Existen dos posibilidades:

- Aplicación en un entorno específico, de carácter privado, de localización definida y que trabaja con equipos propios [49]. En este caso no es necesario seguir el estándar, puesto que al tratarse de una tecnología que no genera interferencias más allá del habitáculo en el que se encuentre el sistema, no hay posibilidad de afectar a sistemas ajenos o públicos. En este primer caso, la ventaja es que el diseño es más sencillo, ya que no se cuenta con ninguna restricción. No obstante, la no utilización del estándar genera que no se pueda garantizar la compatibilidad con otros sistemas *VLC*.

- Aplicación en un entorno público, con convivencia con otros estándares de comunicación. En este caso se debe garantizar la compatibilidad entre sistemas de comunicación y por lo tanto se debe seguir el estándar. En este sentido, el estándar internacional más seguido es el del IEEE 802.15.7.

En el estándar IEEE 802.15.7, dedicado exclusivamente a comunicaciones por *VLC*, se definen dos capas fundamentales, la capa física (PHY) y la capa de acceso (MAC).

Por una parte, la capa física se encarga de las especificaciones eléctricas y físicas, es decir, define la relación entre un dispositivo y el medio físico. Como en cualquier otro sistema de comunicación, el dispositivo transmisor envía datos al medio y el dispositivo receptor recoge datos del medio, pero la forma en la que esta se define es diferente según el modelo de capa física que se utilice. Existen tres modelos: PHY I, PHY II y PHY III.

Por otra parte, la capa MAC o de acceso al medio se encarga del direccionamiento y al control de acceso al medio. Esta capa, superior a la capa física en cuanto a nivel de abstracción, permite proporcionar comunicación *unicast*, *multicast* o *broadcast*. [49]

A continuación, se explicarán cada uno de los modelos de la capa física que se pueden utilizar según el estándar de *VLC*, IEEE 802.15.7.

- PHY I:

  Este modelo se utiliza mayoritariamente en entornos exteriores y para aplicaciones que no requieran de altas tasas binarias. Como mínimo se debe tener una tasa de 11.67 kb/s para una frecuencia de reloj óptico de 200 kHz. En la Tabla 3.3 [50] se recogen los distintos modos de operación de la capa física PHY I. El valor de la tasa de datos que se muestra en cada fila de la Tabla 3.3 depende también del tipo de código utilizado en la detección y corrección de errores. Normalmente, tanto el modelo PHY I como el modelo PHY II, son de tipo SISO (*Single Input Single Output*), es decir, PHY I cuenta con una única fuente de iluminación.

Modulación	Código	Tasa de reloj óptico	Tasa de datos
OOK	Manchester	200 kHz	11.67 kb/s – 73.3 kb/s
VPPM	4B6B	400 kHz	35.56 kb/s – 266.6 kb/s

Tabla 3.3: Modos de operación PHY I.

- PHY II:

Este modelo se utiliza mayoritariamente en entornos internos y para aplicaciones en las que se precisa una tasa de datos media, superiores a las del modelo PHY I. Como mínimo se debe tener una tasa de 1.25 Mb/s para una frecuencia de reloj óptico de 3.75 MHz. En la Tabla 3.4 [50] se recogen los distintos modos de operación de la capa física PHY II. El valor de la tasa de datos que se muestra en cada fila de la Tabla 3.4 depende también del tipo de código utilizado en la detección y corrección de errores. Al igual que en el modelo PHY I, el modelo PHY II es normalmente de tipo SISO, y por lo tanto cuenta también con una única fuente de iluminación.

Modulación	Código	Tasa de reloj óptico	Tasa de datos
VPPM	4B6B	3.75 MHz	1.25 Mb/s – 2 Mb/s
		7.5 MHz	2.5 Mb/s – 5 Mb/s
		15 MHz	6 Mb/s – 9.6 Mb/s
OOK	8B10B	30 MHz	12 Mb/s – 24 Mb/s
		60 MHz	38.4 Mb/s – 48 Mb/s
		120 MHz	76.8 Mb/s – 96 Mb/s

Tabla 3.4: Modos de operación PHY II.

- PHY III:

  PHY III es un modelo diferente a los dos anteriores, ya que normalmente es de tipo MIMO (*Multiple Input Multiple Output*), por lo que se puede utilizar en aplicaciones con varias fuentes de luz y receptores ópticos. En este caso la modulación usada es CSK y sólo existen variaciones en el tamaño de la modulación. En la Tabla 3.5 [50] se puede observar la relación existente entre las modulaciones, el código de detección y corrección de errores utilizado, la tasa de reloj óptico y la tasa de datos obtenida.

Modulación	FEC	Tasa de reloj óptico	Tasa de datos
4-CSK	RS (64,32)	12 MHz	12 Mb/s
8-CSK	RS (64,32)		18 Mb/s
4-CSK	RS (64,32)	24 MHz	24 Mb/s
8-CSK	RS (64,32)		36 Mb/s
16-CSK	RS (64,32)		48 Mb/s
8-CSK	Ninguno		72 Mb/s
16-CSK	Ninguno		96 Mb/s

Tabla 3.5: Modos de operación PHY III.

3.1.4 Ventajas y desventajas

Con el fin de analizar las ventajas y desventajas de *VLC* se resumen a continuación algunas de sus características más importantes.

El rango de frecuencias utilizadas en comunicaciones por luz visible va de 400 THz a 789 THz, que es justamente dónde se encuentra la banda del espectro visible. Por realizar una comparación en cuanto a la amplitud de este espectro, se puede comparar con el espectro radioeléctrico definido por la ITU (*International Telecommunication Union*) desde 3 KHz a 300 GHz. Es decir, se puede afirmar que el espectro visible es aproximadamente, 1000 veces más amplio que el espectro radioeléctrico acotado por la ITU.

Por supuesto, como el nombre de esta tecnología indica, sus emisiones son visibles al ojo humano, por lo que en términos de salud pública puede llegar a afectar a la población. Existen ciertas modulaciones en las que se pueden producir parpadeos en el transmisor y afectar la salud de las personas y/o animales [8]. Un aspecto importante de *VLC* es que para lograr las mejores tasas de datos necesita mantener la fuente de luz encendida, y esto limita algunas aplicaciones. Aunque también se puede buscar un control adaptativo de la luminosidad, y por lo tanto de la comunicación *VLC*.

Se ha demostrado que si se utilizan técnicas de modulación digital de pulsos es posible alcanzar tasas máximas de transmisión del orden de 625 kbit/s de día, usando un 1% de la potencia máxima de la fuente de luz, que supone un 0.36% de la luz total; y una tasa máxima de 320 kbps usando 0.001% de la potencia máxima de la luminaria, que supone un 7.5% de la luz que hay durante la noche, suficiente para aplicaciones de información y señalización con sensores y actuadores. [49] [51]

La tecnología más empleada para *VLC* es la tecnología LED, que además se está imponiendo también como sistema de iluminación más eficiente. Es decir, si la tecnología LED sigue implantándose en la mayoría de los espacios, públicos y privados, las comunicaciones por luz visible tendrán cada vez más fácil su implementación, ya que, si se cuenta con el sistema de iluminación LED, sólo haría falta añadir la electrónica necesaria para realizar las modulaciones y se tendría un sistema *VLC*, o al menos el emisor de este sistema. En resumen, la reutilización de los sistemas de iluminación ya existentes y el uso de tecnología eficiente como la LED son dos características muy importantes de *VLC*.

Otra característica que se debe destacar es que el rango de frecuencias del espectro visible es de libre emisión y no requiere licencias. Esto dota al diseñador del sistema *VLC* de mayor autonomía y lo libera de restricciones temporales por la espera de la licencia. Además, la existencia de este espectro visible ayuda a la descongestión que existe en las bandas ISM (*Industrial Scientific and Medical*).

Además, la tecnología *VLC* cuenta con la ventaja de que el ancho de banda no se comparte entre los usuarios (receptores) que se conecten al transmisor simultáneamente. Y desde el punto de vista de la seguridad en las comunicaciones, cuando se usa *VLC* en interiores, las radiaciones son confinadas por las paredes, lo que genera un sistema de comunicación más

seguro ante interceptaciones o accesos no autorizados. En escenarios de interiores se puede reducir el nivel de codificación con el fin de aumentar la tasa binaria, sabiendo que es imposible que la luz proporcione cobertura fuera de una habitación. Mientras que en escenarios de exteriores si se debe codificar la información puesto que en un espacio abierto si se pueden interceptar comunicaciones.

El concepto de confinamiento de las radiaciones también es importante para la reutilización del espectro visible entre habitáculos. Por ejemplo, en una habitación se puede usar una determinada longitud de onda, y en la habitación de al lado, si existe una pared sin fisuras ni aperturas, se puede usar esa misma longitud de onda sin riesgo de producir interferencias.

Con todo esto, si se realiza un resumen de las ventajas y desventajas que aporta *VLC* se puede obtener una tabla como la de la Tabla 3.6.

Ventajas	**Desventajas**
Espectro amplio	Corto alcance (altas velocidades)
Espectro de libre emisión	Propagación multicamino en algunos modos de transmisión
Ancho de banda no compartido	Interferencias con luces
Seguridad en las comunicaciones en interiores	Reducida movilidad en algunos modos de transmisión
Reutilización de sistemas de iluminación	
Bajo consumo energético	
Reutilización del espectro en habitáculos cercanos	
Altas tasas de datos	
Convivencia total con sistemas RF	
Bajo coste (reutilización)	

Tabla 3.6: Ventajas y desventajas de la tecnología VLC.

3.2 Implementación de *VLC* en la plataforma HW/SW

En este apartado se describe a nivel físico la solución de la comunicación basada en *Visible Light Communication* (*VLC*) para la plataforma HW/SW propuesta en este TFM. La solución de comunicación *VLC* consiste en dos enlaces de comunicación. El primer enlace trata de comunicar un transmisor que se supone fijo en una posición determinada con un receptor que se encuentra en un nodo móvil. Realmente, se denomina nodo fijo a aquel nodo que posee alimentación ininterrumpida, y nodo móvil a aquel que no la posee. Es decir, que el nodo móvil puede estar mecánicamente fijo. Por su parte, el segundo enlace cumple la función inversa, es decir, el transmisor es el nodo móvil y el receptor se encuentra fijo en una posición determinada. A continuación, se estudian ambos enlaces.

3.2.1 Enlace 1: Transmisor en nodo estación base y receptor en nodo móvil

En este enlace el transmisor se puede conectar a la infraestructura eléctrica del inmueble en cuestión, por lo que se considera alimentado de forma ininterrumpida y por lo tanto no se cuenta con una restricción de consumo de potencia. Tanto el transmisor como el receptor deben ser alimentados por el dispositivo IoT seleccionado para esta plataforma, el dispositivo LoPy, o directamente desde la batería que alimenta a este. Es decir, la tensión de alimentación será en ambos casos de 5V. La comunicación entre los módulos *VLC* utilizados y los dispositivos LoPy es de tipo serie haciendo uso de las UART que ofrece LoPy entre sus recursos. El objetivo perseguido con este enlace es obtener comunicación por luz visible para una distancia aproximada de 50 cm. No obstante, el alcance que se puede lograr, como se verá en la presentación de los resultados experimentales de este capítulo, es mayor que esta distancia, pero se ha fijado este valor de alcance para realizar las pruebas y comparaciones entre diferentes opciones para el transmisor y el receptor. Además, el alcance aumenta o disminuye en función de la tecnología *VLC* utilizada, como al emplear, por ejemplo, lámparas de mayor intensidad. Por supuesto, el hecho de utilizar elementos de mayor intensidad conlleva un mayor consumo de potencia. Teniendo en cuenta estas ideas y restricciones a continuación se muestra el procedimiento de diseño de los dos elementos que forman este enlace.

3.2.1.1 Planteamiento inicial: Montaje 1

El planteamiento inicial para el circuito transmisor es el que se muestra en la Figura 3.10. En este esquemático se puede observar que se alimenta la circuitería con 5V. Para este montaje y su correspondiente prueba de funcionamiento, la alimentación procede de un equipo de fuente de alimentación, representado como VS en el circuito de la Figura 3.10. Este elemento pretende simular la alimentación que en el montaje final recibirá el circuito desde la batería integrada en el dispositivo LoPy. La comunicación entre este circuito y el dispositivo LoPy se realizará a través de UART, donde la configuración codificada será de 9600 baudios, 8 bits de datos, sin bit de paridad y 1 bit de stop. El *baud rate* seleccionado influye en la frecuencia de señal que se debe configurar en el generador de funciones VG1 que se muestra en el circuito de la Figura 3.10. No obstante, para este planteamiento inicial, la frecuencia configurada en el generador de funciones es de 1 KHz. La forma de la señal con la que se configure el generador de funciones debe ser cuadrada.

El circuito de la Figura 3.10 es el resultado del diseño, en una primera aproximación, del circuito transmisor para el enlace 1. Consta de un MOSFET IRF840 que permite la conmutación en el estado de la lámpara de 5V. También cuenta con las resistencias R1 y R2 según la notación del circuito de la Figura 3.10 que regulan la corriente en diferentes partes del circuito. La resistencia R1 controla la corriente que llega al MOSFET e inicialmente se fija a 10 KΩ. Este valor puede variar en las siguientes versiones del circuito dependiendo del estado de la señal que se genere en VG1. Por ejemplo, cuando la señal de entrada VG1 provenga del dispositivo LoPy puede que haya que reajustar el valor de esta resistencia.

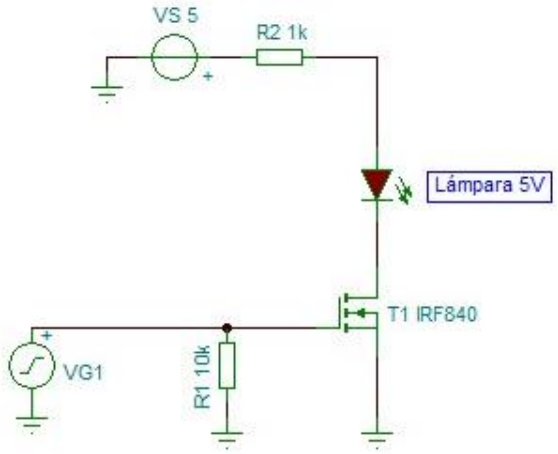


Figura 3.10: Circuito transmisor para el montaje 1.

Por otra parte, la resistencia R2 se encarga de controlar la corriente que circula por la lámpara de 5V. Con el fin de realizar un primer montaje se fija el valor de R2 a 1KΩ. Para este valor se espera que la lámpara brille con una intensidad baja pero que suficiente como para iluminarse. La lámpara de 5V utilizada es la que se muestra en la Figura 3.11.

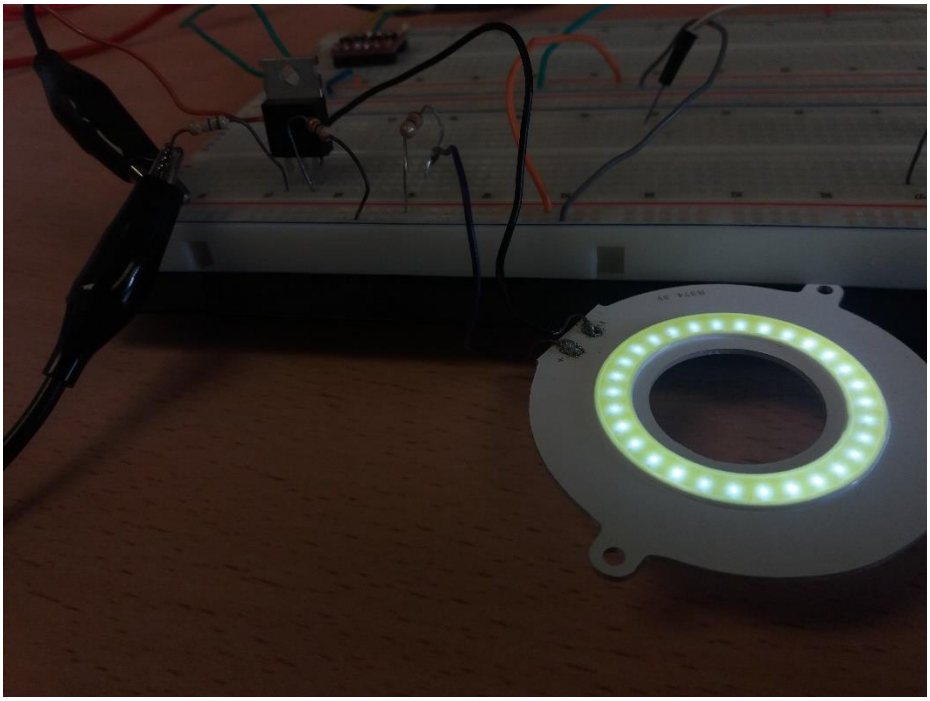


Figura 3.11: Lámpara de 5V conectada al circuito transmisor del montaje 1.

Por otra parte, la primera versión del receptor para este enlace es el que se muestra en la Figura 3.12. En este caso, el circuito se alimenta también con 5V desde la fuente de alimentación, y entre los fototransistores BPW40 y la resistencia de 10 KΩ, se mide la señal recibida con el osciloscopio. El funcionamiento de este circuito consiste en la utilización de elementos sensibles a la luz como son los fototransistores, los cuales conducen más o menos corriente de colector en función de la luz que incide sobre ellos. El valor de la resistencia R2 se establece en 10 KΩ para una primera prueba basándose en el estudio del estado del arte realizado sobre [8]. Posiblemente este valor tendrá que ser ajustado en función de los resultados obtenidos en las diferentes versiones de los montajes.

Los fototransistores utilizados cuentan con una lente que permite una mejor captación de la señal lumínica entrante pero que también hacen del módulo receptor un elemento muy directivo. Es decir, el módulo receptor debe estar en *Line of Sight* (LoS) con el transmisor para recibir correctamente la señal.

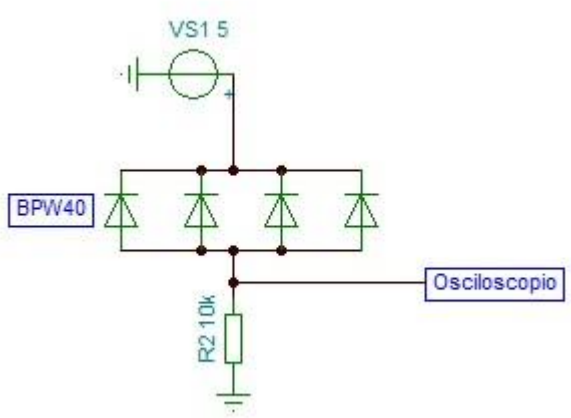


Figura 3.12: Circuito receptor para el montaje 1.

En la Tabla 3.7 se resumen las condiciones de este montaje y de la prueba experimental realizada.

Equipo	Configuración
Generador de funciones HAMEG HM8131-2	Frecuencia: 1 KHz Amplitud: 5 Vpp Offset: 2.5 V
Fuente de alimentación HAMEG HM8142	Tensión: 5 V Intensidad: 0.168 A
Osciloscopio HAMEG HM 407-2	TIME/DIV: 200 µs/div V/DIV: 1 V/div

Tabla 3.7: Condiciones de experimento para el montaje 1.

Las medidas recogidas sobre esta prueba se muestran en la Tabla 3.8, en la que se valora la amplitud de señal recibida en función de la distancia entre transmisor y receptor. Como se puede observar en la Tabla 3.8, el resultado es insuficiente, ya que como máximo sólo se recibía señal a una distancia de 10 cm, siendo, además, el nivel de señal recibido bajo. Aun así, de este primer montaje se concluye que existe comunicación correcta entre los dos circuitos (transmisor y receptor), aunque esta deba mejorarse.

Distancia	Amplitud de señal recibida
1 cm	0.8 Vpp
2 cm	0.5 Vpp
5 cm	0.28 Vpp
10 cm	0.2 Vpp

Tabla 3.8: Resultados del experimento del montaje 1.

La principal conclusión de este primer montaje fue que la luminosidad de la lámpara era muy baja debido al elevado valor de resistencia que hay entre la fuente de alimentación de 5V y la lámpara. El nivel de luminosidad de la lámpara alcanzado se muestra en la Figura 3.11. En el siguiente montaje se varía este valor con el fin de alcanzar una comunicación de mayor alcance.

3.2.1.2 Aumento del alcance: Montaje 2

En el montaje 1 se estableció comunicación por luz visible entre un transmisor y un receptor, recibiendo este último una señal de muy baja amplitud y de muy corto alcance. En este segundo montaje se variará el circuito transmisor con el fin de obtener una comunicación de mayor alcance.

El cambio realizado en el circuito transmisor consistió en disminuir el valor de la resistencia que se ubica entre la fuente de alimentación de 5V y la lámpara. En el circuito transmisor del montaje 1, esta resistencia era de 1 KΩ, y en este nuevo montaje se ha sustituido por una resistencia de valor inferior, en concreto, de 1.5Ω. El circuito receptor utilizado en este segundo montaje es el mismo que para el montaje 1. En la Figura 3.13 se muestra el nuevo circuito transmisor para el montaje 2, mientras que el circuito receptor es el que se mostró en el apartado anterior, en la Figura 3.12.

El montaje completo se puede visualizar en la Figura 3.14. Como se puede observar en esta imagen, la luminosidad de la lámpara de 5V ahora sí es bastante mayor que en el montaje 1.

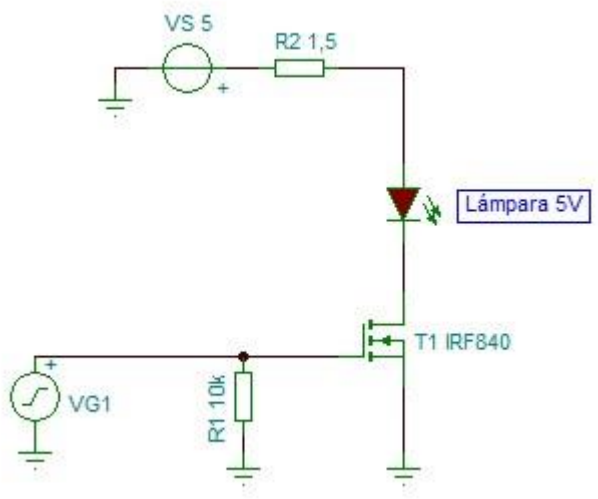


Figura 3.13: Circuito transmisor para el montaje 2.

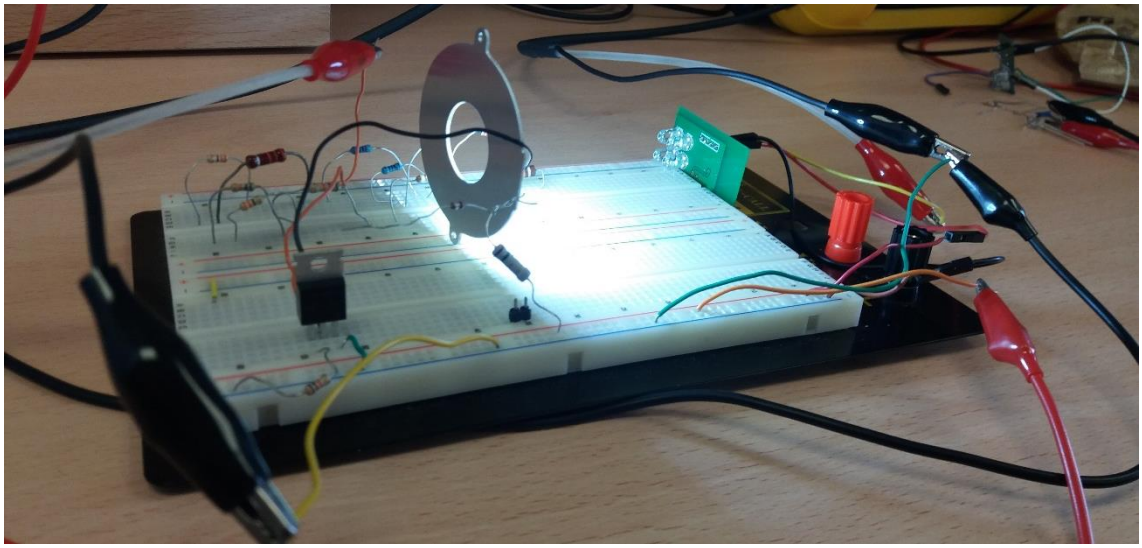


Figura 3.14: Segundo montaje completo.

En la Tabla 3.9 se resumen las condiciones de este montaje y de la prueba realizada.

Equipo	Configuración
Generador de funciones HAMEG HM8131-2	Frecuencia: 1 KHz Amplitud: 5 Vpp Offset: 2.5 V
Fuente de alimentación HAMEG HM8142	Tensión: 5 V

	Intensidad: 0.168 A
Osciloscopio HAMEG HM 407-2	TIME/DIV: 200 µs/div V/DIV: 1 V/div

Tabla 3.9: Condiciones de experimento para el montaje 2.

Los resultados obtenidos para este segundo montaje, en función de la distancia de comunicación, son los que se muestran en la Tabla 3.10.

Distancia	**Amplitud de señal recibida**
30 cm	2.9 Vpp
40 cm	2.5 Vpp
50 cm	2 Vpp
60 cm	1.4 Vpp
70 cm	1.2 Vpp
80 cm	1 Vpp
90 cm	0.8 Vpp
1 m	0.6 Vpp
1.1 m	0.5 Vpp
1.2 m	0.35 Vpp
1.3 m	0.35 Vpp
1.4 m	0.2 Vpp
1.5 m	0.2 Vpp

Tabla 3.10: Resultados del experimento del montaje 2.

Se debe comentar que para distancias inferiores a 25 o 30 cm, la señal recibida se encontraba saturada, por lo que la distancia mínima se fija en 30 cm para el montaje realizado. Para valores de distancia superiores a 1.5 m se recibe bien la señal, pero ya con

una amplitud igual o inferior a 100 mVpp. En la Figura 3.15 se muestra la variación de la amplitud de la señal recibida en función de la distancia de forma gráfica.

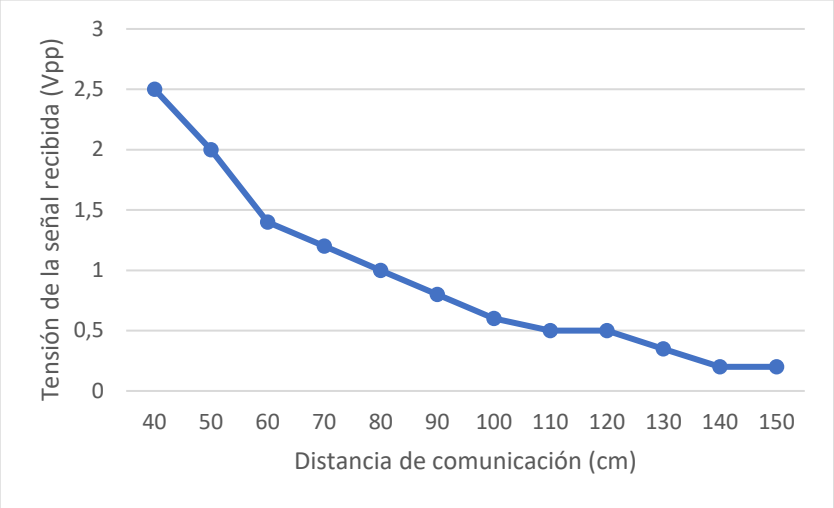


Figura 3.15: Relación distancia - tensión recibida para el montaje 2.

En la Figura 3.16 se muestra una foto de la señal recibida vista en el osciloscopio.

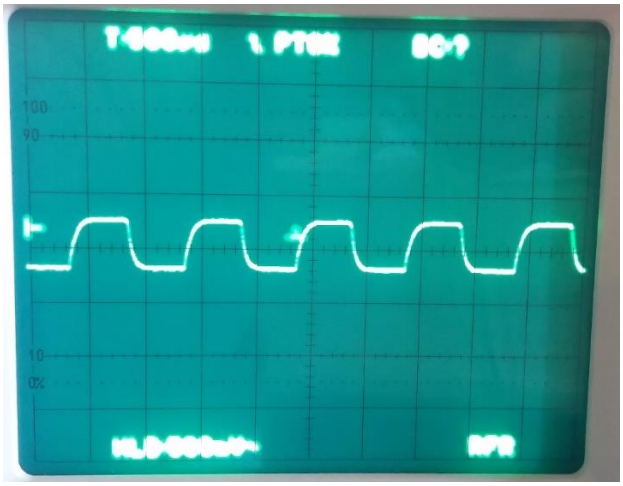


Figura 3.16: Señal recibida para una distancia de 1m con el montaje 2.

Como primera conclusión de este segundo montaje se obtiene que en este caso la señal transmitida es mucho más potente que en el caso 1, de tal forma que, sin realizar modificaciones en el circuito receptor, este es capaz de recibir correctamente señales con más de 1 metro de distancia entre ambos circuitos.

Como prueba de concepto se realizó una comparativa con una lámpara de mayor intensidad lumínica y una alimentación requerida de 12 V. El alcance obtenido al sustituir

la lámpara de 5 V por la lámpara de 12 V aumenta considerablemente, pudiendo llegar a cubrir comunicaciones de más 2.5 metros de distancia. En la Figura 3.17 se muestra el enlace a la distancia de comunicación de 2.5 metros. Con esta prueba se confirma que utilizando elementos con mayor potencia lumínica se logran mejores prestaciones en cuanto a alcance, pero a su vez empeora las prestaciones en términos de consumo de potencia. Debido a la naturaleza de este TFM se prioriza el consumo de potencia sobre el alcance, por lo que se sigue desarrollando el circuito transmisor para una lámpara de 5 V. En la Tabla 3.11 se muestra una comparativa del consumo de potencia y el alcance obtenido para cada una de las dos lámparas estudiadas.

Figura 3.17: Enlace de 2.5 m usando lámpara de 12 V.

Tipo lámpara	Alcance	Corriente pico (osciloscopio)
Lámpara 5V	1.5 m	58,33 mA
Lámpara 12V	>2.5 m	70,83 mA

Tabla 3.11: Comparativa de prestaciones entre lámparas.

No obstante, el buen comportamiento de la señal sigue siendo para una frecuencia de 1 KHz. Por lo que se debe realizar una prueba de variación de frecuencia con el fin de valorar si se alcanza una comunicación para frecuencias mayores. La razón de ser de esta búsqueda de incremento en la frecuencia se fundamenta en que, para la comunicación con el dispositivo LoPy, la configuración de la UART desea establecerse a 9600 baudios, es decir,

se debe alcanzar una comunicación correcta cercana a los 10 KHz para garantizar el funcionamiento del circuito cuando este se integre al dispositivo LoPy.

En la Tabla 3.12 se muestra el resultado para el montaje 2 variando la frecuencia de la señal transmitida desde el generador de funciones. La distancia para las diferentes frecuencias es de 1 metro.

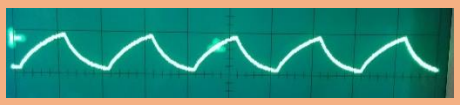
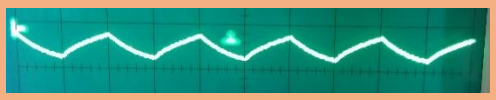
Frecuencia (KHz)	Distancia (m)	Vpp (V)	Señal
1	1	0.6	
5	1	0.5	
10	1	0.3	
15	1	0.25	

Tabla 3.12: Variación de la señal analógica en función de la frecuencia de la señal.

En la Tabla 3.12 se puede ver cómo la forma de la señal recibida se deteriora según se aumenta la frecuencia. En definitiva, se debe adaptar el circuito receptor con el fin de mantener una comunicación estable para una frecuencia de 10 KHz.

Antes de proceder al estudio del circuito que permite mantener una comunicación a 10 KHz, se digitalizó la señal de salida del receptor incluyendo en este circuito un comparador. Es decir, se planteó un nuevo modelo de receptor que permita obtener una señal digital de salida de aproximadamente 5 V a nivel alto. El circuito receptor queda como el que se muestra en la Figura 3.18. Se sigue usando el circuito con salida analógica de la Figura 3.12 pero esta salida se conecta ahora a la entrada de un operacional, que en su entrada no inversora cuenta con dos resistencias a modo de divisor de tensión.

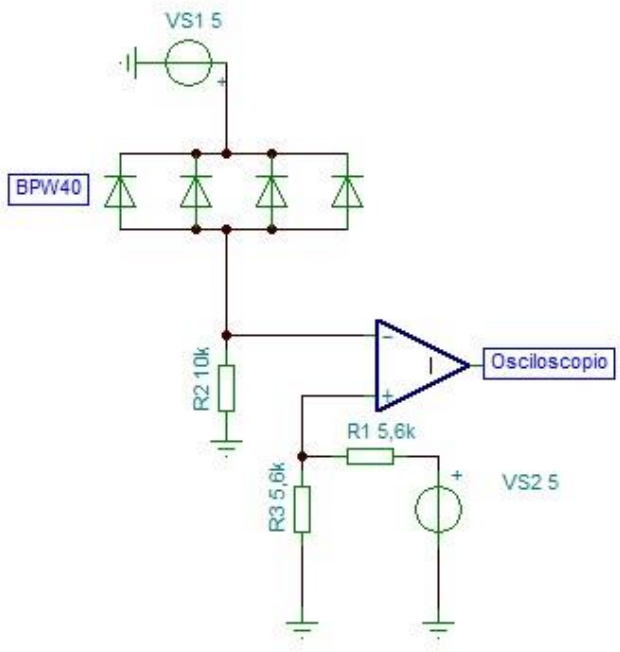


Figura 3.18: Circuito receptor de salida digital para el montaje 2.

Manteniendo el circuito transmisor que se describió en la Figura 3.13 y las condiciones de la Tabla 3.9 se obtienen los resultados mostrados en la Tabla 3.13, en función de la distancia entre transmisor y receptor.

Distancia	Amplitud de la señal recibida
<10 cm	Saturación
10-60 cm	5 Vpp (Señal digital)
>60 cm	Pérdida de LoS

Tabla 3.13: Resultados con el receptor de salida digital para el montaje 2.

Para distancias inferiores a 10 cm, la señal a la entrada del comparador se encuentra saturada y por lo tanto no se puede realizar la comparación de forma adecuada. En el intervalo de 10 cm a 60 cm de distancia se logra una comunicación muy estable y se obtiene una señal de salida como la de la Figura 3.19. Para distancias superiores a 60 cm se puede conseguir recoger la señal, pero el distanciamiento entre transmisor y receptor hace que se pierda el apuntamiento. Se recuerda que los fototransistores cuentan con una lente que

convierte al módulo receptor en muy directivo, de tal forma que desde que se pierde LoS es difícil mantener una comunicación estable.

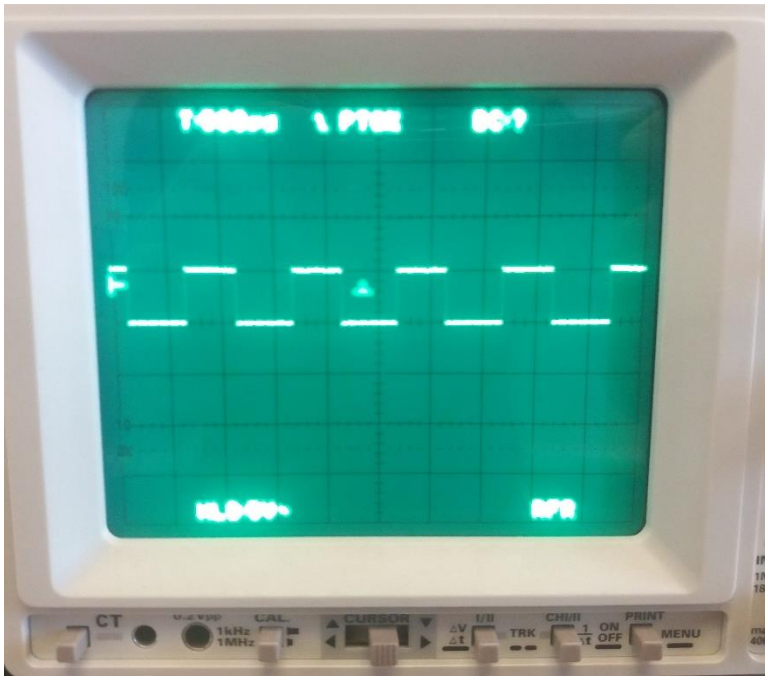
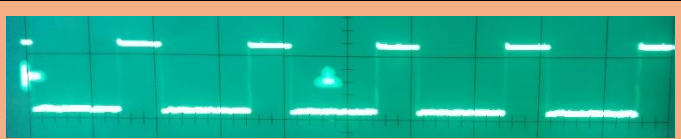


Figura 3.19: Señal de salida del módulo digital para el montaje 2.

Al igual que con el módulo de salida analógica, se varió la frecuencia con el objetivo de conocer la respuesta de este para frecuencias superiores. El resultado de esta variación se recoge en la Tabla 3.14. En ella se puede ver cómo al aumentar la frecuencia, el ciclo de trabajo disminuye. Este efecto se debe a que, al aumentar la frecuencia, la señal analógica que entra al comparador supera el valor umbral menos veces al disminuir su amplitud y variar su forma, como se pudo ver en la Tabla 3.12. Esto hace que se perciba que disminuye el ciclo de trabajo de la señal resultante.

Frecuencia (KHz)	Distancia (m)	Vpp (V)	Señal
1	0,6	5	
5	0,6	5	

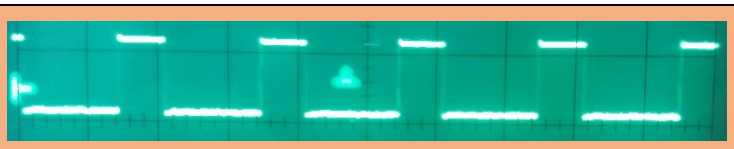
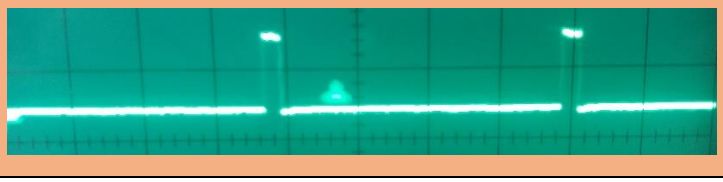
10	0,6	5	
12	0,6	5	

Tabla 3.14: Variación de la señal digital en función de la frecuencia de la señal.

Tanto por los resultados con el módulo de salida analógica como con el módulo de salida digital, se ha visto que es necesario modificar la circuitería de tal manera que la forma de la señal para una frecuencia de 10 KHz no se vea tan deteriorada como en estos casos.

3.2.1.3 Aumento de la frecuencia: Montaje 3

En este tercer montaje realizado en el desarrollo de este TFM el transmisor sigue manteniendo el circuito que se estudió en el apartado anterior, concretamente en la Figura 3.13. Sin embargo, el circuito receptor sí sufre modificaciones con respecto al apartado anterior con el fin de recibir correctamente señales de 10 KHz de frecuencia. El nuevo circuito receptor propuesto se muestra en la Figura 3.20. En este circuito se disminuye el valor de la resistencia que se encuentra en paralelo con los fototransistores, de 10 KΩ a 1 KΩ, se desoldaron las resistencias de 5.6 KΩ que formaban un divisor de tensión y se soldó un potenciómetro de 10 KΩ en su lugar.

En la Tabla 3.15 se muestran las condiciones seguidas para realizar las pruebas de funcionamiento del nuevo circuito receptor implementado.

Con el fin de ajustar el valor umbral con el que se realiza la comparación se ha ido variando el valor del potenciómetro. Para estas pruebas se ha utilizado una señal con una frecuencia de 10 KHz.

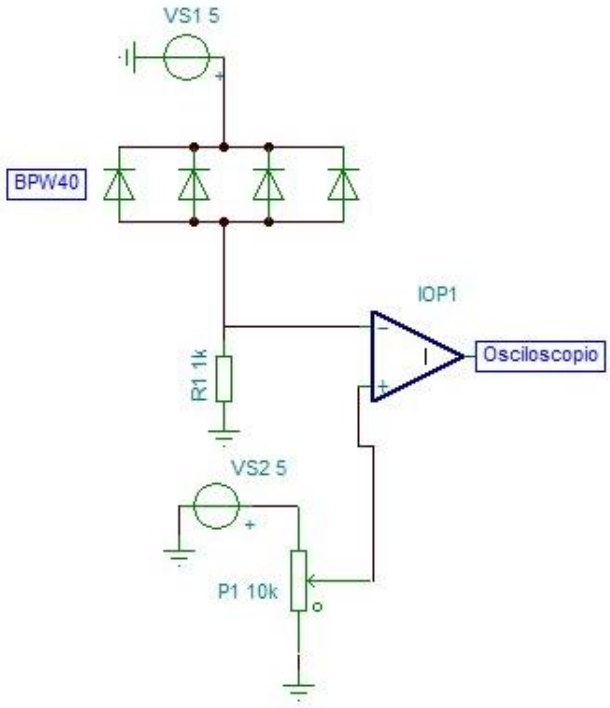


Figura 3.20: Circuito receptor para el montaje 3.

Equipo	Configuración
Generador de funciones HAMEG HM8131-2	Frecuencia: 10 KHz Amplitud: 5 Vpp Offset: 2.5 V
Fuente de alimentación HAMEG HM8142	Tensión: 5 V Intensidad: 0.168 A
Osciloscopio HAMEG HM 407-2	TIME/DIV: 200 µs/div V/DIV: 5 V/div

Tabla 3.15: Condiciones de experimento para el montaje 3.

En estas pruebas se ha fijado la distancia de la comunicación a la que se había establecido como objetivo, 50 cm. Para esta distancia, y con una transmisión a 10 KHz, se han obtenido los siguientes resultados en función del valor del potenciómetro.

En todas las imágenes resultantes que se muestran como resultado, se pueden observar la señal digital y la señal analógica, en el canal 1 y en el canal 2, respectivamente. En la Figura 3.21 se muestra el resultado obtenido para un valor en el potenciómetro de 1.5 KΩ (valor umbral de comparación de 0.75 V). En este caso, para la Figura 3.21, el canal 1 se encuentra en 5 V/div, y el canal 2 en 10 mV/div.

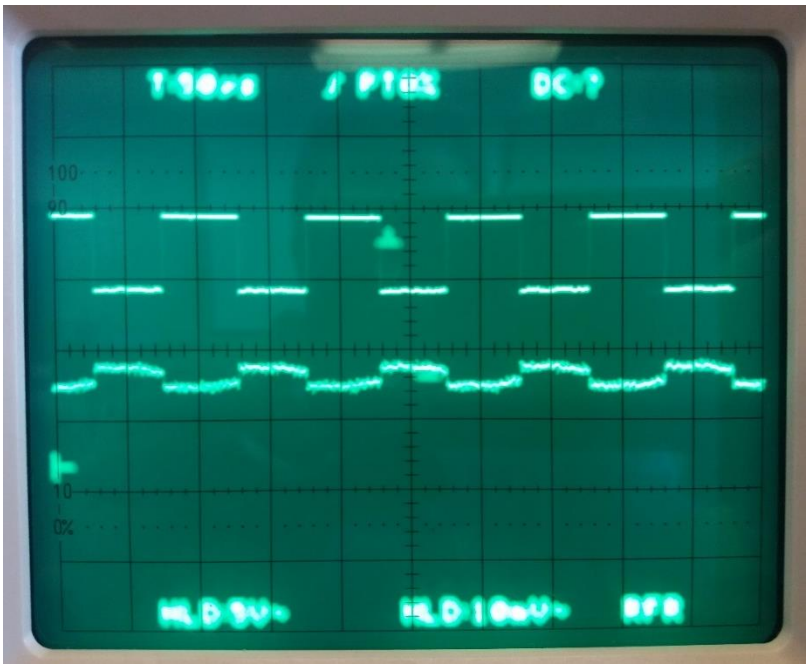


Figura 3.21: Señal digital y señal analógica para valor de potenciómetro de 1.5 KΩ.

En la Figura 3.22 se muestra el resultado obtenido para un valor en el potenciómetro de 2 KΩ (valor umbral de comparación de 1 V). En este caso, para la Figura 3.22 el canal 1 se encuentra en 5V/div, y el canal 2 en 5 mV/div.

En la Figura 3.23 se muestra el resultado obtenido para un valor en el potenciómetro de 1 KΩ (valor umbral de comparación de 0.5 V). En este caso, para la Figura 3.23 el canal 1 se encuentra en 5V/div, y el canal 2 en 5 mV/div.

A la vista de los resultados, se puede observar que la amplitud de la señal analógica en los tres casos es inferior a la obtenida con el circuito receptor correspondiente al montaje 2 del apartado anterior. Esto se debe a la disminución del valor de la resistencia que se encuentra en paralelo con los fototransistores. No obstante, se logra una señal que diferencia bien el nivel alto del bajo, y que permite realizar una comparación aceptable. El

resultado se considera satisfactorio, ya que para un valor de frecuencia de 10 KHz se logra recibir la señal a la salida del comparador en buen estado.

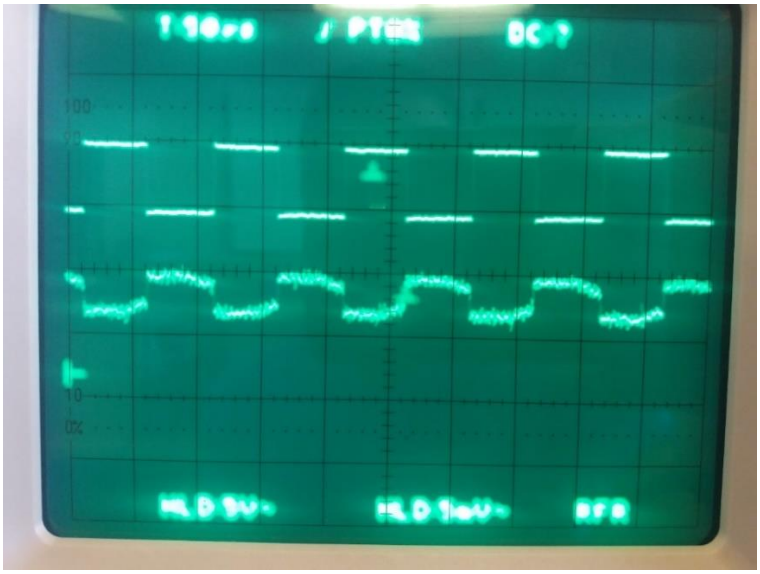


Figura 3.22: Señal digital y señal analógica para valor de potenciómetro de 2 KΩ.

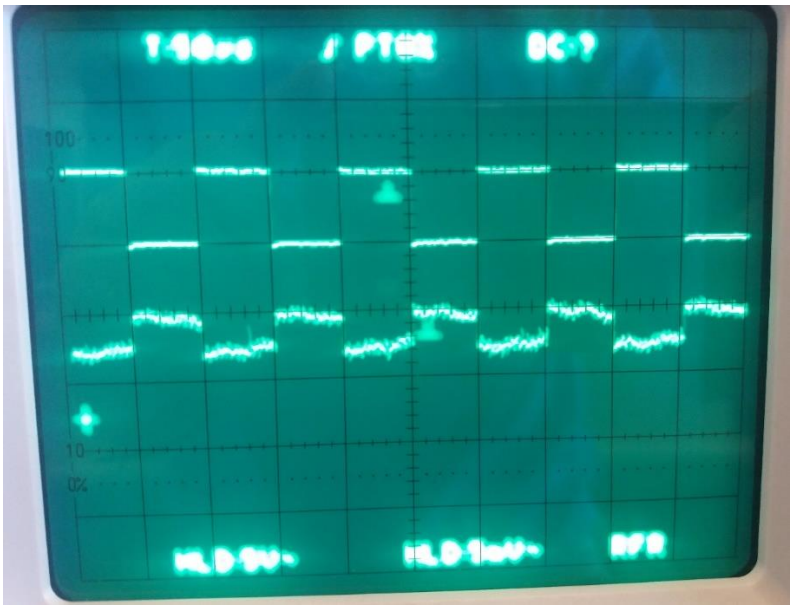


Figura 3.23: Señal digital y señal analógica para valor de potenciómetro de 1 KΩ.

En cuanto a los valores tomados por el potenciómetro, se debe comentar que los valores para los que se lograba una buena señal de salida oscilaban entre 600 Ω y 2.8 KΩ, donde los tres valores probados y documentados gráficamente son los más representativos dentro de ese rango posible. De los valores del potenciómetro 1.5 KΩ, 2 KΩ y 1 KΩ, la mejor señal obtenida por su forma y estabilidad es la lograda con el valor de 1 KΩ.

Se debe comentar también que al contar los fototransistores con una lente que concentra la señal recibida, el ángulo de incidencia de la luz sobre estos es un factor clave, por lo que se debe tener cuidado a la hora de orientar el receptor hacia el sistema emisor óptico.

3.2.1.4 Integración con dispositivo LoPy: Montaje 4

En este apartado se plantea la integración de los módulos transmisor y receptor en dispositivos LoPy. De esta forma, se pretende utilizar la tecnología de comunicación *VLC* en dispositivos comunes en el desarrollo de aplicaciones IoT. Antes de comenzar a analizar la adaptación de los circuitos *VLC* para su integración con los dispositivos LoPy, se debe presentar el propio dispositivo, fabricado y comercializado por la empresa Pycom.

El dispositivo LoPy, mostrado en la Figura 3.24, ha sido desarrollado por la empresa Pycom [52]. LoPy [53] es una placa de desarrollo que integra tres tecnologías de comunicación en todas sus versiones: *Wireless Fidelity* (*WiFi*), *Bluetooth Classic*, *Bluetooth Low Energy (BLE)* y LoRa. En versiones más actuales cuenta también con la tecnología LPWAN *Sigfox* [54]. La programación del dispositivo se puede realizar en el lenguaje *MicroPython*. La placa ha sido diseñada específicamente para aplicaciones de IoT, como la que se presenta en este TFM.

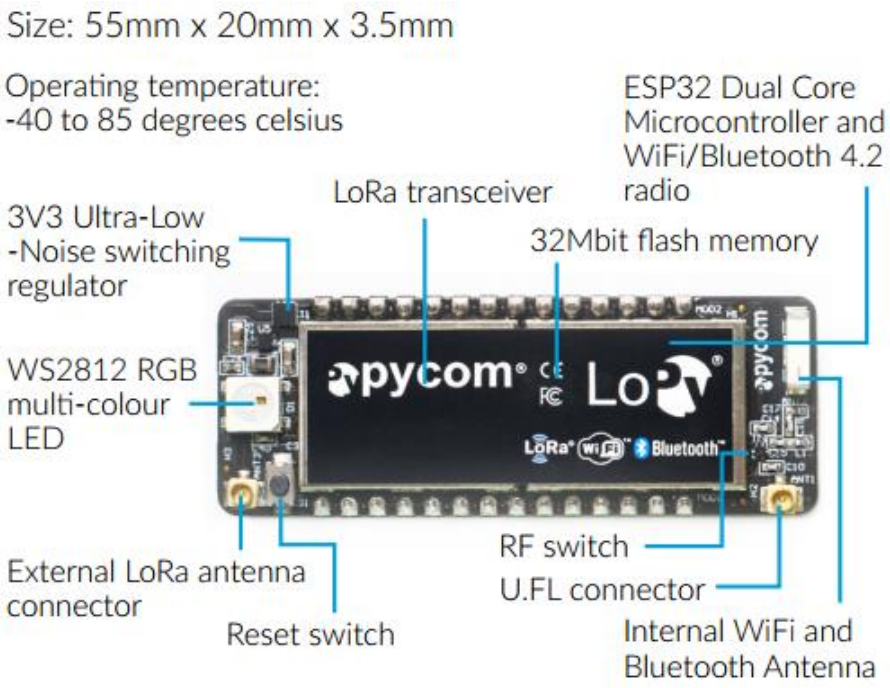


Figura 3.24: LoPy de Pycom.

En su núcleo se encuentra el Sistema en Chip (SoC) Espressif ESP32 que dispone de un microcontrolador de núcleo doble, WiFi, Bluetooth, LoRa y 512 KB de Random Access

Memory (RAM). Como apunte importante se debe comentar que el procesador principal está completamente disponible para ejecutar la aplicación del usuario.

Las principales características técnicas del dispositivo LoPy1.0 son las siguientes [55]:

- Transceptor *Semtech LoRa* SX1272 con LoRaWAN, compatible con dispositivos de Clase A y C.
- Antena interna *WiFi* y *Bluetooth* y conector U.FL de antena *LoRa*.
- Memoria *flash* de 4 MB.
- 2 x UART, 2 x SPI, I^2C, *Inter-IC Sound* (I^2S), tarjeta MicroSD, 8 ADC de 12 bits, Timers: 4x16 bit con *Pulse-Width Modulation* (PWM) y hasta 24 GPIO.
- LED *Red-Green-Blue* (RGB) WS2812.
- *Real Time Clock* (RTC) - 32 KHz.
- IPv6.
- Interrupciones para *Radio Frequency* (RF) y reajuste.
- Temperatura de funcionamiento: -40 °C a 85 °C.
- Dimensiones: 55 x 20 x 3,5 mm.
- 868 MHz (Europa) a +14 dBm máximo.
- 915 MHz (Norteamérica y Sudamérica, Australia y Nueva Zelanda) a +20 dBm máximo.
- Alcance del nodo: hasta 40 km.
- *Nano-Gateway*: hasta 22 km.
- Capacidad *Nano-Gateway*: hasta 100 nodos.

Con respecto al consumo de potencia del dispositivo, se tienen las siguientes especificaciones [55]:

- Entrada: 3,3 V a 5,5 V.
- Salida: 3,3 V con hasta 400 mA.
- *WiFi*: 12 mA en modo activo, 5 µA en espera.
- *LoRa*: 15 mA en modo activo, 10 µA en espera.

El diagrama de bloques interno del dispositivo LoPy se muestra en la Figura 3.25. En este diagrama se puede observar en el núcleo del dispositivo al SoC que gobierna el funcionamiento del dispositivo, los módulos de comunicación *LoRa*, *Bluetooth* y *WiFi* con sus respectivos conectores para las antenas, las interfaces de comunicación disponibles como SPI o I^2C, un regulador de tensión, un LED RGB WS2812B, un botón de reset y una memoria *Flash*.

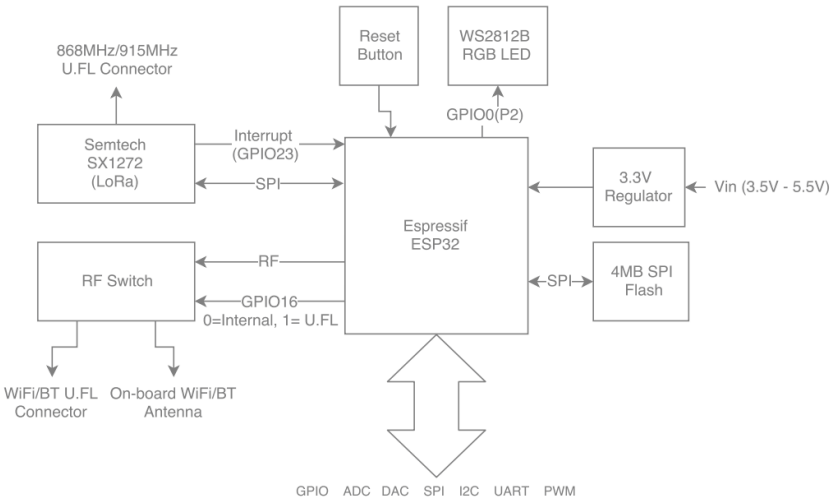


Figura 3.25: Diagrama de bloques del dispositivo LoPy.

Para hacer uso de todos los periféricos que incluye el dispositivo LoPy, se cuenta con 28 pines, distribuidos como se muestra en la Figura 3.26.

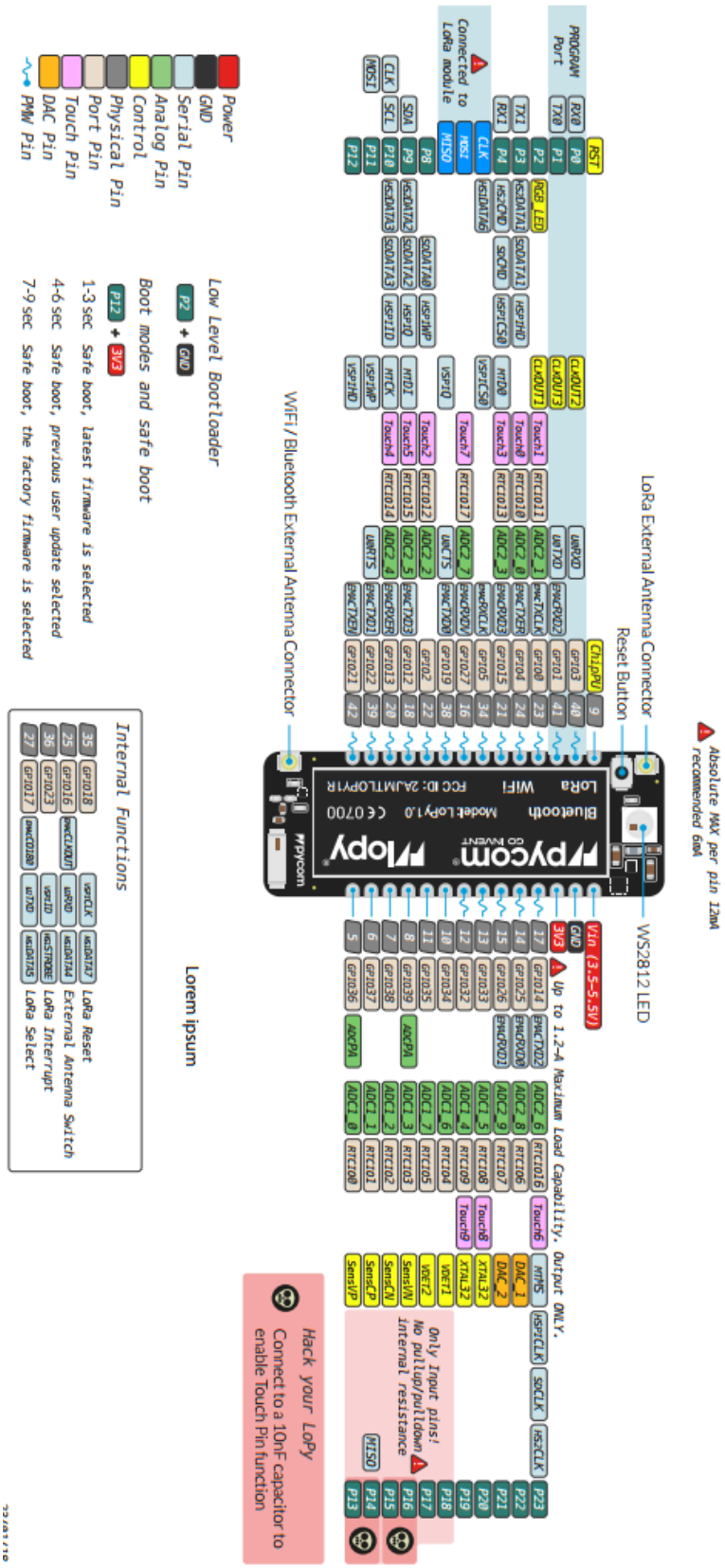


Figura 3.26: Pinout del dispositivo LoPy.

Además del propio dispositivo LoPy, se hace uso de una placa de extensión que ofrece la misma empresa, Pycom. Con la *Expansion Board* [56], que se muestra en la Figura 3.27, se puede programar el dispositivo LoPy a través de su puerto *Micro Universal Serial Bus* (MicroUSB), que sirve de alimentación y de comunicación serie. Además, cuenta con una conexión más cómoda para acceder a los pines del dispositivo LoPy, así como un conector de tarjeta *Micro Secure Digital* (MicroSD), y de batería de Polímero de Litio (LiPo).

El diagrama de bloques de la *Expansion Board* de Pycom se muestra en la Figura 3.28. En ella se pueden observar las dos vías de alimentación de las que dispone la placa, y sus principales bloques funcionales.

Para ubicar cada una de las funcionalidades que incorpora la placa de expansión, en la Figura 3.29 se adjunta un esquemático que enumera los principales elementos que integra.

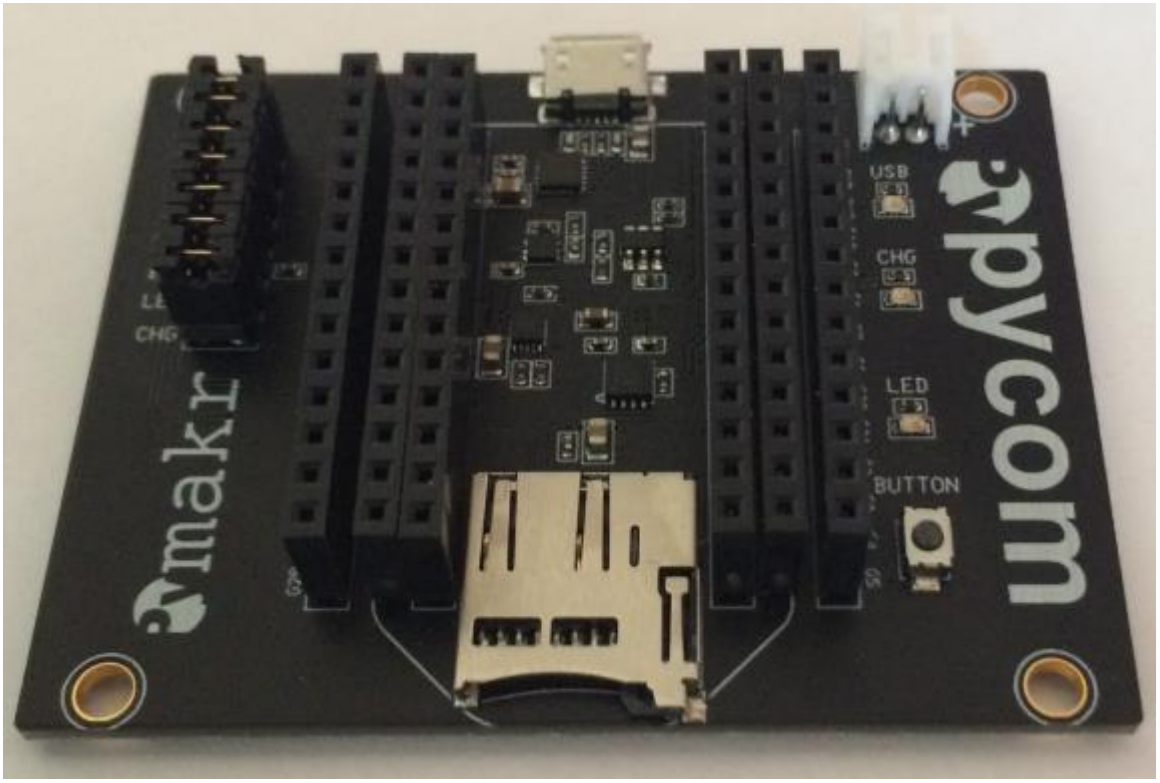


Figura 3.27: Expansion Board.

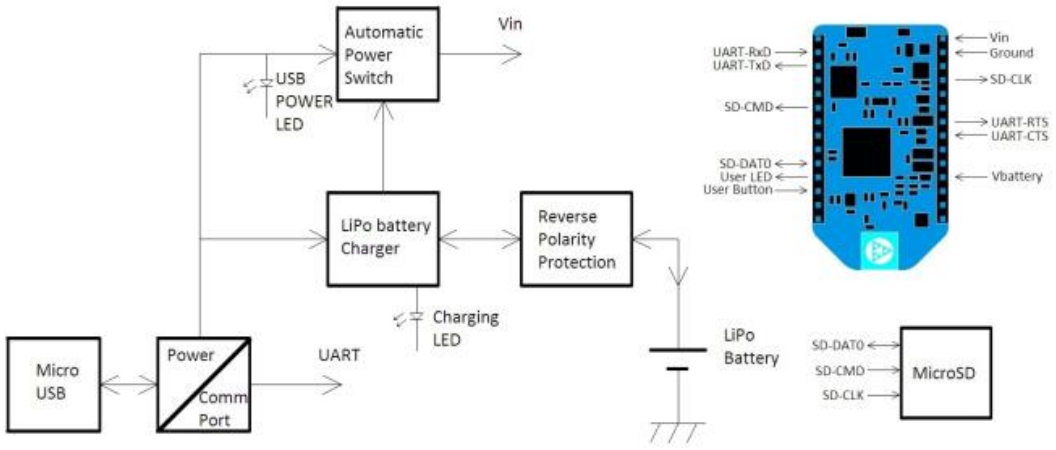
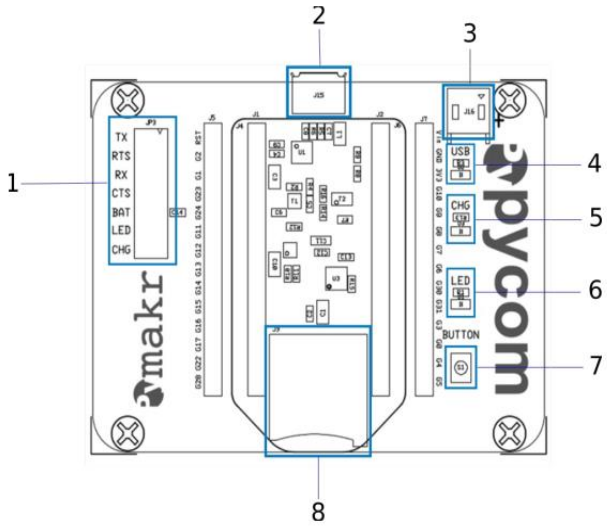


Figura 3.28: Diagrama de bloques.



1: Jumpers selectores de funcionalidad.

2: Conector Micro USB.

3: Conector JST de batería LiPo.

4: LED de carga USB.

5: LED indicador de carga de batería.

6: LED de usuario.

7: Botón de usuario.

8: *Socket* de tarjeta MicroSD.

Figura 3.29: Utilidades de la Expansion Board.

La *Expansion Board* también cuenta con unos pines cuya funcionalidad varía en función de si el *jumper* correspondiente está conectado (cerrado), o no (abierto). En la Tabla 3.16 se recoge la información asociada a estos pines.

Número de Jumper	Nombre del Pin	Jumper cerrado	Jumper abierto
1	GPIO2	RxD	GPIO2
2	GPIO6	CTS	GPIO6

3	GPIO1	TxD	GPIO1
4	GPIO7	RTS	GPIO7
5	GPIO3	VBat	GPIO3
6	GPIO16	Habilitado el LED de usuario	Deshabilitado el LED de usuario
7	-	Corriente de carga de batería: 450 mA	Corriente de carga de batería: 100 mA

Tabla 3.16: Funciones asociadas al estado de los Jumper en la Expansion Board.

La programación del dispositivo LoPy, como se explicó anteriormente se puede realizar en lenguaje *MicroPython*. Existen varios *Integrated Development Environment* (IDE) que soportan, no sólo este lenguaje de programación, sino también el *plugin Pymakr* que ofrece Pycom para la comunicación con estos dispositivos desde los IDE que permiten su instalación. En el caso de este TFM, se ha utilizado como IDE la herramienta ATOM [57]. Se trata de un editor de código abierto para macOS, Linux, y Windows con soporte para *plugins* codificados en *Node.js* y control de versiones *Git* integrado, desarrollado por GitHub [58].

Para poder trabajar en este editor con los dispositivos de Pycom es necesario instalar el *plugin Pymakr* [59] que se comentó anteriormente. Este paquete añade una consola *Read-Eval-Print-Loop* (REPL) al editor Atom que permite la conexión con los dispositivos Pycom y ejecutar código en ellos. La principal diferencia entre una consola REPL y la ejecución de un *script*, es que en REPL múltiples clases o funciones imprimen por pantalla automáticamente su respuesta, mientras que en el caso de los *scripts*, no se muestra necesariamente la respuesta por pantalla [60].

Con el editor y el *plugin* instalado ya se puede proceder a la programación de la funcionalidad del dispositivo. En la Figura 3.30 se muestra la interfaz principal del entorno Atom. En la interfaz mostrada se puede ver que en la zona superior izquierda se encuentra la barra de herramientas que permite gestionar el entorno. También se cuenta con un espacio para visualizar el proyecto que se encuentra abierto, con sus respectivos *scripts*. La mayor parte del espacio habilitado se destina a la edición de los *scripts*. Por otra parte, en

la zona inferior de la interfaz se puede visualizar la barra de herramientas asociada al *plugin Pymakr*. Por último, debajo de la barra de herramientas *Pymakr* se muestra la ventana de comandos, en la que se pueden ejecutar instrucciones y aparece la respuesta del *software* correspondiente.

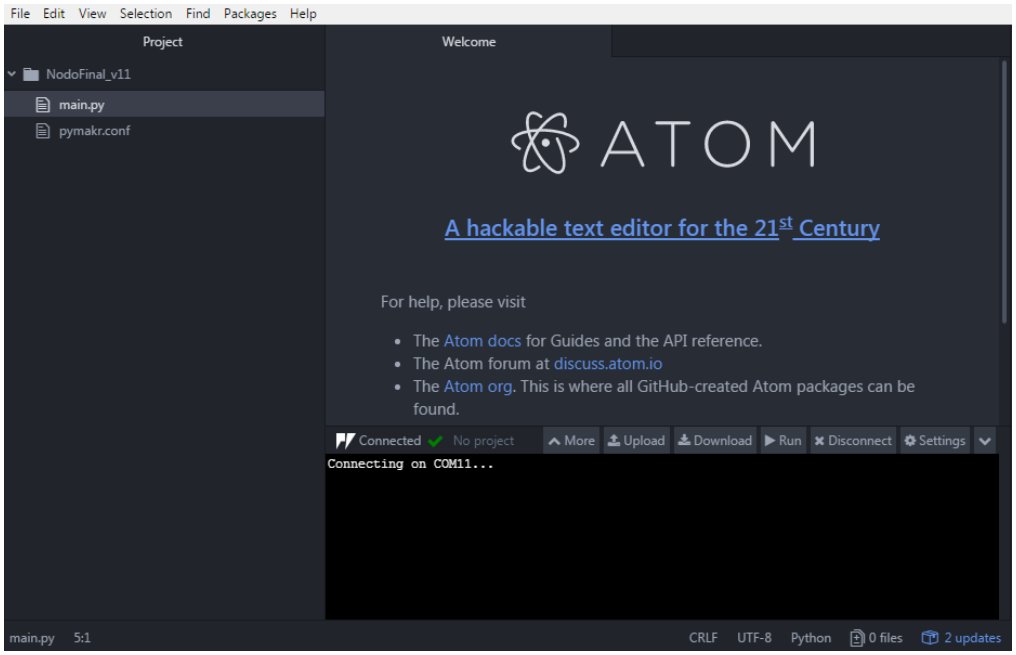


Figura 3.30: Interfaz principal del entorno Atom.

Una vez que se conoce el dispositivo LoPy y el IDE en el que se va a codificar la comunicación de prueba *VLC*, se puede pasar a comentar las modificaciones de los circuitos desarrollados hasta este momento, para su correcta integración con dicho dispositivo.

La primera modificación que se realizó sobre el circuito transmisor fue la propia entrada de señal que se aplica al circuito. En los montajes 1, 2 y 3 la señal que se deseaba transmitir se recibía desde un generador de funciones con un nivel de señal fijo de 5 V. El dispositivo LoPy, sin embargo, en las señales que se obtienen en sus pines tienen un nivel de 3.3 V. Esta es la razón por la que se debe incluir un módulo convertidor de nivel de señal antes de aplicar las señales de salida del dispositivo LoPy al circuito transmisor diseñado. El dispositivo utilizado para cumplir con este propósito es el *Bi-directional Logic Level Converter* de Sparkfun [61]. Este dispositivo permite convertir las señales con niveles lógicos entre 0 y 3,3 V a señales con niveles lógicos entre 0 y 5 V y viceversa. Los diferentes niveles lógicos de conversión del dispositivo vienen determinados por la tensión con la que

se alimente los pines LV y HV de este dispositivo. En la Figura 3.31 [61] se muestra el *pinout* del convertidor desarrollado por Sparkfun.

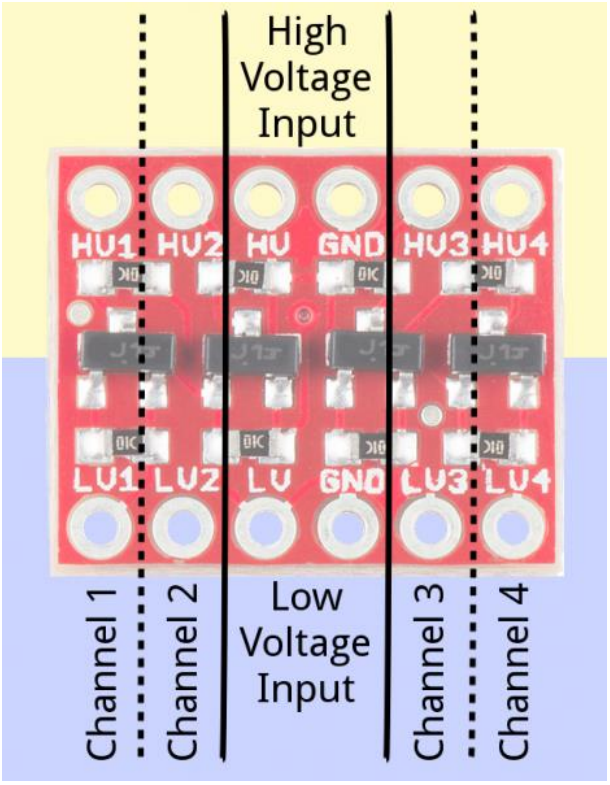


Figura 3.31: Pinout del dispositivo Bi-directional Logic Level Converter.

En este dispositivo se puede observar como el pin LV con su correspondiente pin GND, y el pin HV con su correspondiente pin GND, son de entrada de tensión, y son los que determinan los dos niveles lógicos de la conversión. Una vez que se fijan los dos niveles lógicos con los que se desea trabajar, cada entrada en uno de los cuatro canales tiene su salida en el canal correspondiente del otro nivel lógico. Por ejemplo, si se fija como nivel lógico bajo la tensión de 3.3 V, y a nivel lógico alto la tensión de 5 V, al introducir una señal de nivel 3.3 V en el canal 1 del nivel lógico bajo, es decir, en el pin LV1, se obtendrá como salida, en el pin HV1, la misma señal, pero un nivel lógico de 5 V. Se pueden utilizar los cuatro canales de conversión disponibles de forma simultánea. Internamente el esquemático individual de cada canal de conversión es el que se muestra en la Figura 3.32 [61].

Para el montaje que se desea realizar en este TFM, se alimentan los dos niveles lógicos a 3.3 V y a 5 V. Y se utiliza uno de los canales para realizar la conversión de nivel lógico desde el nivel bajo (3.3 V) al nivel alto (5 V). La alimentación del dispositivo *Bi-directional Logic*

Level Converter se realiza desde el pin '3V3' de la *Expansion Board* del dispositivo LoPy para el nivel lógico bajo, y desde el pin 'Vin' también de la *Expansion Board* del dispositivo LoPy para el nivel lógico alto.

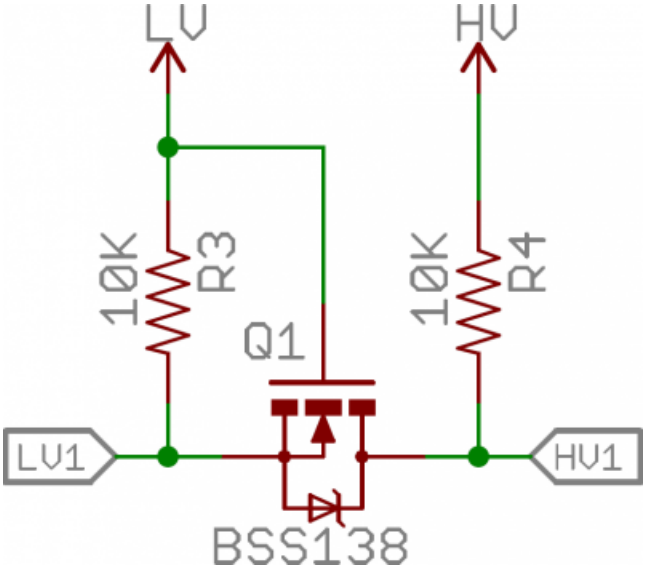


Figura 3.32: Circuito interno de cada canal de conversión.

La señal digital de nivel lógico bajo que proviene de la UART del dispositivo LoPy, y que se desea convertir a nivel alto, se conecta al pin 'LV1' del dispositivo *Bi-directional Logic Level Converter,* obteniéndose la misma señal, pero con nivel lógico alto en el pin 'HV1'. Desde este último pin se lleva la señal hacia el circuito propuesto para la transmisión *VLC* del montaje 3, que se estudió en el apartado anterior.

Así, el montaje queda de forma similar al que se muestra en la Figura 3.33. En el ejemplo de la Figura 3.33 se ha incluido un diodo LED en lugar de la lámpara por motivos de disponibilidad de librerías gráficas, pero el conexionado es el mismo que el montado. En la Figura 3.34 se muestra la leyenda asociada a la Figura 3.33. En ella se puede obervar que la señal de nivel lógico de entrada proviene del pin 'G24' de la *Expansion Board* del dispositivo LoPy. Este pin es el determinado por el propio *pinout* para la transmisión en la comunicación UART.

Tras realizar el conexionado de la Figura 3.33, el circuito transmisor *VLC* queda como se muestra en la Figura 3.35. La señal de entrada proviene del pin 'HV1' del dispositivo *Bi-directional Logic Level Converter,* y la alimentación del circuito proviene directamente del dispositivo LoPy, mediante los pines 'VIN' y 'GND' de la *Expansion Board*.

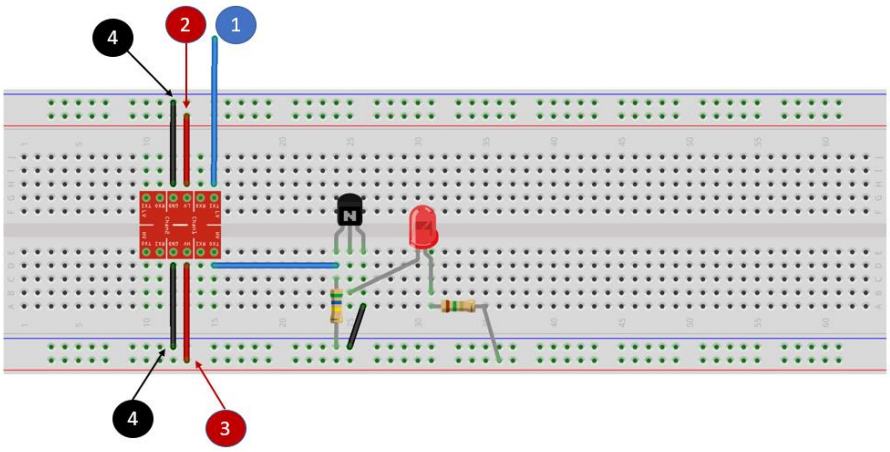


Figura 3.33: Conexionado LoPy-Level Converter-circuito transmisor VLC.

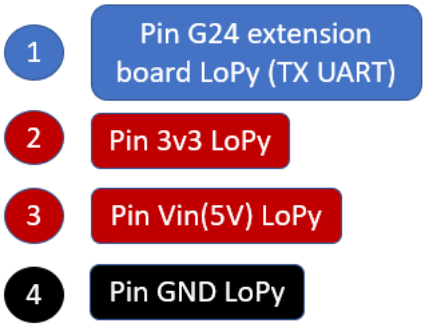


Figura 3.34: Leyenda asociada a los pines del conexionado LoPy-Level Converter-circuito transmisor VLC.

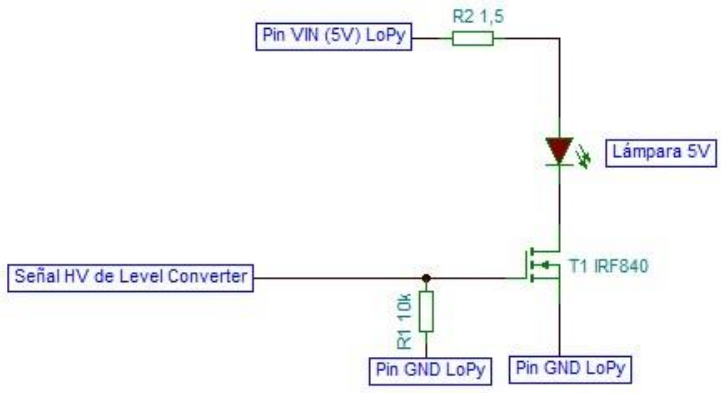


Figura 3.35: Primera versión del circuito transmisor VLC para la integración con LoPy.

No obstante, con la configuración *hardware* actual del circuito transmisor *VLC*, la lámpara no se encendía debido a que no se le proporcionaba corriente suficiente. La solución ante

este inconveniente fue incrementar el valor de la resistencia R1, según la notación del circuito de la Figura 3.35. El valor de esta resistencia se fijó en función de que se cumpliera la comunicación de alcance de 50 cm. En la Tabla 3.17 se muestran los resultados experimentales obtenidos para diferentes valores de resistencia R1. Como se puede ver en esta Tabla 3.17, las resistencias de 270 KΩ y de 560 KΩ son los valores más bajos que permiten establecer una comunicación de al menos 50 cm entre transmisor y receptor. Se ha escogido la resistencia de 560 KΩ por contar con mayor luminosidad, aunque ambas son válidas para el montaje objetivo.

Valor de R1	Resultado de comunicación obtenido
R1=10 KΩ	No se enciende la lámpara.
R1=56 KΩ	Se enciende (con poca luminosidad) pero no se recibe señal en el receptor.
R1=67 KΩ	Se enciende (con poca luminosidad) pero no se recibe señal en el receptor.
R1=123 KΩ	Se enciende (con luminosidad mejorable) pero no se recibe señal en el receptor.
R1=270 KΩ	Se enciende y se recibe bien la señal en el receptor.
R1=560 KΩ	**Se enciende (con mayor luminosidad) y se recibe bien la señal en el receptor.**
R1 > 560 KΩ	No se aprecia mejora ni en luminosidad de la lámpara ni en la señal recibida.

Tabla 3.17: Influencia del valor de R1 en la comunicación del montaje 4.

Por lo tanto, el circuito transmisor *VLC* resultante queda como el que se muestra en la Figura 3.36.

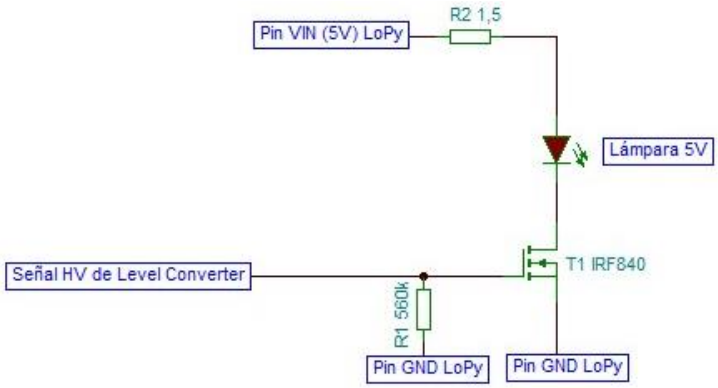


Figura 3.36: Circuito transmisor VLC utilizado para la integración con LoPy.

En cuanto al receptor, este también debe modificar su forma de alimentación, ya que se va a integrar en otro dispositivo LoPy, y por lo tanto ya no se alimentará desde una fuente de alimentación, como en los montajes 1, 2 y 3, sino que la alimentación la recibe del propio dispositivo LoPy o de la batería que este dispositivo integra. En la Figura 3.37 se muestra el circuito receptor *VLC* utilizado para la integración con el dispositivo LoPy. Como se puede ver en este esquemático, la salida del comparador se conecta directamente al pin 'G11' de la *Expansion Board* asociada al dispositivo LoPy que es el definido según el *pinout* correspondiente para realizar la recepción de la comunicación UART. También se deben conectar al pin 'GND' de la *Expansion Board* todos aquellos puntos del circuito que van a masa.

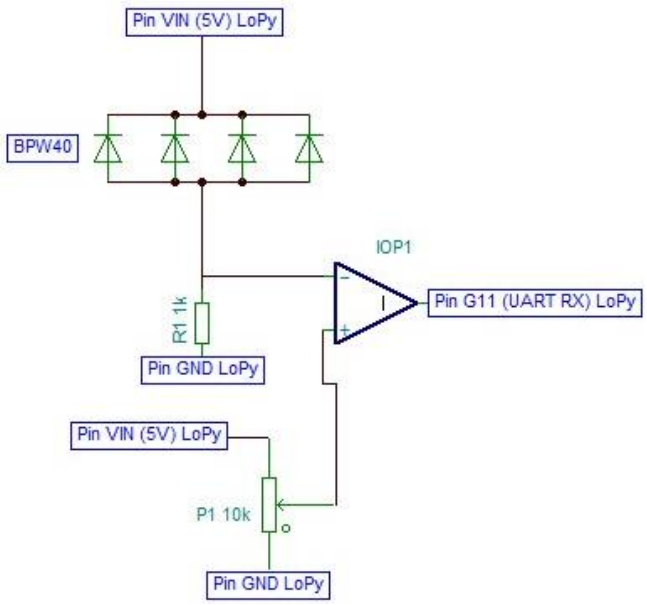


Figura 3.37: Circuito receptor VLC utilizado para la integración con LoPy.

En este punto ya se cuenta con los dos circuitos preparados y totalmente funcionales para poder llevarlos a una prueba de comunicación final. Para realizar esta prueba se debe codificar primero la funcionalidad del dispositivo LoPy que actuará como transmisor, y la del dispositivo LoPy que actuará como receptor. En la Figura 3.38 se muestra la codificación de la funcionalidad del transmisor, mientras que en la Figura 3.39 la funcionalidad del receptor. Como se explicó anteriormente, se ha utilizado el IDE ATOM, que permite la programación en lenguaje *MicroPython*, habiéndose usado el *plugin Pymakr* para comunicarse con el dispositivo desde el IDE.

Tanto en la Figura 3.38 como en la Figura 3.39 se puede ver cómo se usan diferentes librerías que son fundamentales para el correcto desarrollo de la codificación. Estas librerías son la librería *Pycom* para gestión de elementos *hardware* genéricos del propio dispositivo, la librería *UART* para la comunicación entre el dispositivo LoPy y los circuitos de transmisión y recepción, así como la librería *Binascii* que permite manejar los datos en formato binario o ASCII.

También en ambos casos se configura el periférico de comunicación UART y se inicializa. Para ello se establece la tasa de baudios a 9600 baudios, los bits de datos a 8 bits, se especifica que no se desea paridad y que se incluya 1 bit de stop. El objeto UART creado debe indicar los pines correspondientes, que según el *pinout* son los pines 'P3' y 'P4'. Estos pines 'P3' y 'P4' se corresponden con los pines de la *Expansion Board* 'G24' y 'G11' respectivamente, que son los pines a los que se conectan a los circuitos transmisores y receptores.

En la Figura 3.38, correspondiente a la codificación de la funcionalidad del transmisor *VLC*, se crea una variable de tipo *bytearray* de tamaño cinco y se inicializa con unos valores determinados. Una vez que se entra en el bucle '*while True*', es decir, un bucle que se ejecuta continuamente, se envía de forma repetida por la UART la variable de tipo *bytearray* creada e inicializada. Por último, se imprime por pantalla cada valor de dicha variable.

```
from machine import UART
import time
import binascii
import pycom

pycom.heartbeat(False)
print('Main start transmisor')
uart = UART(1, baudrate=9600, pins=('P3','P4'))
uart.init(9600, bits=8, parity=None, stop=1) # init with given parameters
buf= bytearray(5)
buf[0]=0x0A
buf[1]=0x0B
buf[2]=0x0C
buf[3]=0x0D
buf[4]=0xF2
while True:
# this uses the UART_1 default pins for TXD and RXD (``P3`` and ``P4``)
   uart.write(buf)
   print('0x{0:02x}'.format(buf[0]))
   print('0x{0:02x}'.format(buf[1]))
   print('0x{0:02x}'.format(buf[2]))
   print('0x{0:02x}'.format(buf[3]))
   print('0x{0:02x}'.format(buf[4]))
```

Figura 3.38: Codificación de la funcionalidad del transmisor VLC.

En la Figura 3.39, correspondiente a la codificación de la funcionalidad del receptor *VLC*, se crea también una variable de tipo *bytearray* de tamaño cinco donde se desea almacenar los valores recibidos desde la UART. Una vez que se entra en el bucle '*while True*' se realiza una lectura constante de la UART y se almacena dicha lectura en la variable de tipo *bytearray* creada. Del mismo modo que en el transmisor, se imprime por pantalla cada valor de dicha variable.

```
from machine import UART
import time
import binascii
import pycom

pycom.heartbeat(False)
print('Main start receptor')
uart = UART(1, baudrate=9600, pins=('P3','P4'))
uart.init(9600, bits=8, parity=None, stop=1) # init with given parameters
buf= bytearray(5)
while True:
# this uses the UART_1 default pins for TXD and RXD (``P3`` and ``P4``)
   uart.readinto(buf) # read 1 byte
   print('0x{0:02x}'.format(buf[0]))
   print('0x{0:02x}'.format(buf[1]))
   print('0x{0:02x}'.format(buf[2]))
   print('0x{0:02x}'.format(buf[3]))
   print('0x{0:02x}'.format(buf[4]))
```

Figura 3.39: Codificación de la funcionalidad del receptor VLC.

Como evidencia de la correcta comunicación UART en cada terminal y de la comunicación *VLC* entre ambos terminales (transmisor y receptor) se muestran los resultados obtenidos tras la ejecución de las dos codificaciones representadas en la Figura 3.38 y en la Figura 3.39. Así, en la Figura 3.40 se muestran los valores leídos desde la UART y que han sido impresos por pantalla.

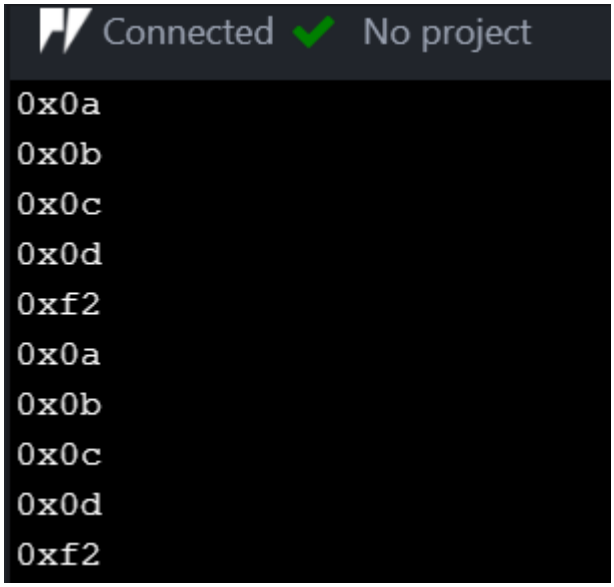


Figura 3.40: Resultado de la correcta comunicación entre transmisor y receptor VLC.

En la Figura 3.40 se puede observar que los valores leídos son exactamente los mismos que los enviados por el transmisor, es decir se ha establecido una comunicación correcta y estable entre ambos terminales.

Físicamente, el montaje final de este enlace 1 ha quedado tal y como se muestra en la Figura 3.41.

En la Figura 3.42 se muestra la señal a nivel lógico alto correspondiente a los datos enviados desde el dispositivo LoPy, una vez ha pasado por el dispositivo *Bi-directional Logic Level Converter*. Es decir, se trata de la señal de entrada del circuito transmisor *VLC*, o bien la salida del canal 1 del dispositivo *Bi-directional Logic Level Converter*, visto desde otra perspectiva. Los mandos del osciloscopio se encuentran en 2V/div y 500µs/div.

Figura 3.41: Montaje final del enlace 1.

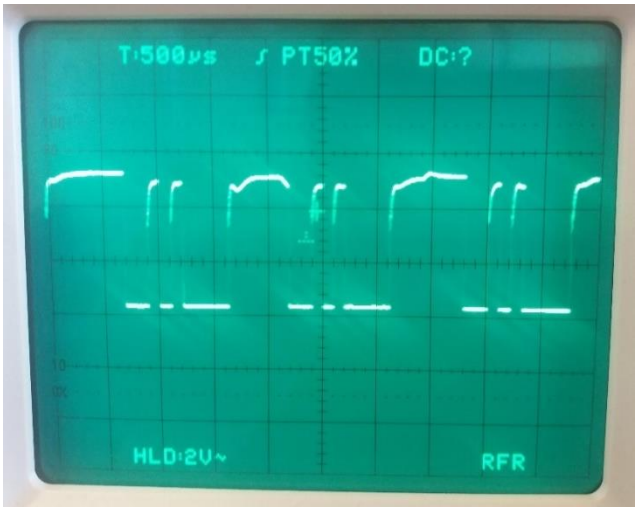


Figura 3.42: Señal a la salida del dispositivo Bi-directional Logic Level Converter.

En la Figura 3.43 se muestra la señal a la salida del comparador del circuito receptor *VLC*. En este caso los mandos del osciloscopio se encuentran en 2V/div y 500µs/div.

Ante los resultados obtenidos, se da por concluido el desarrollo *hardware* de la comunicación *VLC* del enlace 1. Se ha logrado establecer una comunicación por luz visible a 50 cm de distancia y se ha integrado dicha comunicación en dispositivos de carácter IoT.

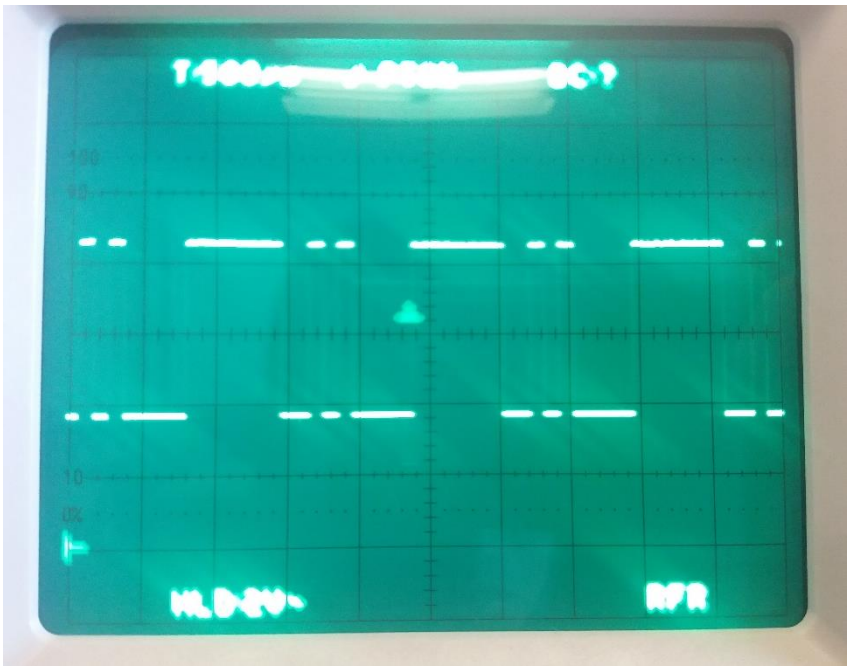


Figura 3.43: Señal a la salida del comparador en el circuito receptor VLC.

3.2.2 Enlace 2: Transmisor en nodo móvil y receptor en nodo estación base

En este segundo enlace se considera que el receptor se puede conectar a la infraestructura eléctrica del inmueble en cuestión, por lo que puede ser alimentado de forma continuada y por lo tanto no se cuenta con una restricción de consumo de potencia. Tanto el transmisor como el receptor deben ser alimentados a partir del dispositivo LoPy, o directamente desde la batería que alimenta a este. Es decir, la tensión de alimentación será en ambos casos de 5V. La comunicación entre los módulos *VLC* utilizados y los dispositivos LoPy es en este caso también de tipo serie, haciendo uso de las UART que ofrece LoPy entre sus recursos. El objetivo perseguido con este enlace es obtener comunicación por luz visible para una distancia de 50 cm. El alcance máximo que se puede lograr, como se verá en los resultados de este capítulo, es menor que en el enlace 1, ya que en el transmisor no se hace uso de una lámpara debido a su elevado consumo. En este caso, el transmisor se encuentra en el nodo móvil, por lo que no se puede alimentar de forma externa, ni se puede cargar la batería, es decir, en este caso sí se encuentra una restricción de consumo de potencia que limita el uso de tecnologías como la lámpara utilizada en el enlace 1.

Teniendo en cuenta estas planteamiento y las restricciones que conlleva, a continuación, se muestra el procedimiento seguido para el desarrollo de los dos elementos que forman este enlace.

3.2.2.1 Planteamiento inicial: Montaje 1

Con respecto al circuito transmisor, se propone un circuito similar al implementado para el caso del enlace 1, salvando las distancias relacionadas por la no posibilidad de usar la lámpara de 5 V. El circuito se alimenta con 5 V que provienen directamente del dispositivo LoPy, o de la batería que este lleva integrada. La señal que se desea transmitir proviene del pin 'G24' de la *Expansion Board* del dispositivo LoPy, que como se indicó en el apartado anterior, corresponde según el *pinout* del propio dispositivo al pin de transmisión de la comunicación UART. En este caso no es necesario realizar una conversión de nivel lógico de la señal transmitida desde la UART, ya que el LED que se va a utilizar en este circuito transmisor responde adecuadamente a la señal transmitida directamente desde la UART del dispositivo LoPy. En la Figura 3.44 se muestra el circuito de transmisión propuesto inicialmente para el enlace 2. En este caso el transistor utilizado es el BJT BC547B, que se fuerza a un en estado de corte/saturación con el fin de que funcione como interruptor electrónico. El valor de la resistencia R1 influye directamente en el consumo de potencia del LED ya que regula la corriente que circula. Inicialmente se fija a 120 Ω. Por otra parte, la R2 tiene un valor de 330 Ω con el fin de configurar el circuito correctamente para que el transistor se encuentre en estado de saturación.

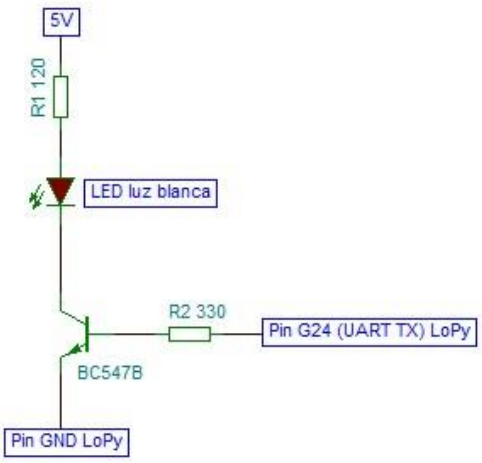


Figura 3.44: Circuito transmisor inicial para el enlace 2.

Existen diferentes tipos de LED que podrían ser apropiados para el circuito transmisor propuesto en la Figura 3.44. En el presente TFM se han probado tres tipos de LED: LED de luz blanca, LED de luz roja y LED infrarrojo. Tal y como se puede ver en la Figura 3.45, los diodos LED de luz blanca (primero por la izquierda) y luz roja (centro) se iluminan correctamente. Como se verá en los resultados posteriormente, ambos cumplen con los requisitos de comunicación deseados, al igual que el diodo LED infrarrojo que por su característica de estar fuera del rango de la luz visible el ojo humano no es posible ver. No obstante, se muestra el montaje de este diodo (primero por la derecha) también en la Figura 3.45. Todos los LED contaban con las mismas condiciones de circuito y alimentación.

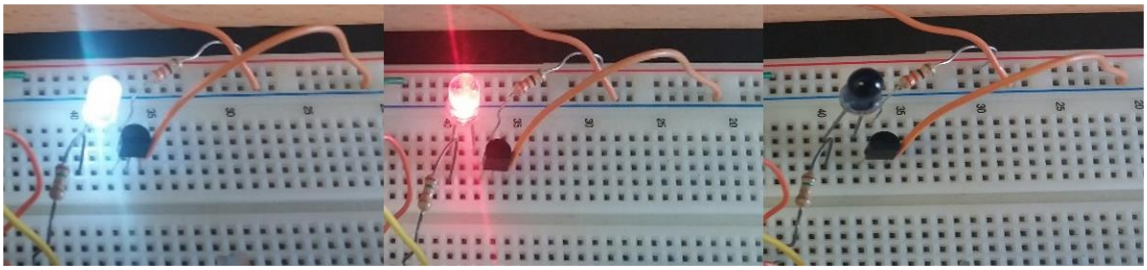


Figura 3.45: Utilización de varios diodos LED para la transmisión del enlace 2.

Al poder mantener una comunicación estable con los tres tipos de LED, como se verá posteriormente, la decisión de seleccionar uno u otro se fundamenta en el consumo de potencia de cada uno, ante las mismas condiciones de comunicación. En la Tabla 3.18 se muestra la relación entre cada tipo de LED y su consumo medio en continua medido con un multímetro, y el pico de corriente alcanzado por cada uno medido con el osciloscopio.

Tipo LED	Corriente LED (multímetro- DC)	Corriente pico LED (osciloscopio)
LED luz blanca	1,7 mA	90 mA
LED luz roja	2,56 mA	140 mA
LED infrarrojo	3,23 mA	160 mA

Tabla 3.18: Relación LED – consumo de potencia.

A la vista de los resultados obtenidos se puede ver que para las mismas condiciones de comunicación (50 cm de distancia, misma señal transmitida, mismo nivel de alimentación

y mismo circuito transmisor) el diodo LED de luz blanca presenta un consumo inferior al de los diodos LED rojo e infrarrojo, siendo este último el de mayor consumo. También se puede valorar que, ante estos resultados, el consumo de potencia obtenido es bastante elevado para lo esperado y deseado en un dispositivo IoT que se alimenta únicamente a partir de batería o pila de botón. Es decir, se debe modificar el circuito transmisor actual con el objetivo de disminuir el consumo de potencia. En el siguiente apartado se estudia el nuevo circuito transmisor desarrollado para el enlace 2.

Con respecto al receptor del enlace 2, se plantea hacer uso de un fotodiodo PIN. El fotodiodo tiene el mismo funcionamiento que un fototransistor, pero no cuenta con la ganancia intrínseca de estos. El planteamiento inicial es recoger la señal del fotodiodo PIN y transmitirla a un amplificador de transimpedancia con su respectivo comparador a la salida. El circuito receptor planteado para el enlace 2 se muestra en la Figura 3.46.

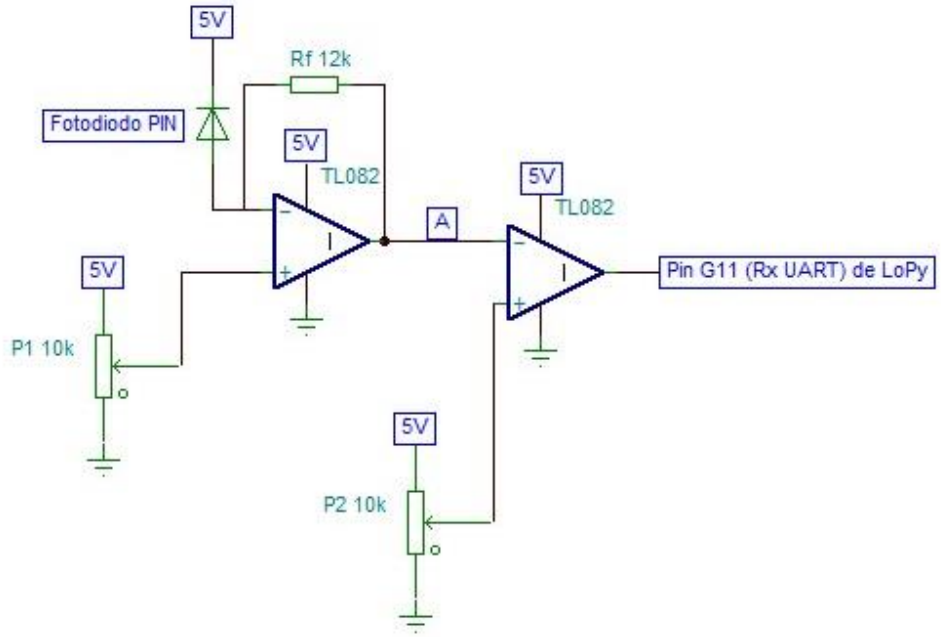


Figura 3.46: Circuito receptor inicial para el enlace 2.

En este circuito se han utilizado potenciómetros en las entradas no inversoras de los amplificadores operacionales TL082, siendo la alimentación de estos asimétrica de 5 V, como se puede observar en la Figura 3.46. El funcionamiento del circuito consiste en el aumento de la circulación de corriente cuando incide luz sobre el fotodiodo, así como la disminución de esa circulación cuando se deja de recibir iluminación sobre el fotodiodo. Tras esta etapa se recoge la señal del fotodiodo PIN y se pasa por un amplificador de

transimpedancia en el que mediante el valor de tensión introducido en la entrada no inversora del amplificador operacional y el valor de la resistencia Rf se amplifica la señal. Finalmente, en la última etapa se añade un comparador.

El montaje 1 del enlace 2 completo se ilustra en la Figura 3.47. Para una primera prueba, en la que la distancia de comunicación es de 16 cm y la señal se transmite desde el circuito transmisor de la Figura 3.44, los resultados obtenidos se consideran satisfactorios, obteniendo en el punto A del receptor, según la Figura 3.46, la señal mostrada en la Figura 3.48 (señal superior). Por su parte, a la salida del comparador, es decir, la señal que se va a introducir en el pin 'G11' de la *Expansion Board* del dispositivo LoPy correspondiente a la recepción de comunicación UART, se obtiene la señal de la Figura 3.48 (señal inferior). Los mandos del osciloscopio se encuentran configurados como 1 V/div para el canal 1 (señal superior) y 2 V/div para el canal 2 (señal inferior), mientras que la referencia temporal es de 200 µs/div. En la Figura 3.48 se puede observar cómo la señal recibida a la salida del comparador presenta una forma de onda aceptable, pudiendo introducirse dicha señal en el pin 'G11' de la *Expansion Board* del dispositivo LoPy.

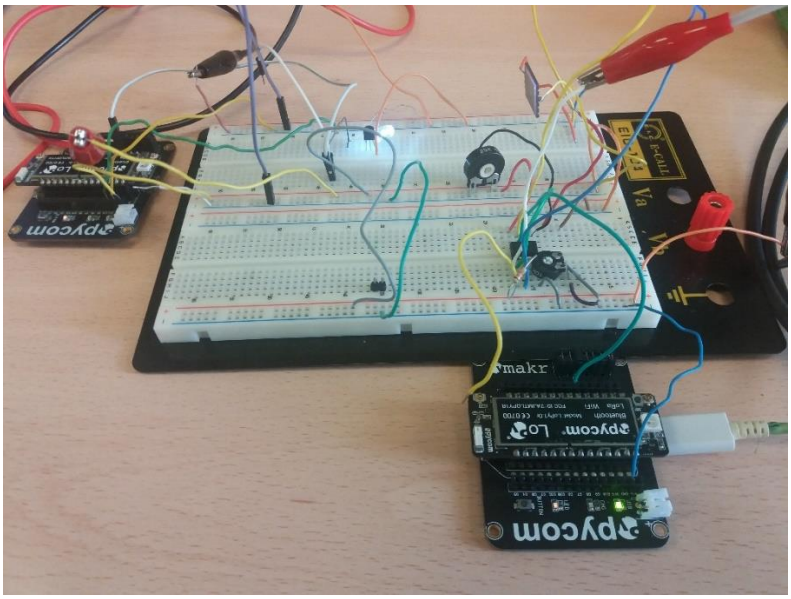


Figura 3.47: Montaje 1 del enlace 2.

Figura 3.48: Señal recibida en el circuito receptor inicial del enlace 2.

No obstante, a pesar de los resultados positivos de la prueba de comunicación realizada entre los circuitos transmisor y receptor representados en la Figura 3.44 y en la Figura 3.46, esta comunicación se ha realizado para una distancia de 16 cm. Por lo tanto, se debe comprobar si este mismo montaje es capaz de mantener la comunicación en un enlace de 50 cm.

Tras varias pruebas de comunicación en la que se iba aumentando la distancia y comprobando el resultado obtenido en la señal captada por el fotodiodo PIN, así como la señal a la salida del comparador, se comprobó que la distancia máxima de comunicación entre el circuito transmisor y el receptor era de 25 cm. A pesar de la ganancia que aporta el circuito de transimpedancia, el fotodiodo PIN no cuenta con la ganancia interna ni con elementos como lentes concentradoras que permitan una mejor recepción en distancias más largas. Esta distancia máxima de comunicación es la mitad de la distancia establecida inicialmente, por lo que se debe modificar el circuito receptor para poder establecer la comunicación deseada. En el siguiente apartado se comenta la solución propuesta.

3.2.2.2 Planteamiento final e integración con el dispositivo LoPy

Como se estudió en el apartado anterior, el circuito transmisor funcionaba correctamente, pero su consumo de potencia era mejorable. En esta nueva propuesta de circuito de transmisión para el enlace 2 se decidió aumentar el valor de la resistencia del colector etiquetada en la Figura 3.49 como R1. Aumentando el valor de esta resistencia, se pretende

disminuir el consumo de potencia. En la Tabla 3.19 se actualiza la comparación entre los diferentes tipos de LED que se realizó en la Tabla 3.18 del aparatado anterior, incluyendo el consumo de potencia para una resistencia R1 de mayor valor, concretamente de 220 Ω.

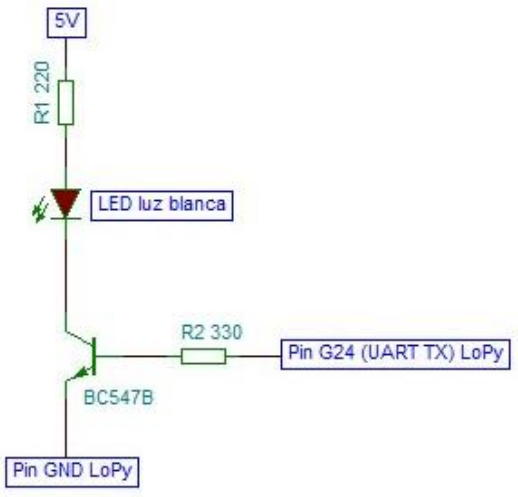


Figura 3.49: Circuito transmisor utilizado para el enlace 2.

En la Tabla 3.19 se puede observar cómo disminuye el consumo de potencia para los tres tipos de LED. También se aprecia cómo se mantiene el orden de consumo de potencia entre los diferentes tipos de LED, manteniéndose el LED de luz blanca como el que menos consumo de potencia presenta, y el LED infrarrojo como el que más posee.

En cuanto al receptor, en el apartado anterior se pudo establecer una comunicación de 16 cm de distancia entre transmisor y receptor, pero no se alcanzó una comunicación de 50 cm debido a las características intrínsecas del fotodiodo PIN empleado. En este apartado se presenta el mismo circuito receptor que se empleó en el enlace 1, dado que es el circuito receptor con el que se han realizado las pruebas del transmisor del enlace 2, y se garantiza la comunicación para los deseados 50 cm de distancia.

Tipo LED	**Corriente pico LED (osciloscopio) R1=120Ω**	**Corriente pico LED (osciloscopio) R1=220Ω**
LED luz blanca	90 mA	18 mA
LED luz roja	140 mA	23 mA

LED infrarrojo	160 mA	27 mA

Tabla 3.19: Estudio comparativo entre tipos de LED y su consumo en diferentes configuraciones.

En el enlace 2, el receptor se encuentra en un nodo estación base, y por lo tanto no tiene restricción en consumo de potencia como si pudiera tener en el enlace 1. Por tanto, si el receptor utilizado en el enlace 1 es suficientemente apto para las condiciones de un receptor móvil, para el caso de un receptor en un nodo estación base también lo es. En definitiva, el circuito receptor utilizado en el enlace 2 es el que se muestra en la Figura 3.50.

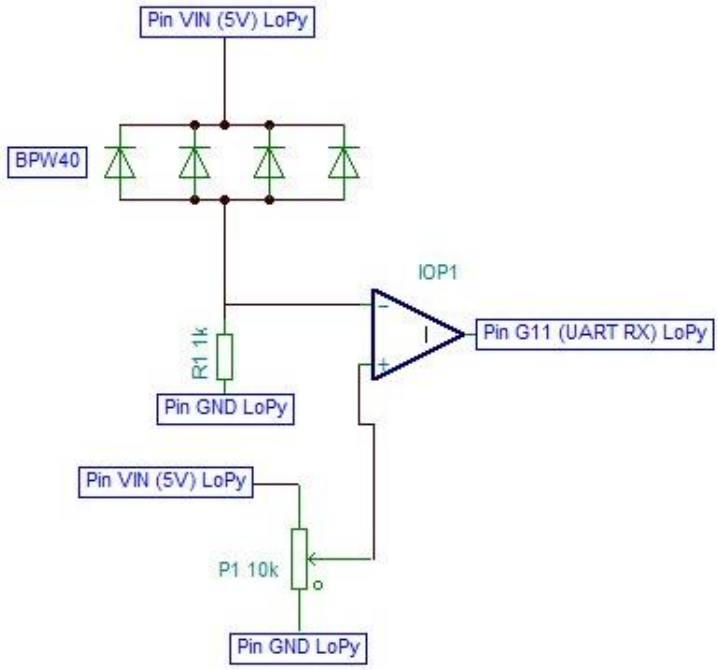


Figura 3.50: Circuito receptor utilizado en el enlace 2.

En el enlace 2, al haber introducido en la documentación del enlace 1 los conceptos *hardware* y *software* de la integración de los circuitos transmisor y receptor con el dispositivo LoPy, se puede repetir el conexionado establecido en el enlace 1, pero con los circuitos transmisor y receptor propios del enlace 2. Es decir, se conectan en ambos circuitos la alimentación con la batería integrada en el dispositivo LoPy o bien con el pin 'Vin' y 'GND' de la *Expansion Board*. En el caso del transmisor se conecta el pin 'G24' de la *Expansion Board* a la entrada del circuito transmisor, es decir en la base del transistor. En el caso del receptor se conecta la salida del comparador con el pin 'G11' de la *Expansion Board*.

Para la prueba de funcionamiento se ha implementado la misma funcionalidad *software* que en el caso del enlace 1 para los dispositivos LoPy que actúan como transmisor y como receptor. El código implementado en el transmisor es el de la Figura 3.51, y el código implementado en el receptor es el de la Figura 3.52.

```
from machine import UART
import time
import binascii
import pycom

pycom.heartbeat(False)
print('Main start transmisor')
uart = UART(1, baudrate=9600, pins=('P3','P4'))
uart.init(9600, bits=8, parity=None, stop=1) # init with given parameters
buf= bytearray(5)
buf[0]=0x0A
buf[1]=0x0B
buf[2]=0x0C
buf[3]=0x0D
buf[4]=0xF2
while True:
# this uses the UART_1 default pins for TXD and RXD (``P3`` and ``P4``)
   uart.write(buf)
   print('0x{0:02x}'.format(buf[0]))
   print('0x{0:02x}'.format(buf[1]))
   print('0x{0:02x}'.format(buf[2]))
   print('0x{0:02x}'.format(buf[3]))
   print('0x{0:02x}'.format(buf[4]))
```

Figura 3.51: Codificación de la funcionalidad del transmisor en el enlace 2.

```
from machine import UART
import time
import binascii
import pycom

pycom.heartbeat(False)
print('Main start receptor')
uart = UART(1, baudrate=9600, pins=('P3','P4'))
uart.init(9600, bits=8, parity=None, stop=1) # init with given parameters
buf= bytearray(5)
while True:
# this uses the UART_1 default pins for TXD and RXD (``P3`` and ``P4``)
   uart.readinto(buf) # read 1 byte
   print('0x{0:02x}'.format(buf[0]))
   print('0x{0:02x}'.format(buf[1]))
   print('0x{0:02x}'.format(buf[2]))
   print('0x{0:02x}'.format(buf[3]))
   print('0x{0:02x}'.format(buf[4]))
```

Figura 3.52: Codificación de la funcionalidad del receptor en el enlace 2.

En la Figura 3.53 se muestra el resultado de la correcta comunicación *VLC* entre transmisor y receptor en el enlace 2. Se trata de la respuesta por pantalla que se obtiene en el receptor. En la Figura 3.53 se observa cómo los datos enviados por el circuito transmisor, que se pueden ver en la Figura 3.51, se leen en la Figura 3.53.

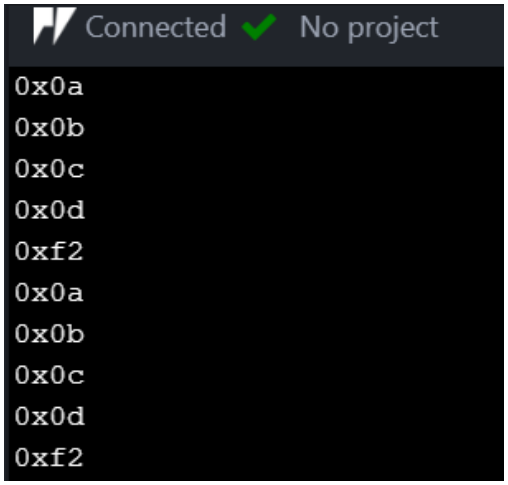


Figura 3.53: Respuesta obtenida en el receptor del enlace 2.

3.2.3 Presentación y análisis de los circuitos desarrollados

3.2.3.1 Enlace 1

Como transmisor *VLC* del enlace 1 se ha desarrollado el circuito de la Figura 3.54, mientras que como receptor *VLC* del enlace 1 se ha desarrollado el circuito mostrado en la Figura 3.55. El consumo de potencia para el circuito transmisor *VLC* propuesto es de 58.33 mA, como valor de corriente pico. En la Figura 3.56 se muestra el montaje completo de este enlace.

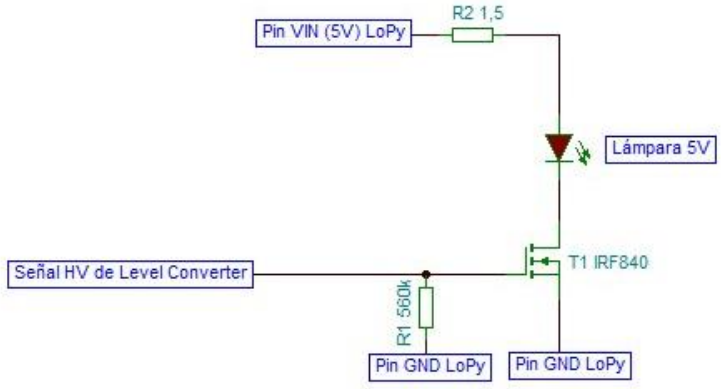


Figura 3.54: Circuito transmisor VLC para el enlace 1.

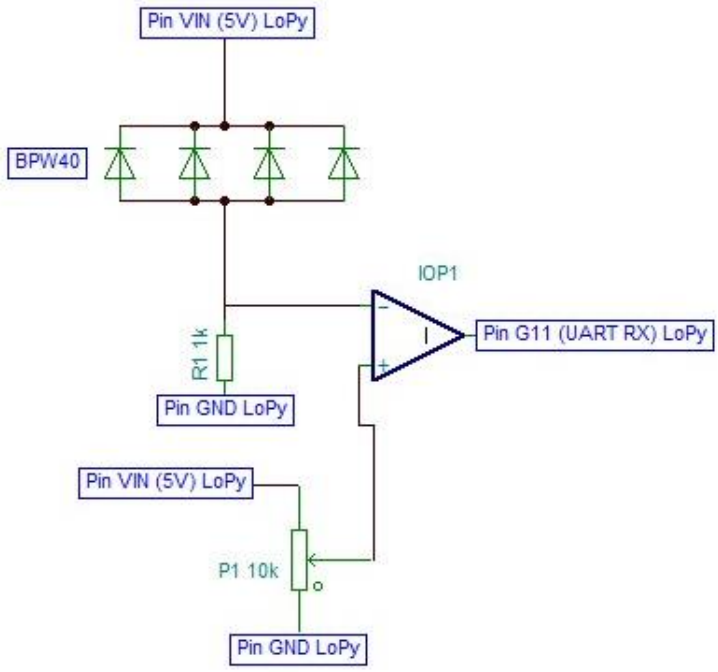


Figura 3.55: Circuito receptor VLC para el enlace 1.

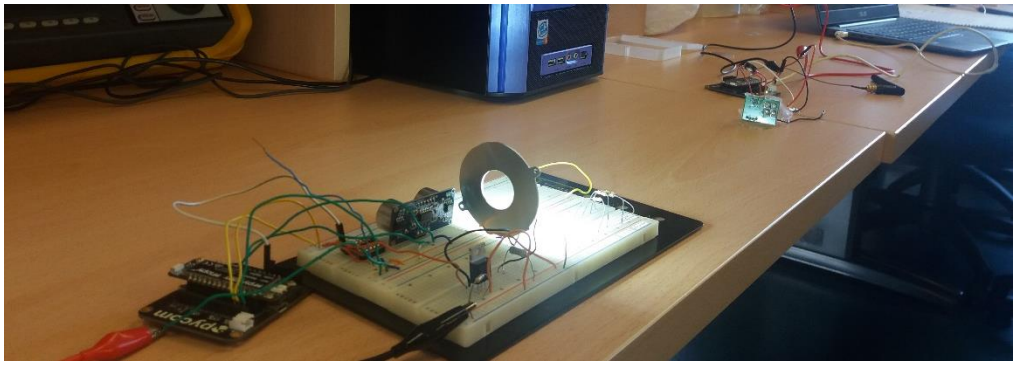


Figura 3.56: Montaje completo del enlace 1.

3.2.3.2 Enlace 2

Como transmisor *VLC* del enlace 2 se ha desarrollado el circuito de la Figura 3.57, mientras que como receptor *VLC* del enlace 2 se ha seleccionado el circuito mostrado en la Figura 3.55, que coincide con el receptor utilizado en el enlace 1. El consumo de potencia para el circuito transmisor *VLC* propuesto es de 18 mA como valor de corriente pico. En la Figura 3.58 se muestra el montaje completo del enlace 2.

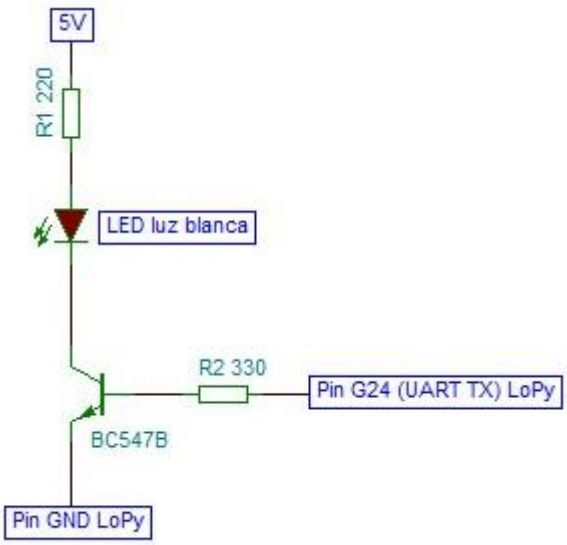


Figura 3.57: Circuito transmisor VLC para el enlace 2.

Los circuitos propuestos para los enlaces 1 y 2 cumplen con el objetivo de alcance de comunicación establecido en 50 cm. Las señales recibidas son suficientemente adecuadas para poder transmitirlas a través de la UART hacia los dispositivos LoPy correspondientes y procesar la información recibida desde estos. Con la presentación de estos circuitos y el análisis de consumo de potencia para cada circuito transmisor, concluye el presente capítulo de comunicación *VLC*.

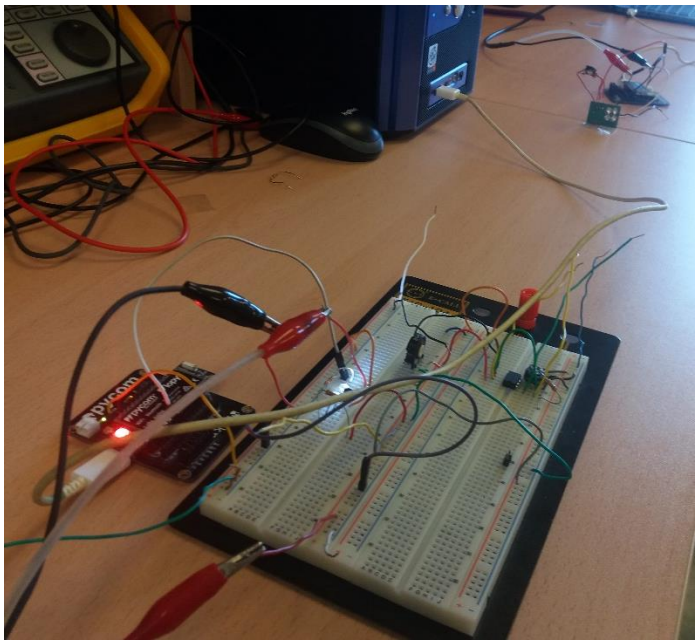


Figura 3.58: Montaje completo del enlace 2.

Capítulo 4. BLE

4.1 *Bluetooth Low Energy*: Descripción

En dispositivos IoT que se alimentan de baterías o de pilas (fuentes de energía agotables) es fundamental mantener una política de comunicación de bajo consumo de potencia, tanto en la frecuencia en la que se realizan las interacciones, como en el consumo energético de cada interacción. A partir de esta premisa, en muchas aplicaciones de IoT se utilizan tecnologías alternativas a las más conocidas y populares, que a la vez están pensadas para equipos con una autonomía, en cuánto a independencia energética, mucho mayor que la de la mayoría de los dispositivos IoT. Dependiendo del alcance requerido en cada aplicación IoT se opta por escoger una tecnología u otra. Para el caso de una aplicación de corto alcance o de área personal una de las tecnologías más empleadas es *Bluetooth Low Energy* (*BLE*).

4.1.1 Definición

Bluetooth Low Energy (*BLE*) es una tecnología de comunicación *PAN* (*Personal Area Network*), es decir, posee un alcance de pocos metros, estableciéndose 10 metros como el límite aproximado que permite una comunicación fiable. Si se compara con *Bluetooth clásico*, *BLE* aporta un menor consumo de potencia, lo cual supone una ventaja de uso para los dispositivos que se alimenten por una batería o una pila, como pueden ser los dispositivos utilizados en IoT. Además, esta disminución en el consumo de potencia no

causa una reducción de alcance, sino que se mantiene aproximadamente igual al de la versión clásica.

4.1.2 Antecedentes

En el año 2007, Nokia y Bluetooth SIG (*Special Interest Group*) llegaron a un acuerdo para que la tecnología de bajo consumo de potencia que había desarrollado Nokia, Wibree, perteneciera a la especificación *Bluetooth*. Dos años más tarde, en el año 2009, Bluetooth SIG denominó la tecnología de bajo consumo de potencia como una nueva versión, la versión *Bluetooth 4.0*. Ya en el año 2010, adoptó el nombre de *BLE*. En el año 2011 siguió el desarrollo de esta tecnología y se presentaron los conceptos *Smart Bluetooth Ready* y *Smart Bluetooth*. Posteriormente, en el año 2016, Bluetooth SIG anunció la nueva versión *Bluetooth 5.0*, afirmando que se duplica la velocidad, se obtiene mayor fiabilidad y alcance. Finalmente, en el año 2019 se presentó la versión 5.1, que cuenta con la gran novedad del estudio de posicionamiento de los dispositivos conectados. La precisión del posicionamiento es inferior a la de *Global Positioning System* (GPS), o por supuesto que la de *Global Navigation Satellite System* (GNSS), pero se estima que su precisión es de aproximadamente cuatro centímetros.

Las versiones de *Bluetooth Low Energy* son, actualmente, *Bluetooth 4* y *Bluetooth 5*, incluyendo las versiones intermedias. La principal ventaja que introdujo *Bluetooth 4* fue el propio concepto de *Low Energy*, enfocado para aplicaciones como las del ámbito IoT. Por otra parte, *Bluetooth 5* busca mejorar las prestaciones de la versión anterior con un incremento en la tasa de datos y un mayor alcance. También se conoce a la tecnología *BLE* como *Bluetooth Smart*. Bajo esta terminología se pueden clasificar los dispositivos que cuentan con esta tecnología de la siguiente manera:

- *Bluetooth Smart Ready*: Este tipo de dispositivos permite compatibilidad con *Bluetooth clásico* y con *BLE*. Mayoritariamente se trata de ordenadores y *smartphones*.

- *Bluetooth Smart*: Este tipo de dispositivos usa exclusivamente la tecnología *BLE*. Mayoritariamente se trata de dispositivos IoT.

4.1.3 Impacto actual de *BLE*

BLE es una tecnología cuya implantación va en aumento según recoge el documento *Bluetooth Market Study 2019* [62] que genera cada año *Bluetooth SIG* (*Special Interest Group*). Este estudio indica que en los próximos años el número de dispositivos *Bluetooth* seguirá incrementándose como ha hecho en los últimos años, estimándose para 2023 la cifra de 5.4 mil millones de dispositivos *Bluetooth*, tal y como se recoge en la Figura 4.1 [62]. De esa cifra, que incluye todos los tipos de dispositivos *Bluetooth*, se estima también, que el 90% de esos dispositivos sean de bajo consumo de energía, es decir, BLE.

BLE cuenta con numerosas aplicaciones y casos de uso, la mayoría de ellas, por las características técnicas que posee, orientadas a aplicaciones de IoT.

Es, además, una de las primeras tecnologías abiertas de comunicación inalámbrica, que ofrece comunicación entre dispositivos móviles u ordenadores y otros dispositivos, comúnmente utilizados como nodos sensoriales (con alimentación de pila de botón o batería), por lo que existe un amplio estado del arte sobre la comunicación entre aplicaciones móviles o de escritorio y redes de dispositivos sensoriales.

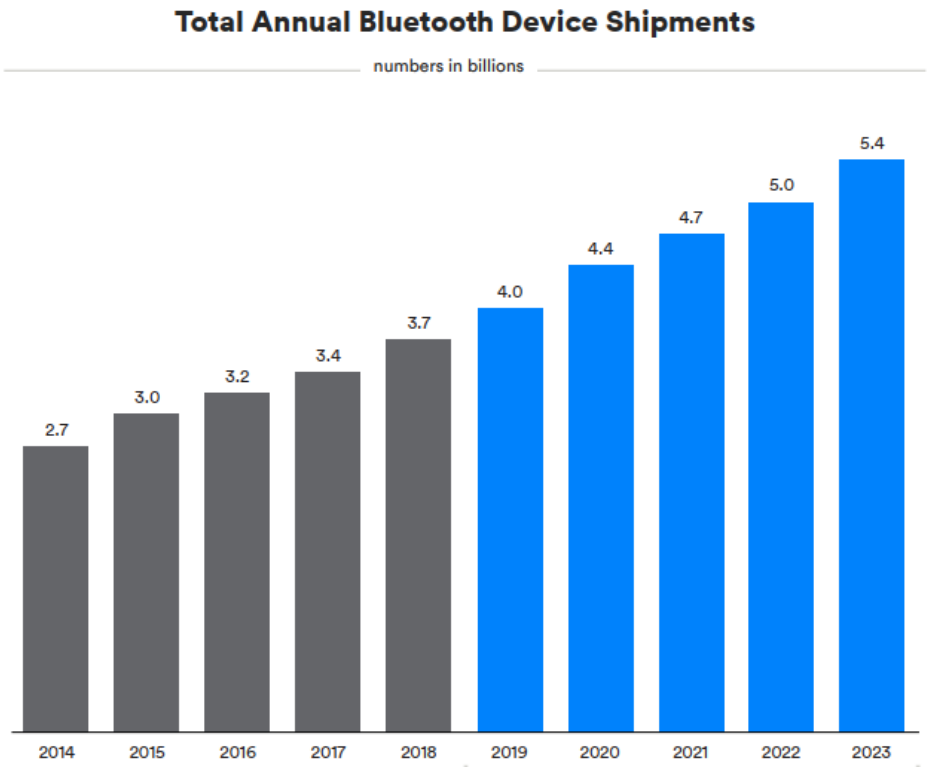


Figura 4.1: Número de dispositivos Bluetooth.

BLE, por lo tanto, es una versión de *Bluetooth clásico* que no puede usarse para aplicaciones de IoT. *Bluetooth clásico* cuenta con determinadas características que no permiten su uso en aplicaciones de IoT, en concreto en dispositivos que se alimentan por una fuente de energía limitada. Estas características son:

- Tasa binaria que excede las necesidades de una aplicación IoT.
- Mantenimiento de la conexión activa constante.
- Consumo de energía excesivo y no optimizado.

Con respecto a sus características técnicas, BLE opera en la banda de 2.4 GHz, al igual que la versión de *Bluetooth clásico*. Pero se diferencia de este en que el protocolo está optimizado para evitar colisiones y de esta forma reducir el consumo de energía.

BLE, como se ha explicado anteriormente, está pensado para aplicaciones de IoT y por lo tanto también se espera que funcione correctamente sobre redes de sensores inalámbricas. La versión *Bluetooth* 4.0 no incluye esta funcionalidad, pero la versión 4.1 si dispone de ella.

Cuenta con una tasa de transferencia teórica de 32 Mb/s, y según la referencia [63], en un caso libre de errores, la tasa no supera los Mb/s, sino que se mantiene en el orden de Kb/s. No obstante, para aplicaciones IoT de bajo consumo de potencia, en las que la información se transmite periódicamente, la tasa de errores no suele convertirse en un factor determinante.

4.1.4 Capa física: Modulación

Bluetooth opera en la banda ISM (*Industrial, Scientific and Medical Bands*), concretamente en la frecuencia de 2.4 GHz. Aunque las últimas versiones de *Bluetooth* como la versión 5.0 permiten comunicación en una banda menos saturada, como la de 5 GHz. La banda de 2.4 GHz es una banda libre, por lo que para evitar interferencias y pérdida de información, se utiliza la modulación de espectro ensanchado por salto de frecuencia (FHSS). Esta modulación consiste en transmitir información durante un período, de tiempo sobre una

frecuencia, y pasado ese período, transmitir la información por otra frecuencia distinta. La secuencia de salto de frecuencias es pseudoaleatoria. Esta secuencia debe ser conocida, tanto por el transmisor como por el receptor. El número de canales de la banda de 2.4GHz para *Bluetooth* es de 79 con un ancho de banda de 1 MHz [64], mientras que para *BLE* son de 40 con un ancho de banda de 2MHz. Cada canal se divide en subcanales a los que van asociados una única frecuencia, y según la señal de reloj del dispositivo maestro *Bluetooth,* se va realizando el salto en frecuencia de un subcanal a otro. En la Figura 4.2 [65] se muestra un ejemplo de funcionamiento de esta modulación. En la función de la izquierda de la Figura 4.2 se observa la asignación de frecuencias a cada subcanal del espectro. En la función de la derecha de la Figura 4.2 se muestra el uso de los canales con el salto en frecuencia de forma periódica.

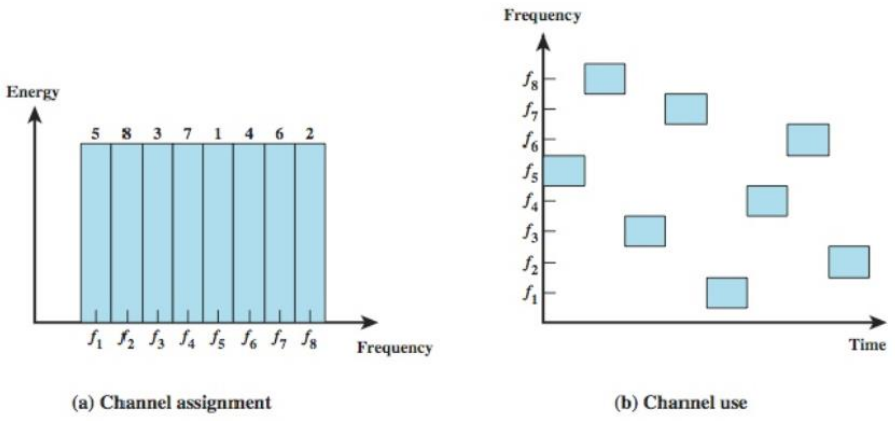


Figura 4.2: Modulación FHSS.

En la Figura 4.3 [64] se muestra el espectro utilizado por *BLE*, con 37 canales de datos y 3 canales de establecimiento de conexión y búsqueda de dispositivos.

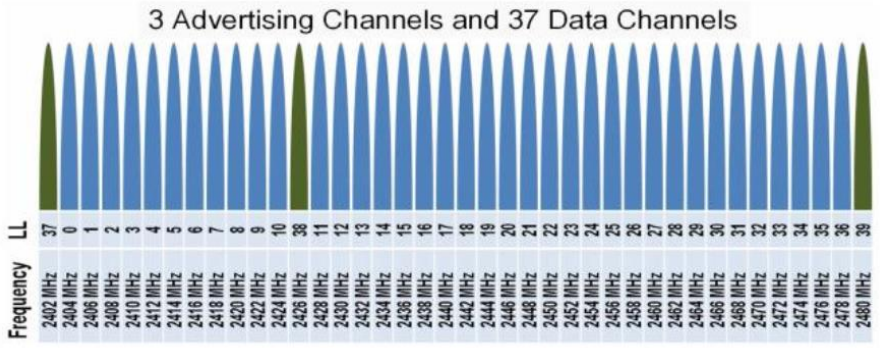


Figura 4.3: Espectro utilizado por BLE.

4.1.5 Protocolos de comunicación

En una comunicación *BLE* se definen generalmente dos roles. Los dispositivos pueden comportarse como *Peripheral* o como *Central*. Los dispositivos *Peripheral* suelen ser aquellos en los que se busca un bajo consumo de potencia porque son alimentados por una pila de botón o una batería, pretendiéndose que este tipo de fuente de alimentación dure lo máximo posible y de esta forma evitar la sustitución temprana de esta. Por otra parte, el dispositivo *Central* normalmente no cuenta con la restricción de un bajo consumo de potencia, y suele verse asociado a dispositivos como *smartphones* o *tablets*.

Con el fin de conocer el protocolo de *BLE*, se debe introducir el concepto de *Generic Access Profile* (GAP) [66]. GAP es un perfil que se encarga de controlar las conexiones y permite que el dispositivos sea público a la vista de los demás. Existen dos formas de transmitir información entre el dispositivo *Peripheral* y el dispositivo *Central* que permite el perfil GAP. Estas dos transmisiones se denominan *Advertising Data Payload* y *Scan Response Payload*. Los dos paquetes pueden tener una longitud máxima de 31 bytes y ambos se utilizan para proporcionar información del dispositivo *Peripheral* al dispositivo *Central*. No obstante, solo *Advertising Data Payload* es de carácter obligatorio, mientras que *Scan Response Payload* es opcional y se envía bajo previa petición del dispositivo *Central*.

Advertising Data Payload se envía periódicamente desde el dispositivo *Peripheral* con el fin de asegurar la recepción por parte del nodo *Central*, mientras que *Scan Response Payload* se envía únicamente cuando se recibe una petición desde el nodo *Central* y suele contener información como el nombre del dispositivo o alguna característica especial definida por el fabricante [66].

Por lo tanto, el protocolo de envío de estos paquetes es como el que se representa en la Figura 4.4 [66].

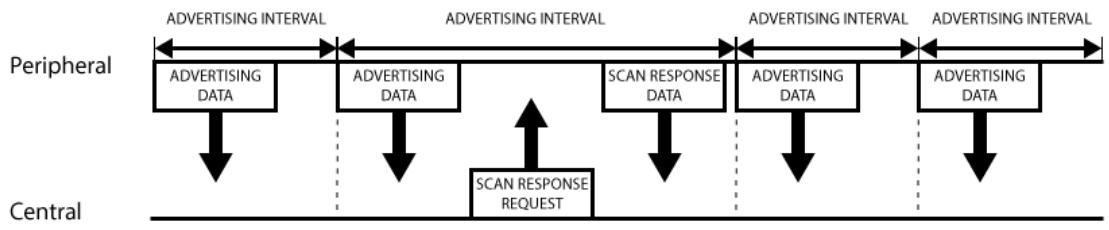


Figura 4.4: Protocolo de establecimiento de comunicación BLE.

En la Figura 4.4 se puede ver que periódicamente se envía el paquete *Advertising Data*, y este sólo se ve interrumpido por una petición *Scan Response* desde el nodo *Central* y su propia respuesta en el nodo *Peripheral*. El intervalo con el que se envían los paquetes *Advertising Data* es modificable según la aplicación. Si se desea un ahorro en el consumo de energía, se debe mantener un intervalo alto, mientras que si se busca establecer una conexión en el menor espacio de tiempo posible se debe mantener un intervalo corto.

Un concepto clave de la arquitectura de red utilizada en *BLE* es que para comunicar un dispositivo *Peripheral* con otro es necesario comunicarse previamente con un dispositivo *Central*.

Una vez que se ha establecido la conexión haciendo uso del perfil GAP, se puede hacer uso de otro perfil, *Generic Attribute Profile* (GATT). Este perfil define cómo dos dispositivos *BLE* pueden comunicarse a través de los Servicios y Características. GATT usa el protocolo ATT que almacena en una tabla los Servicios y Características en cuestión en entradas con identificadores de 16 bits.

Una característica importante de GATT es que una vez que se ha establecido una conexión entre un dispositivo *Peripheral* y un dispositivo *Central*, el *Peripheral* deja de anunciarse con el perfil GAP y los demás dispositivos ya no pueden verlo, y en consecuencia tampoco conectarse a él, al menos hasta que concluya la conexión establecida [66]. En resumen, en el perfil GATT las conexiones son exclusivas. Este comportamiento no es simétrico, es decir, el dispositivo *Central* si puede establecer más de una conexión con diferentes dispositivos *Peripheral*.

También se debe tener en cuenta que la única forma de mantener una comunicación bidireccional es estableciendo una conexión entre los dos dispositivos en cuestión.

Una comunicación bidireccional ofrecida por el perfil GATT es muy similar a la relación que existe en el modelo cliente servidor. En este caso, el rol del servidor lo ejecuta el dispositivo *Peripheral*, que tiene almacenados los Servicios y Características, así como los datos del protocolo ATT. Por otra parte, el nodo *Central* actúa como cliente y envía solicitudes al nodo *Peripheral*. El protocolo de comunicación que se sigue en el perfil GATT es el que se muestra en la Figura 4.5 [66]. En la Figura 4.5 se puede ver cómo el dispositivo *Peripheral* establece un intervalo de tiempo para la conexión con un nodo *Central*, y es en esos

espacios de tiempo cuando el nodo *Central* debe enviar una solicitud de datos al dispositivo *Peripheral*, y dando respuesta dentro de ese mismo intervalo. No obstante, en ocasiones, si el dispositivo *Central* se encuentra ocupado realizando otras tareas, no hace uso de el intervalo de tiempo establecido por el dispositivo *Peripheral*.

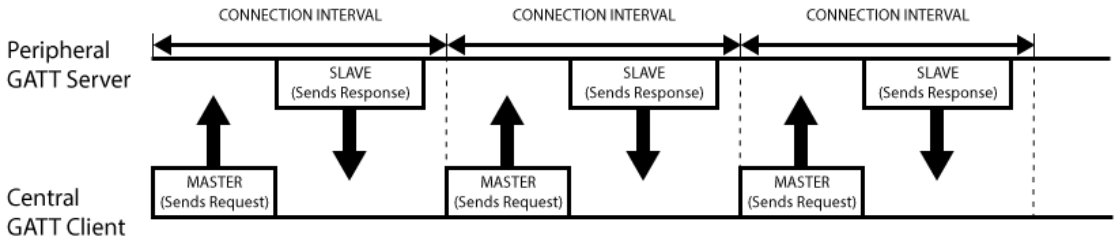


Figura 4.5: Protocolo de comunicación de perfil GATT.

En la Figura 4.6 se resume en una máquina de estados el funcionamiento de un dispositivo *Central* en una comunicación *BLE*. Mientras que en la Figura 4.7 se muestra la máquina de estados correspondiente a un dispositivo *Peripheral*.

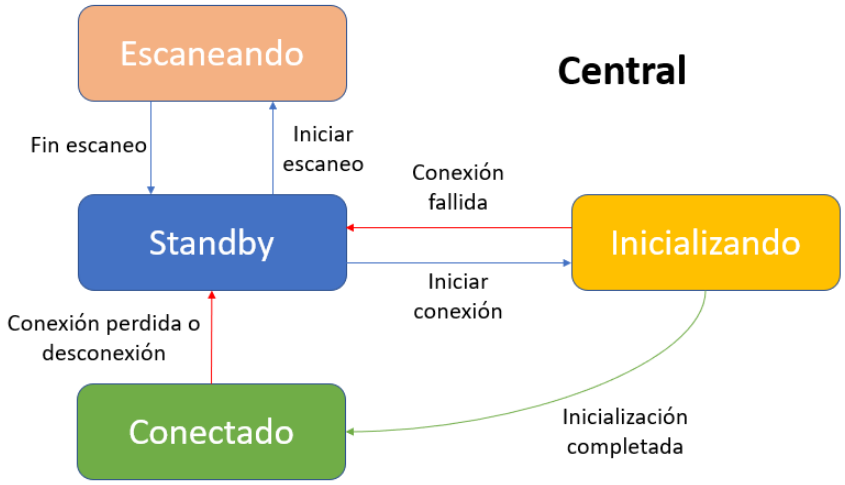


Figura 4.6: Máquina de estado de un dispositivo BLE Central.

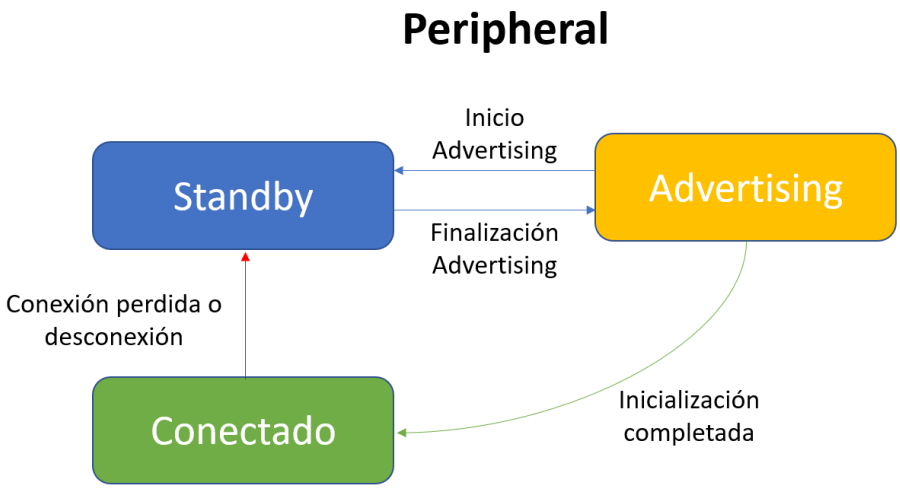


Figura 4.7: Máquina de estado de un dispositivo BLE Peripheral.

Dentro de un perfil GATT se definen Servicios, y a su vez, dentro de los Servicios se incluyen Características. La arquitectura descrita se puede visualizar en la Figura 4.8 [67].

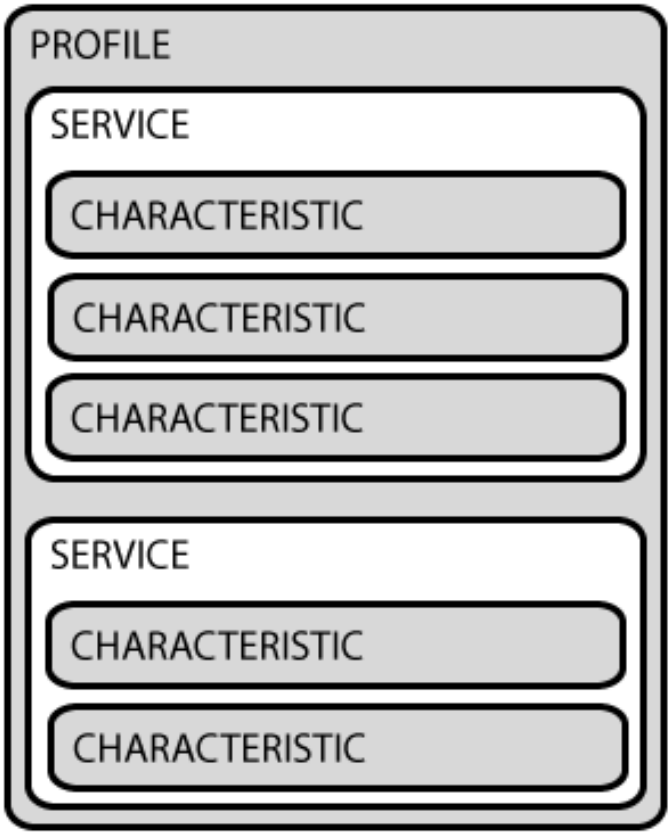


Figura 4.8: Arquitectura de perfil GATT.

El concepto de perfil viene de un conjunto de Servicios que ha sido especificado por Bluetooth SIG, o por el propio fabricante del dispositivo. En la web oficial de *Bluetooth* [67] se encuentra la lista de perfiles oficiales certificados por Bluetooth SIG.

Un perfil contiene uno o más Servicios, al igual que un Servicio puede tener una o más Características. Se puede diferenciar cada Servicio de los demás por un ID numérico único (UUID) de 16 bits en el caso de Servicios oficiales recogidos en la web oficial, o de 128 bits para Servicios personalizados. [66]

Bajando un nivel en la estructura GATT se encuentran las Características. Las Características contienen un único tipo de dato, y pueden tener un tamaño de identificador (UUID) de 16 o 128 bits, siguiendo el mismo criterio que con los Servicios. Existen Características ya definidas en la web oficial de *Bluetooth* [67]. El uso de Características y Servicios oficiales asegura la compatibilidad entre el *software* y el *hardware BLE*. Crear Características y Servicios propios dota de un mayor grado de libertad al desarrollador, pero puede provocar incompatibilidades. Generalmente, el formato de las Características suele incluir la utilización de los primeros 8 bits para indicar el tipo de dato (UINT8, UINT16…) y seguidamente se encuentran los datos en cuestión. Pueden ser Características de escritura o de solo lectura, y de esta forma se logra una comunicación bidireccional entre el nodo *Central* y el nodo *Peripheral*.

El protocolo que se ha descrito está orientado a conexión, pero hay dispositivos *Peripheral* que, en lugar de mantener una comunicación exclusiva con un nodo *Central*, simplemente trata de anunciar sus datos. Un dispositivo *Peripheral* puede transmitir datos a más de un dispositivo a la vez, denominándose este proceso *broadcast*. Esta transmisión, al no haberse establecido una conexión, debe realizarse a través del paquete *Advertising Data Payload*. En bastantes casos de uso, se usa el campo *Manufacturer Specific Data* para introducir la información que se desea enviar a los nodos *Central* que se encuentran dentro del alcance del dispositivo *Peripheral*. [66]

En la Figura 4.9 se muestra la topología que se debe seguir si se desean realizar transmisiones *broadcast* desde un dispositivo *Peripheral*, a uno o más de un dispositivo *Central*.

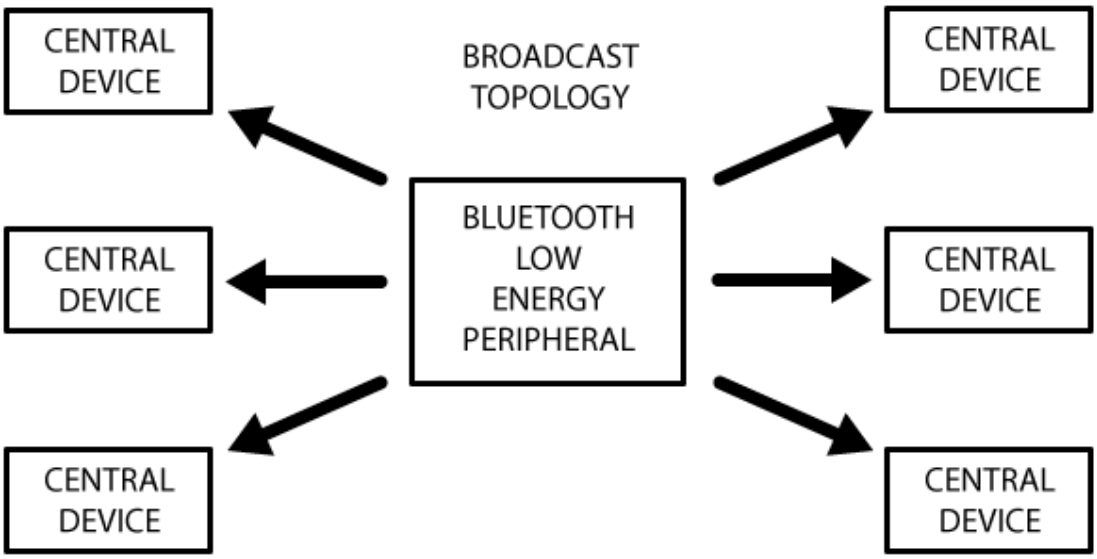


Figura 4.9: Topología broadcast.

4.2 Implementación de *BLE* en la plataforma HW/SW

A continuación, se describe la implementación de la comunicación basada en *Bluetooth Low Energy* (*BLE*) para la plataforma HW/SW propuesta en este TFM utilizando el dispositivo LoPy. Como se ha visto en la descripción teórica de esta tecnología, se requieren mínimamente dos roles para establecer una comunicación *BLE*, un rol de dispositivo *Central* y un rol de dispositivo *Peripheral*. En este caso concreto, se desea establecer una comunicación unidireccional entre dos dispositivos, por lo que el rol de nodo *Peripheral* lo interpreta el dispositivo transmisor, y el rol de nodo *Central* lo interpreta el dispositivo receptor.

4.2.1 Transmisor-*Peripheral*

El dispositivo transmisor, al funcionar como nodo *Peripheral* en la comunicación *BLE,* debe realizar un proceso de *advertisement* previo al envío de datos. Una vez que se haya enlazado el dispositivo transmisor con el receptor, se procede al envío de los datos mediante la estructura presentada en el apartado anterior, es decir, haciendo uso de Servicios y Características. Cada Servicio y Característica debe estar identificado con un *Universally Unique IDentifier* (UUID).

Se ha desarrollado en el dispositivo LoPy la funcionalidad de comunicación *BLE* del transmisor en el código que se muestra en la Figura 4.10. En la Figura 4.10 se importa la librería asociada a la comunicación *BLE*. Inicialmente se crea un objeto *Bluetooth* y se configura un proceso de *advertisement* con un nombre y un identificador.

```
1  from network import Bluetooth
2  import pycom
3
4  pycom.heartbeat(False)
5  bluetooth = Bluetooth()
6  bluetooth.set_advertisement(name='LoPy_Server', service_uuid=b'1234567890123456')
7
8  def conn_cb (bt_o):
9      events = bt_o.events()
10     if  events & Bluetooth.CLIENT_CONNECTED:
11         print("Client connected")
12         pycom.rgbled(0x007f00) # green
13     elif events & Bluetooth.CLIENT_DISCONNECTED:
14         print("Client disconnected")
15         pycom.rgbled(0x7f0000) # red
16
17 bluetooth.callback(trigger=Bluetooth.CLIENT_CONNECTED | Bluetooth.CLIENT_DISCONNECTED, handler=conn_cb)
18
19 bluetooth.advertise(True)
20
21 srv1 = bluetooth.service(uuid=b'1234567890123456', isprimary=True)
22
23 chr1 = srv1.characteristic(uuid=b'ab34567890123456', value=5)
24
25 char1_read_counter = 0
```

Figura 4.10: Codificación de la funcionalidad BLE del transmisor.

El siguiente paso es establecer la conexión entre el transmisor y el receptor. Para ello se define la función *conn_cb()*. En esta función se realiza una consulta de eventos *BLE* y se evalúa si el dispositivo está conectado o desconectado al nodo maestro. En caso de estar conectado se enciende el LED RGB del dispositivo LoPy en color verde, mientras que si no se encuentra conectado se enciende el LED en color rojo.

La llamada a la función *conn_cb()* se produce ante un evento en la comunicación *BLE*. En la línea 17 de la Figura 4.10 se llama a la función *conn_cb()* en caso de que se produzca este evento, haciendo uso de la función *callback()* del objeto *Bluetooth* creado anteriormente.

Una vez que se ha configurado la comunicación *BLE,* se puede iniciar el proceso de *advertisement* y enviar mediante Servicios y Características la información deseada. En este caso, se desea enviar el dato “5”. Para ello se introduce este dato en el campo *value* de la

Característica asociada a un determinado Servicio. Tanto el Servicio como la Característica deben estar identificadas con un UUID.

4.2.2 Receptor-*Central*

El dispositivo receptor, al funcionar como nodo *Central* en la comunicación *BLE,* debe realizar un proceso de escucha con el fin de recibir los *advertisements* que envía el transmisor. Una vez que se haya enlazado el dispositivo receptor con el transmisor, se procede a la recepción de los datos mediante la estructura presentada en el apartado anterior de descripción de *BLE*, es decir, haciendo uso de Servicios y Características. Cada Servicio y Característica debe estar identificado con un *Universally Unique IDentifier* (UUID) y por lo tanto en el proceso de lectura se leen Servicios y Características que tengan el UUID de interés para el receptor.

Se ha desarrollado la funcionalidad de comunicación *BLE* del receptor en el dispositivo LoPy a partir del código que se muestra en la Figura 4.11. En la Figura 4.11 se importa la librería asociada a la comunicación *BLE*. Inicialmente se crea un objeto *Bluetooth* y se realiza un escaneo. En este caso, el valor "-1" que se pasa como parámetro a la función de escaneo *start_scan(),* indica que se trata de un escaneo indefinido. En caso de que se desee acotar el tiempo de escaneo se pasa el valor por parámetro igualmente, pero con valores naturales.

Seguidamente se entra en un bucle infinito que adquiere los *advertisements* procedentes del transmisor. En este bucle se compara el *advertisement* recibido con el nombre esperado en este. En este caso, el nombre que identifica al *advertisement* enviado por el transmisor es "*LoPy_Server*". Si la comparación demuestra que coincide el nombre esperado con el recibido, se conecta con el dispositivo que ha enviado la información de *advertisement*, ya que se corresponde con el transmisor.

Tras realizar la conexión con el transmisor, se realiza una lectura de los Servicios que este presenta y se extraen las Características. Se muestra en el terminal cada Servicio y Característica, mostrando su UUID y en el caso de las Características, se muestra también su contenido.

Una vez concluida la lectura, se finaliza la conexión *BLE* y se evalúan nuevamente y de forma periódica nuevos *advertisements*.

```
from network import Bluetooth
import time
bt = Bluetooth()
bt.start_scan(-1)

while True:
  adv = bt.get_adv()
  if adv and bt.resolve_adv_data(adv.data, Bluetooth.ADV_NAME_CMPL) == 'LoPy_Server':
      try:
          conn = bt.connect(adv.mac)
          services = conn.services()
          print(services)
          for service in services:
              print(service.uuid())
              time.sleep(0.050)
              if type(service.uuid()) == bytes:
                  print('Reading chars from service = {}'.format(service.uuid()))
              else:
                  print('Reading chars from service = %x' % service.uuid())
              chars = service.characteristics()
              for char in chars:
                  if (char.properties() & Bluetooth.PROP_READ):
                      print('char {} value = {}'.format(char.uuid(), char.read()))
          conn.disconnect()
          break
      except:
          print("Error while connecting or reading from the BLE device")
          break
  else:
      time.sleep(0.050)
```

Figura 4.11: Codificación de la funcionalidad BLE del receptor.

4.2.3 Resultado de la comunicación

Una vez que se implementan las funcionalidades del transmisor *BLE* y del receptor *BLE,* se puede llevar a cabo una prueba de comunicación que valide el funcionamiento desarrollado. Para ello se carga el *firmware* del transmisor *BLE* en un dispositivo LoPy, y el *firmware* del receptor *BLE* en otro dispositivo LoPy. Como resultado de la comunicación, se muestra por pantalla la recepción realizada por parte del nodo receptor y en el que se pueden apreciar diferentes mensajes. Este resultado se muestra en la Figura 4.12 La mayoría de los mensajes muestran el UUID de Servicios o Características, pero en el círculo rojo de la Figura 4.12, se muestra el dato que realmente se deseaba enviar.

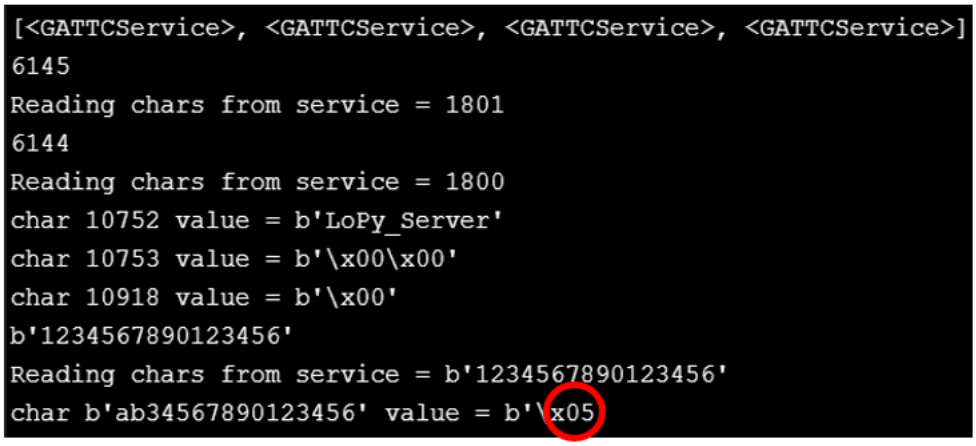


Figura 4.12: Resultado de la comunicación BLE.

Con esta prueba experimental, se considera validada la implementación de la comunicación basada en *Bluetooth Low Energy* (*BLE*) para la plataforma HW/SW propuesta en este TFM. De cara a su integración en el *firmware* completo de la plataforma genérica, se adecuará esta implementación a los requisitos exigidos para su correcto funcionamiento.

Capítulo 5. LoRa/LoRaWAN

5.1 *LoRa/LoRaWAN*: Descripción

LoRa/LoRaWAN es una tecnología que permite realizar comunicaciones de largo alcance, a la par que se mantiene un bajo consumo de potencia. Esta característica principal permite su uso en aplicaciones IoT que demanden comunicaciones de largo alcance, pero que mantengan la restricción propia de IoT de un bajo consumo de potencia.

Esta tecnología, de tipo *Low Power Wide Area Network* (LPWAN), es una buena alternativa a la comunicación celular, que presenta un elevado consumo, y en muchos casos esto es inasumible en aplicaciones IoT. Por otro lado, supone un aumento del alcance de la comunicación con respecto a tecnologías que sí respetan el bajo consumo de potencia, pero que no alcanzan grandes longitudes de comunicación.

Actualmente, estas redes de tipo LPWAN se encuentran en pleno auge y se siguen desarrollando mejoras que agilicen su implementación en soluciones *Smart City*. Las tecnologías de este tipo que más relevancia tienen hoy en día son *Sigfox* y *LoRa/LoRaWAN*. Más adelante se explicará el motivo de la selección de esta última.

En concreto, en este TFM se hace uso de *LoRa/LoRaWAN*, y por ello, a continuación, se presenta detalladamente esta tecnología de comunicación.

5.1.1 Introducción

LoRa y *LoRaWAN* son especificaciones de redes de elevado alcance y reducido consumo de potencia. Están concebidas para utilizarse en el ámbito de IoT. Esta tecnología fue ideada por la *startup* francesa Cycleo, y posteriormente fue adquirida por la empresa SemTech, quien la desarrolló y patentó. Actualmente siguen manteniendo la patente, pero es la fundación *LoRa Alliance* la que gestiona su desarrollo y su evolución [68]. *LoRaWAN* presenta una red del tipo LPWAN (*Low Power Wide Area Network*).

Se debe diferenciar desde un primer momento qué es *LoRa* y qué es *LoRaWAN*, porque no son lo mismo, habiendo puntualizaciones que se deben comentar, como, por ejemplo, el nivel o capa en el que desarrollan su comunicación, o qué misión tiene cada una dentro de una red *LoRa/LoRaWAN*.

Por una parte, *LoRa* solo trabaja a nivel de la capa de enlace y su uso está destinado a la comunicación *Peer to Peer* (P2P) entre diferentes nodos, mientras que en *LoRaWAN* se añade la capa de red o Internet, a partir de la cual se permite la conexión con cualquier estación base que se encuentre conectada a la nube [69].

Una definición que ayuda a entender la arquitectura que utiliza *LoRa,* y el papel que desempeñan *LoRa* y *LoRaWAN* dentro de ella, es la que se muestra en la referencia [70]: "*LoRaWAN* se encarga de unir diferentes dispositivos *LoRa* gestionando sus canales y parámetros de conexión: canal, ancho de banda, cifrado de datos, etc." Por tanto, se puede afirmar que los dispositivos *LoRa* son los elementos que permiten incluir la electrónica, entendida como sensores o actuadores finales, como elementos de la red *LoRaWAN*. Dicho con otras palabras, *LoRa* representa la conexión física entre los nodos de la red, y *LoRaWAN* es la propia red que interconecta y gestiona dichos nodos, teniendo en cuenta diferentes parámetros propios de una red de comunicación.

5.1.2 Comparación con otros estándares

Para comparar diferentes tecnologías de comunicación es necesario enmarcar bien para qué especificaciones se usa un estándar u otro. La primera comparativa que se realiza en

este documento pretende ubicar la tecnología en términos de alcance y consumo de potencia. En la Figura 5.1 [71] se muestran los estándares más usados internacionalmente junto con la tecnología *LoRa/LoRaWAN*.

Como se puede observar en la Figura 5.1, dentro de las tecnologías de corto alcance se tienen, entre otras, *WiFi* y *Bluetooth*, siendo la primera una red de área local, y la segunda de área personal, pero en comparación con las otras dos que se muestran en la gráfica, se consideran de bajo alcance. Entre las tecnologías de largo alcance se pueden observar las redes móviles y *LoRaWAN*.

Las redes móviles y *WiFi* soportan mayor transferencia de datos y sus velocidades de transmisión son superiores a las de *Bluetooth* y *LoRa/LoRaWAN*, pero por esta misma razón consumen mucha más potencia. En el caso de *LoRa/LoRaWAN* se consigue el equilibrio perfecto entre una red de largo alcance y de bajo consumo de potencia. El alcance y la tasa binaria dependen de factores como el valor del parámetro *Spreading Factor* (SF) o el ancho de banda utilizado, por lo tanto, no son valores fijos, pero aun así superan en alcance a los estándares de *WiFi* y *Bluetooth*.

El reducido consumo de potencia y su elevado alcance, hace que *LoRa/LoRaWAN* se postule como una tecnología puntera en el ámbito de IoT y *Smart City*.

Sin embargo, dentro de las redes LPWAN existen otros estándares que están cobrando fuerza en el mercado e intentan posicionarse, siendo los principales los siguientes:

- *Sigfox*: Junto con *LoRa/LoRaWAN* es una de las tecnologías de comunicación LPWAN más importantes. Tiene un coste de $1 por dispositivo y año conectado, de forma que es propietaria y presta servicios bajo licencia de uso. Toda su tecnología es privada, y se basa en ancho de banda ultra-estrecho. Un dispositivo de *Sigfox* puede enviar hasta 140 mensajes al día de 12 bytes como máximo. Su mapa de cobertura es mayor que el de *LoRa/LoRaWAN*, ya que cuenta con conectividad en toda Francia, la mayoría de los territorios de España, y algunos puntos de otros países como Reino Unido. *LoRa/LoRaWAN* solo tiene cobertura en Francia por ahora.

- Cat NB-IoT: Representa una evolución o adaptación de la tecnología *Long Term Evolution* (LTE) al ámbito de IoT. Cuenta con un ancho de banda y un consumo superior a los de *LoRa/LoRaWAN* y *Sigfox*. Aún está por ver su futuro para aplicaciones de requerimiento energéticos muy reducidos. [72]

Figura 5.1: Comparativa por alcance y consumo de potencia.

En líneas generales, las diferentes ventajas de *LoRaWAN* frente al resto de redes del tipo LPWAN son [73].

- La tasa binaria puede ir desde 0,3 kbps hasta 5 kbps para un ancho de banda de 125 KHz, lo que permite un ahorro en la vida útil de la batería y una duración de transmisión superior.

- La comunicación es bidireccional e ilimitada.

- Seguridad a nivel de red, de aplicación y de dispositivo.

- Localización sin uso de GPS al usar multilateración *Time Difference of Arrival* (TDoA).

- Se ofrece una gran variedad de *gateways* según el uso que se vaya a dar a la red.

- Posibilidad de crear redes públicas o privadas.

- Utilización de *Adaptative Data Rate* (*ADR*) para la optimización de la red, permitiendo que puedan incluirse más nodos, disminuyendo la tasa binaria de otros nodos.

5.1.3 *LoRa*

LoRa representa la capa física o la modulación inalámbrica utilizada para crear una conexión de comunicación de larga distancia [74]. Esta capa física se basa en la modulación de espectro ensanchado. El procedimiento que se sigue para esta modulación consiste en primer lugar en codificar la señal con una secuencia de alta frecuencia, propagando la señal con un ancho de banda superior, y reduciendo con ello el consumo de energía, además de aumentar la resistencia a interferencias electromagnéticas, tal y como se indica en la Figura 5.2 [75].

LoRa soporta seis valores de factores de propagación *Spreading Factor* (SF7 - SF12) y tres anchos de banda diferentes (125 kHz, 250 kHz, 500 kHz). Más adelante, en el apartado de características técnicas, se indicarán las implicaciones que conlleva la elección del factor de propagación, y también que al ancho de banda más utilizado sea el de 125 KHz. Los factores de propagación y anchos de banda permitidos están definidos por las agencias reguladoras, en el caso europeo, por ETSI *(European Telecommunications Standards Institute)*.

Debido a la tecnología de espectro de propagación utilizada, los mensajes con diferentes velocidades de datos son ortogonales y no interfieren unos con otros.

El esquema *LoRa* se basa en una variante de *Spread Spectrum Modulation* (SSM) denominada modulación de *Chirp Spread Spectrum* (CSS). CSS codifica los datos con un

chirp, que es esencialmente una señal sinusoidal de frecuencia modulada en banda ancha que aumenta o disminuye con el tiempo [75].

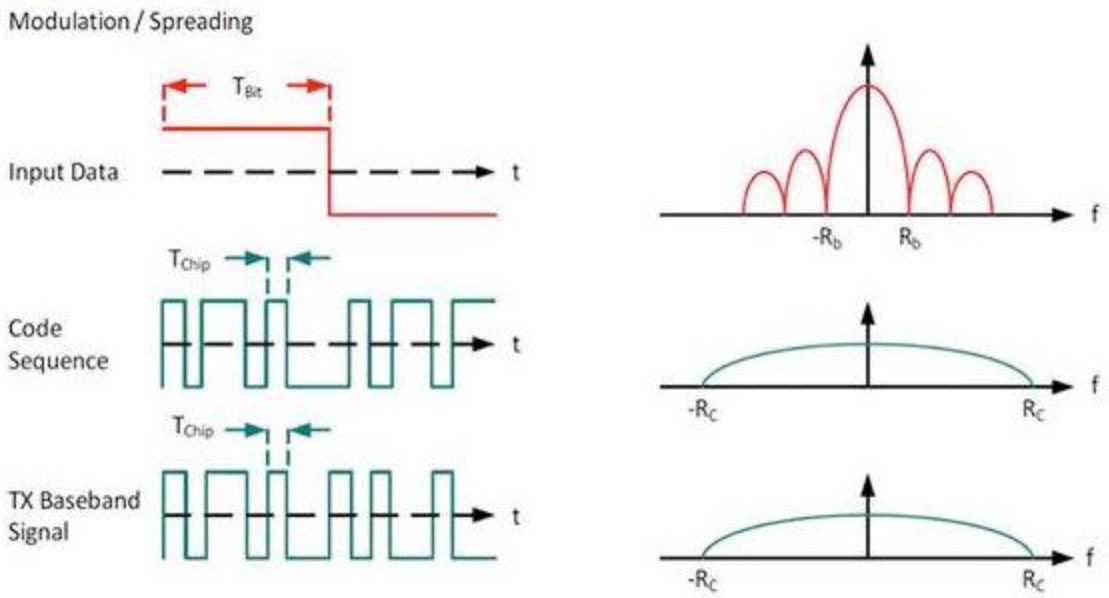


Figura 5.2: Modulación de espectro ensanchado.

En definitiva, *LoRa* es una comunicación al nivel de la capa física y no atiende a parámetros propios de la capa de red, como en el caso de *LoRaWAN*.

5.1.4 *LoRaWAN*

LoRaWAN define el protocolo de comunicación y la arquitectura del sistema para la red mientras que, a nivel físico, *LoRa* permite crear la conexión para una comunicación a larga distancia. Las tareas de *LoRaWAN* resultan determinantes para el buen funcionamiento de la red, pues en función del protocolo usado y la arquitectura de la red adoptada, se determina la duración de vida de las baterías de los nodos, la capacidad de comunicación de la red, la calidad del servicio, la seguridad, y la variedad de aplicaciones que puede ofrecer la red.

En otras palabras, la especificación *LoRaWAN* define la capa de control de acceso al medio (MAC) para la red del tipo LPWAN. Según el modelo *Open Systems Interconnection* (OSI) de *International Organization for Standardization* (ISO), esta capa se ubica por encima de la capa física que define *LoRa*. Esta capa de acceso al medio se desarrolla de forma abierta

por la entidad *LoRa-Alliance*, a diferencia de *LoRa*, que es una capa física propiedad de la empresa SemTech.

5.1.5 Topología

Una vez se han introducido *LoRa* y *LoRaWAN*, ya se pueden exponer los campos que se deben incluir en cualquier estudio sobre una determinada tecnología de comunicación, como, por ejemplo, la topología utilizada para la transmisión y recepción de datos.

En el caso de la red *LoRaWAN*, existe una topología claramente definida por los desarrolladores de esta tecnología, la disposición en estrella. Esta topología se divide en dos componentes fundamentales, los nodos finales y el nodo central. Este nodo central, que también se denomina *gateway*, es el punto principal de la red, pues es donde confluyen todos los datos procedentes de los distintos nodos finales, y además también es donde se redirige la información que se desea enviar a cada nodo.

Por tanto, en una red *LoRa/LoRaWAN*, todos los nodos finales se conectan punto a punto con este *gateway*, y esta comunicación la realizan bajo la tecnología que ofrece *LoRa*. Por otra parte, el *gateway* es el que proporciona a la nube o a la plataforma de todos los datos recibidos por parte de los nodos sensoriales, y allí es donde se procesan los datos y se ejecutan acciones que se trasladan a los nodos actuadores, de la misma manera en que se transmitió la información desde los sensores, pero en este caso se tiene el flujo de datos en sentido inverso. La comunicación entre el *gateway* y la nube escapa a la comunicación *LoRa/LoRaWAN*, y lo normal es encontrar comunicaciones basadas en TCP/IP.

Sin embargo, la red *LoRaWAN* no tiene que limitarse a varios nodos asociados a un *gateway,* pudiendo escalar esta topología a una red de redes en estrella o una red en estrella extendida. Esta topología también recibe el nombre de estrella de estrellas y en ella, algunos nodos de una subred en estrella se conectan como un nodo más de otra subred, lo cual puede repetirse más veces en la estructura de la red. En la Figura 5.3 se representa este tipo de red de redes de forma gráfica.

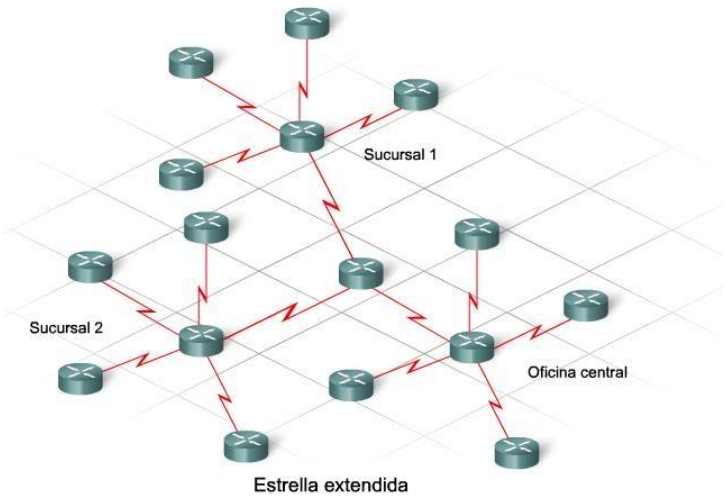


Figura 5.3: Topología: "Estrella de estrellas".

En lo que respecta a las comunicaciones que esta topología puede soportar, se tienen diferentes niveles de comunicación. Resumiendo lo comentado anteriormente, y añadiendo algún apunte más, los distintos tipos de comunicaciones que los elementos de la red pueden soportar, son los siguientes:

- Los nodos finales de una misma subred en estrella pueden comunicarse entre ellos, a través de un *gateway* usando *LoRa*.

- Pueden comunicarse con la nube a través de un *gateway*, siguiendo primero una comunicación nodo-*gateway* tipo *LoRa*, y posteriormente *gateway*-nube usando Ethernet, 3G/4G/5G u otras tecnologías con acceso a Internet.

- Los nodos finales de distintas subredes en estrella pueden comunicarse entre sí, a través de los *gateways* de sus respectivas subredes.

- Una utilidad muy importante que permite *LoRa/LoRaWAN* es la de realizar multidifusión. La multidifusión suele utilizarse para actualizar nodos o distribuir mensajes masivos de emergencia.

En resumen, la mayor parte de las comunicaciones en este tipo de redes son bidireccionales, aunque hay casos excepcionales como el de la multidifusión.

Otro aspecto que se debe estudiar es el tipo de nodos finales que se pueden tener en una red *LoRaWAN*, y su forma de interactuar con ella. Existen tres tipos de nodos, todos ellos bidireccionales, ordenados según el consumo de potencia y su interacción con la red.

- Clase A: Son los nodos de más baja potencia y por cada subida de datos desde el nodo final, le siguen dos ventanas de recepción para la descarga de datos. Por lo tanto, el *slot* programado para la transmisión se adecua a las propias necesidades de comunicación, con una pequeña variación basada en un tiempo aleatorio (sistema de slot del protocolo ALOHA). Esta clase de nodo final es ideal para aquellos casos donde un sensor debe enviar un dato puntual cada cierto tiempo, y que no suele funcionar como receptor de una comunicación. En caso de que se desee enviar información a este nodo final desde el servidor, se debe esperar a que este realice una transmisión, y que tras completarla, se habiliten las ventanas de recepción. En la Figura 5.4 [76] se muestra el eje temporal que sigue un nodo final de clase A. Se debe apuntar también que en este tipo de nodos finales no existe restricción en términos de latencia y que cualquier dispositivo puede mostrarse como de clase A.

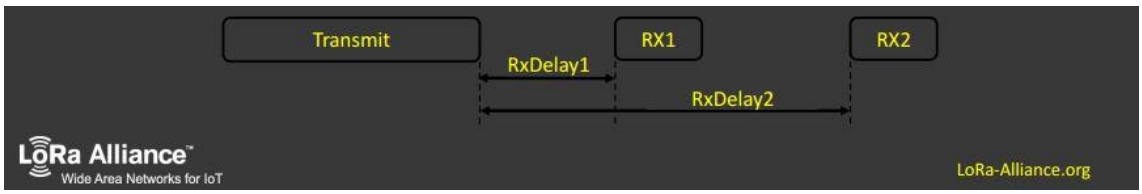


Figura 5.4: Estudio temporal de un nodo final de clase A.

- Clase B: Para los nodos finales de tipo B, aparte de las ventanas de recepción que presentan los de clase A, se añaden ventanas de recepción en momentos determinados previamente establecidos. Cuando ha llegado el momento establecido, desde la puerta de enlace o *gateway* se envía un *Beacon* (un paquete que se suele usar para mantener activo un determinado receptor). Con esto lo que se consigue, aparte de habilitar la ventana de recepción del nodo final, es que el servidor sepa que el nodo final está en

modo “escucha”. En la Figura 5.5 [76] se muestra el eje temporal que sigue un nodo final de clase B. En este caso la latencia de descarga es controlada.

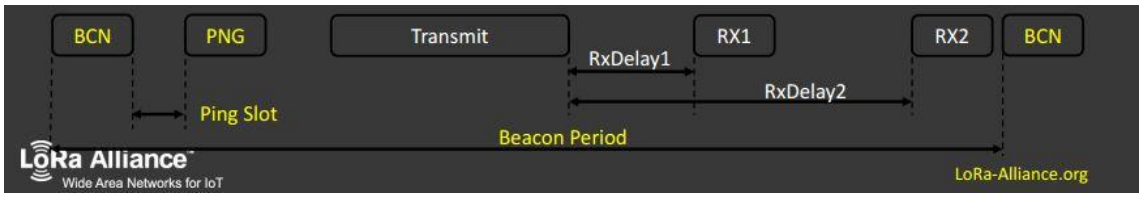


Figura 5.5: Estudio temporal de un nodo final de clase B.

- Clase C: Para este tipo de nodos, la ventana de recepción está prácticamente siempre abierta, salvo cuando se está transmitiendo, al estar en un modo de transmisión semidúplex, en el que la comunicación es bidireccional, pero no se puede transmitir y recibir información simultáneamente. El inconveniente de este tipo de nodos es que se consume mucha más potencia que en los otros dos casos. En la Figura 5.6 [76] se muestra el eje temporal que sigue un nodo final de clase C. En este último tipo no hay latencia, y solo lo soportan los dispositivos que puedan mantenerse a la escucha continuamente.

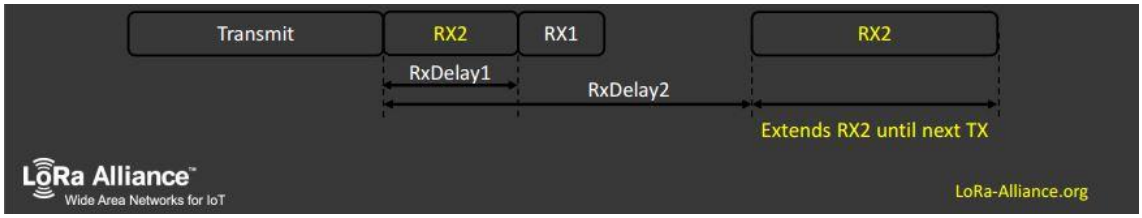


Figura 5.6: Estudio temporal de un nodo final de clase C.

En la Figura 5.7 [76] se muestra a nivel de capas la arquitectura que sigue *LoRa*, y en ella se puede ver cómo desde la capa en la que trabaja *LoRaWAN* se atiende de forma diferente a los nodos finales de diferentes clases. En la propia Figura 5.7 se enuncia esta atención a esos nodos finales como MAC *options*.

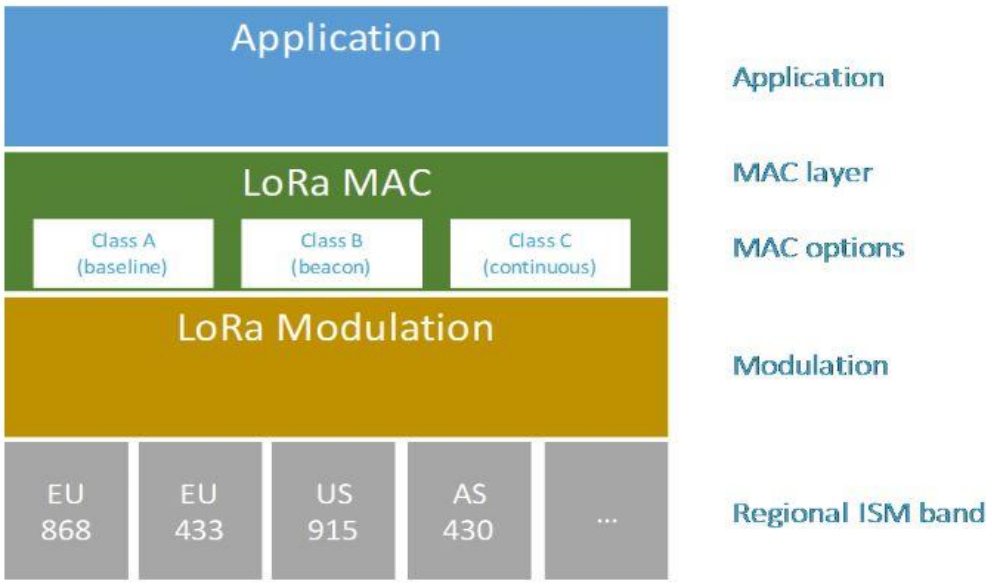


Figura 5.7: Arquitectura LoRa/LoRaWAN.

5.1.6 Características técnicas

En este apartado se presentarán las características técnicas de la tecnología *LoRa/LoRaWAN*.

5.1.6.1 Modulación

LoRa representa la capa física o la modulación inalámbrica utilizada con el fin de crear conexiones para comunicaciones de larga distancia. También se conoce que muchos de los sistemas inalámbricos usados hasta el momento se basan en la modulación *Frequency Shift Keying* (FSK) porque resulta una modulación muy eficiente para obtener un bajo consumo. Sin embargo, *LoRa* se basa en la modulación CSS que mantiene la característica de bajo consumo que aporta la modulación FSK, pero amplía significativamente la distancia de comunicación. Durante décadas ya se usaba esta modulación en aplicaciones militares o espaciales debido a la larga distancia que es capaz de soportar y la robustez que posee ante interferencias. La diferencia es que *LoRa* es la primera implementación de esta tecnología comercializada al público en general, y con un coste reducido. No obstante, en el nivel físico de la comunicación también se puede usar la modulación FSK en este tipo de redes *LoRaWAN*.

En el eje temporal para esta modulación se muestra un *chirp* como el de la gráfica mostrada en la parte derecha de la Figura 5.8. Como se observa, el *chirp* incrementa su frecuencia en

el tiempo, por lo que se estaría ante un *Up-chirp*, mientras que, en el caso contrario, si se decrementa la frecuencia a medida que avanza el tiempo, se estaría ante un *Down-chirp*. Estos dos casos se muestran en la Figura 5.9. También se puede obtener el valor nulo para el *chirp*. La diferencia entre la frecuencia inicial del *chirp* y la frecuencia final es una buena aproximación del ancho de banda B del pulso del *chirp* en el espectro, como se representa en la Figura 5.8 [77], esta vez en la gráfica izquierda de la ilustración.

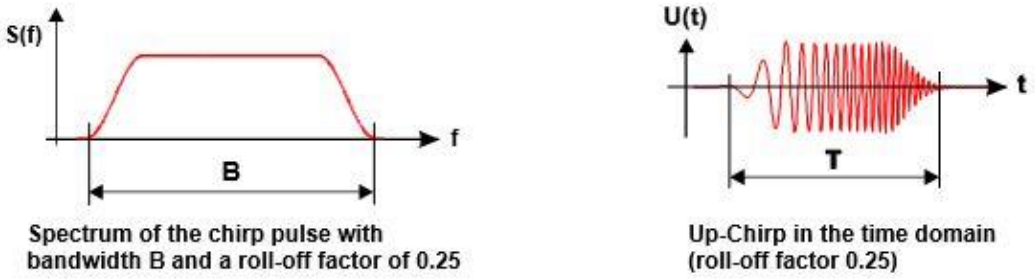


Figura 5.8: Chirp en el espectro y en el eje temporal.

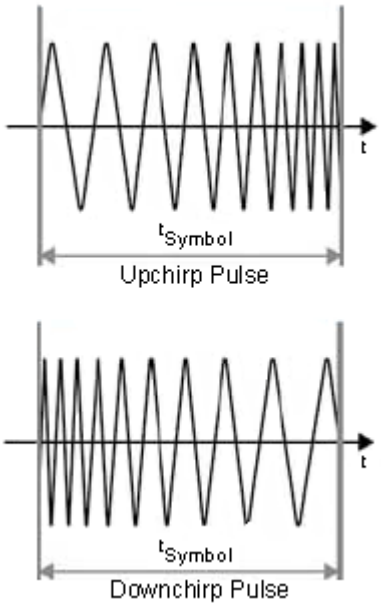


Figura 5.9: Pulsos Upchirp y Downchirp.

Sobre esta modulación hay que comentar algunos aspectos que resultan imprescindibles para poder entender por qué se utiliza en *LoRa*.

Así, los beneficios de CSS se manifiestan de forma clara cuando el ancho de banda *(B)* del pulso del *chirp* es mucho mayor que la tasa binaria *(R)*. Es decir, interesa que *B >> R*. De aquí procede el interés del ensanchamiento en la frecuencia.

Por otro lado, la duración del pulso del *chirp* se puede elegir libremente, y así conseguir un producto *B·T* elevado que aporte robustez en la señal [77]. Es una modulación que favorece los sistemas en los que se varía la distancia y la tasa binaria rápidamente. Especialmente para aquellos sistemas de banda ancha o de ultra banda ancha, en las que el ancho de banda es muy superior a la tasa binaria.

La combinación de *LoRa* con esta modulación ofrece una solución para cubrir la demanda de una comunicación de baja potencia, de largo alcance y asequible económicamente.

5.1.6.2 Estudio de parámetros asociados

La comunicación entre los nodos finales y el *gateway* se lleva a cabo en diferentes canales de frecuencia y con diferentes tasas binarias. Posteriormente, se verá que se debe llegar a un compromiso entre la tasa binaria, el alcance de la transmisión, y la duración del mensaje [71], pero por ahora se indica que esta tasa binaria es cambiante. *LoRa* puede mantener tasas binarias desde 300 bps hasta 5 kbps para un ancho de banda de 125 KHz. El ancho de banda puede llegar a ser de 250KHz, y para este caso la tasa binaria máxima alcanza los 11 kbps. También se puede escoger un ancho de banda de 500 KHz, si bien es el menos usado. En este rango, *LoRa* se adecua perfectamente a los márgenes en los que la modulación CSS proporciona sus mejores funcionalidades, ya que el ancho de banda es muy superior al régimen binario.

La red *LoRa* usa el modelo *Adaptative Data Rate* (*ADR*) para obtener un equilibrio entre la optimización de la vida útil de la batería y las prestaciones que el sistema es capaz de proporcionar. *ADR* organiza las tasas binarias individuales de cada nodo que se va a comunicar y las salidas en los canales de frecuencia que se van a usar.

Teóricamente los nodos finales pueden transmitir en cualquier momento y por cualquier canal que esté disponible, siempre y cuando respete ciertas restricciones. Una de estas restricciones es que los nodos finales deben cambiar de canal de transmisión de forma

pseudo-aleatoria en cada transmisión que realicen. El beneficio que se obtiene al cumplir este requisito es que se consigue una variedad en frecuencias, y con esto, un sistema más robusto ante interferencias.

Otra de las restricciones se centra en la legislación local del lugar en el que se vaya a usar esta tecnología. Para el caso de este TFM, se debe considerar la restricción aplicable al marco europeo. *LoRaWAN* usa franja libre del espectro, concretamente la banda *Industrial Scientific and Medical bands* (ISM). ETSI ha limitado el acceso a las bandas de 868 MHz y 433 MHz. *LoRaWAN* trabaja en concreto en la banda de 868 MHz ISM de Europa, y en esta franja el nodo final debe respetar el ciclo de trabajo máximo en referencia a la sub-banda utilizada y las regulaciones locales (1% para nodos finales y 10% para *gateways*) [74]. Otra restricción está relacionada con la potencia transmitida por los nodos, y que no debe exceder los 14 dBm o los 25mW.

En la referencia [74] se define el *ADR* como el proceso por el cual la red ordena a cada nodo la tasa binaria que debe adoptar, que será un valor suficiente para cumplir con la transmisión requerida. Se espera que, en un futuro no muy lejano, también se indique a cada nodo con qué potencia ha de transmitir.

Para poder ahondar sobre el funcionamiento de *ADR* es necesario introducir el concepto de *Spreading Factor* (SF). El *Spreading Factor* es un valor que indica la relación entre la tasa binaria y la tasa de símbolos, o lo que es lo mismo del período de bit y el período de símbolo. En la Ecuación 5.1 se muestra dicha expresión.

$$SF = \frac{T_{Símbolo}}{T_{Bit}} = \frac{R_{Bit}}{R_{Símbolo}} \qquad (5.1)$$

Aprovechando que se ha expuesto ya el concepto de *Spreading Factor*, en las siguientes líneas se mostrarán, mediante expresiones, todas las variables que intervienen en el cálculo de parámetros que se usarán en el desarrollo de la funcionalidad del nodo a la plataforma HW/SW propuesta en este TFM. Como se muestra en la Ecuación 5.2, el ancho de banda es la inversa del período del *chirp* [78]. Por otra parte, el período de símbolo guarda relación con los dos parámetros anteriores, tal y como se muestra en la Ecuación 5.3. Por último, en la Ecuación 5.4 se adjunta la expresión para el cálculo de la tasa binaria.

$$BW = \frac{1}{T_{Chirp}} \qquad (5.2)$$

$$T_{Símbolo} = 2^{SF} \cdot T_{Chirp} \qquad (5.3)$$

$$R_{Bit} = SF \cdot R_{Símbolo} = SF \cdot \frac{1}{T_{Símbolo}} = \left(\frac{SF}{2^{SF}}\right) \cdot BW \qquad (5.4)$$

En la Figura 5.10 [78] se muestra un ejemplo sobre la relación de todos estos parámetros. De esta manera, se aprecia la importancia de determinar el valor correcto de SF según la tasa de bits necesaria para transmitir un dato determinado. En esta figura se aprecia cómo la tasa de bits aumenta cuando se disminuye el *Spreading Factor*.

		$T_{Symbol} = 2^{SF}/BW$		ohne FEC	4/5 FEC (CR=1)	
BW / kHz	SF	T_{symbol} / ms	R_{symbol} / Hz	R_{bit} / bps	R_{bit} / bps	Sensitivity
125	7	1.024	976.56	6835.94	5468	S_{ref} = -125 dBm
125	8	2.048	488.28	3906.25	3125	- 2.4 dB
125	9	4.096	244.14	2197.27	1757	- 4.9 dB
125	10	8.192	122.07	1220.70	976	- 7.5 dB
125	11	16.384	61.04	671.39	537	- 10 dB
125	12	32.768	30.52	366.21	292	- 12.7 dB

Figura 5.10: Ejemplo para el estudio de los parámetros de transmisión.

Además, en la Figura 5.10 se puede apreciar cómo mejora la sensibilidad a medida que se aumenta el SF. En el ejemplo se adjunta la tasa binaria para cuando no hay corrección de errores, y también para cuando se usa *Forward Error Correction* (FEC) como método de detección y corrección de errores.

En la Figura 5.11 [71] se muestra otra relación con más parámetros que varían en función del SF escogido, como el caso del ya comentado *bitrate*, el alcance y el tiempo de transmisión.

Spreading factor (at 125 kHz)	Bitrate	Range (indicative value, depending on propagation conditions)	Time on Air (ms) For 10 Bytes app payload
SF7	5470 bps	2 km	56 ms
SF8	3125 bps	4 km	100 ms
SF9	1760 bps	6 km	200 ms
SF10	980 bps	8 km	370 ms
SF11	440 bps	11 km	740 ms
SF12	290 bps	14 km	1400 ms
(with coding rate 4/5 ; bandwidth 125Khz ; Packet Error Rate (PER): 1%)			

Figura 5.11: Realación SF-BitRate-Alcance-Tiempo de transmisión.

5.1.6.3 Estructura de paquetes

5.1.6.3.a Paquete físico: LoRa

LoRa emplea dos tipos de formato de paquete: explícito e implícito. El modo explícito incluye una cabecera que contiene información acerca del número de bytes, codificación de error, y si se usa o no Código de Redundancia Cíclica (CRC) para la detección de errores. El modo implícito, por el contrario, no contiene cabecera. A continuación, en la Figura 5.12 [79] se muestra el formato del paquete *LoRa,* que consta principalmente de tres partes fundamentales: preámbulo, cabecera opcional y área de datos.

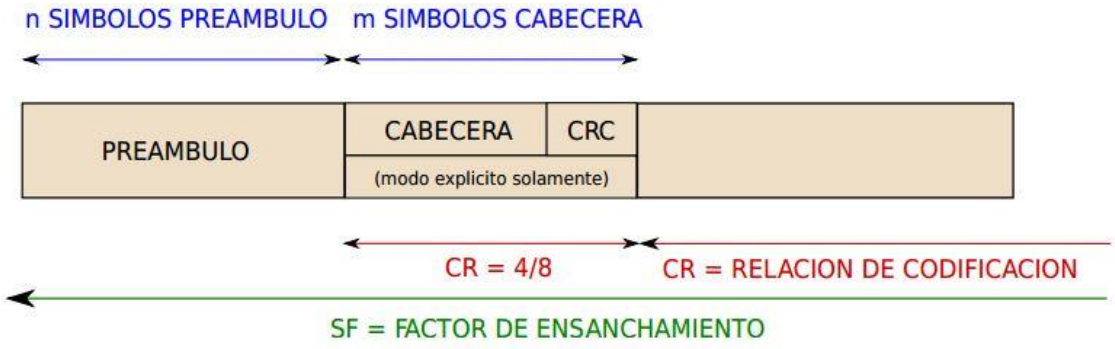


Figura 5.12: Formato de paquete LoRa.

El objetivo del preámbulo es sincronizar al receptor con el flujo de datos entrante. Por defecto, la longitud del paquete es de 12 símbolos de longitud, pero este valor es modificable mediante el registro *PreambleLength,* que permite configurar su valor desde 6 a 65535, obteniendo así la posibilidad de tener una longitud de símbolos desde 6+4 hasta 65535+4, dónde se alcanza el límite y se entiende que se ha llegado a la sobrecarga fija de los datos del preámbulo. Como se puede apreciar por el límite máximo que acepta este registro, se puede determinar un campo de preámbulo de longitud amplia.

Un apunte importante es que, tanto en el receptor como en el transmisor, la configuración de la longitud del preámbulo debe ser idéntica, pues el receptor realiza un proceso de detección de preámbulo que se reinicia periódicamente, y manteniendo la misma longitud se facilita este proceso. En caso de que no se conozca la longitud, o esta varíe, se recomienda desde la empresa que desarrolla *LoRa*, SemTech, que se programe la longitud de preámbulo del receptor con el mayor valor posible. [79]

Para seleccionar un tipo de paquete implícito o explícito se modifica el valor del bit *ImplicitHeaderModeOn* que se encuentra en el registro *RegModemConfig1* [79]. Por defecto se tiene el tipo de paquete explícito.

La cabecera en el modo explícito proporciona información sobre diferentes aspectos del área de datos, como la longitud del área de datos (en bytes), el código de corrección de errores, y la presencia o no de un código de verificación CRC opcional de 16 bits. La cabecera se transmite con un código de corrección máximo, de 8/4, y la misma cabecera contiene su propio código de verificación CRC para desechar cabeceras inválidas. Para la inclusión del campo CRC se debe activar el bit *RxPayloadCrcOn* en el registro *RegModemConfig1* en el transmisor, mientras que en el receptor una vez se ha recibido el área de datos, se debe revisar el valor del bit *CrcOnPayload* en el registro *RegHopChannel*. Si el bit *CrcOnPayload* se encuentra al valor "1", el usuario deberá revisar el *flag* IRQ *PayloadCrcError* para asegurarse que el CRC es válido. Si el bit *CrcOnPayload* se encuentra al valor "0", el área de datos no lleva CRC, por lo que el *flag* IRQ*PayloadCrcError* no se debería activar aún en el caso de que el área de datos presentase errores. [79]

El uso de la cabecera implícita se justifica en los casos en las que interesa reducir el tiempo de transmisión. En este tipo de formato de paquete, el área de datos, la codificación de

error, y la presencia de código de verificación CRC, son fijos y conocidos previamente. Para que esta información sea conocida hay que configurarla manualmente en el transmisor y en el receptor.

Existe un caso en el que solo se puede usar este formato de paquete, y es cuando el valor del *Spreading Factor* es 6. La activación del CRC en la cabecera implícita es fija, debiendo estar activado siempre el bit *RxPayloadCrcOn* en el registro *RegModemConfig1*, tanto en el transmisor como en el receptor [79].

Respecto al área de datos, este es un campo de longitud variable que contiene los datos codificados con el error especificado en la cabecera explícita, o en el registro de configuración en caso de cabecera implícita, siendo la verificación CRC opcional.

Cuando se utilizan valores de *Spreading Factor* elevados, la duración de transmisión de un paquete es bastante alargada, y para estos casos existe un bit que al activarlo mejora la robustez de la transmisión mediante variaciones en la frecuencia durante la transmisión y recepción del paquete. Este bit es *LowDataRateOptimize* y su uso es recomendable para velocidades de transmisión bajas, siendo obligatorio cuando la duración de un símbolo excede los 16 ms. Si se usa este bit, hay que configurarlo tanto en el transmisor como en el receptor.

Con el objetivo de calcular la duración de la transmisión de un mensaje, se presentan a continuación diferentes expresiones que permiten obtener la duración de cada campo, y por tanto, la duración del paquete completo. Lo primero que se debe indicar es que la duración de un símbolo es inversamente proporcional a la tasa de símbolos, tal y como se muestra en la Ecuación 5.5 [79].

$$T_S = \frac{1}{R_S} \qquad (5.5)$$

La Ecuación 5.6 muestra la duración del preámbulo en función del número de la longitud de este campo. Dicha longitud se puede extraer de los registros *RegPreambleMsb* y *RegPreambleLsb* [79].

$$T_{Preámbulo} = (n_{Preámbulo} + 4{,}25) \cdot T_S \qquad (5.6)$$

Por otro lado, la duración del área de datos viene determinada por la Ecuación 5.7 [79].

$$T_{Datos} = n_{Datos} \cdot T_S \qquad (5.7),$$

donde n_{Datos} es el número de símbolos del área de datos y se calcula a partir de la Ecuación 5.8 [79].

$$n_{Datos} = 8 + max\left(ceil\left[\frac{8PL - 4SF + 28 + 16CRC - 20IH}{4(SF - 2DE)}\right](CR + 4), 0\right) \qquad (5.8),$$

en la que:

- *PL* es el número de bytes del área de datos: [1 – 255].
- *SF* es el Spreading Factor: [6 – 12].
- *IH=0* cuando la cabecera está activada, y IH = 1 cuando no lo está.
- *DE=1* cuando *LowDataRateOptimize*=1, y DE=0 cuando el bit *LowDataRateOptimize* no está activado.
- *CR* es el *coding rate*: [Adquiere el valor 1 cuando es 4/5, y 4 cuando es 4/8].

Finalmente, se muestra en la Ecuación 5.9 [79] la duración total del paquete, que es la suma de la duración del preámbulo y de la duración del área de datos.

$$T_{Paquete} = T_{Preámbulo} \cdot T_{Datos} \qquad (3.9)$$

5.1.6.3.b Paquete MAC: *LoRaWAN*

LoRaWAN usa el formato de paquete físico que se indicó en el apartado *Paquete Físico*: *LoRa*, si bien existen ciertas puntualizaciones que se deben comentar, como que en comunicación ascendente en *LoRaWAN* la inclusión de la cabecera y del código de verificación CRC son obligatorios. Por lo tanto, se imposibilita directamente la opción de usar un *Spreading Factor* de valor 6, pues como se explicó en el apartado anterior, existe un caso en el que solo es posible usar el formato de paquete implícito, y es cuando el valor del *Spreading Factor* es 6. Es decir, con SF=6 no se incluye cabecera, y esto va en contra de los requisitos del paquete de *LoRaWAN*. Otro apunte es que los paquetes descendentes

llevan cabecera, pero no CRC. El formato del paquete *LoRaWAN* se muestra en la Figura 5.13 [80].

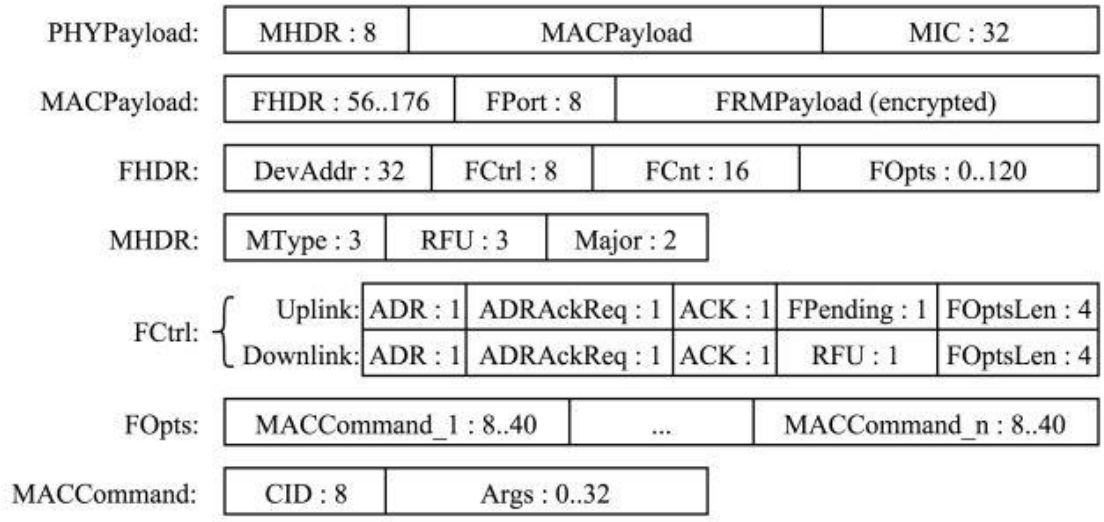


Figura 5.13: Formato de paquete LoRaWAN.

De forma estructurada y con el formato de paquete físico *LoRa* presente, en la Figura 5.14 se muestra el orden de encapsulado del mensaje [81].

A continuación, se procede a indicar el significado de cada campo de las tramas que forman el paquete.

- *Frame Header*:
    - *DevAddr*: Es la dirección corta del dispositivo que envía el paquete. 32 bits.
    - *FCtr*l: Control de trama. 8 bits.
    - *FCnt*: Contador de trama. 16 bits.
    - *FOpts*: Opciones de trama para transportar comandos MAC. 0 - 120 bits.
- *MAC Payload*:
    - *FHDR*: Cabecera de trama. 56 - 176 bits.

- *FPort*: Puerto opcional por multiplexado. 8 bits.

- *FRMPayload*: Payload opcional encriptado usando *Advanced Encryption Standard* (AES) con una clave de 128 bits.

- *PHY Payload*:

  - *MHDR*: Cabecera MAC. 8 bits.

  - *MACPayload*: Datos de la capa superior.

  - *MIC*: Código criptográfico de integridad del mensaje, calculado sobre los campos *MHDR*, *FHDR*, *FPort* y *FRMPayload* encriptado. 32 bits.

- *Radio PHY Layer*:

  - *MType*: Indica el tipo de mensaje, si es ascendente o descendente, y si es o no un mensaje de confirmación. 3 bits.

  - *RFU*: Reservado para uso futuro. 3 bits. 1 bit en *Downlink FCtrl*.

  - *Major*: Indica la versión de *LoRaWAN*. 2 bits.

  - *ADR*: *Adaptative Data Range*. 1 bit.

  - *ADRAckReq*: Control sobre el mecanismo *ADR*.

  - *ACK*: Confirmación. 1 bit.

  - *FPending*: Indica que el servidor de red aún tiene datos pendientes para enviar e insta al nodo final a transmitir lo antes posible para de esta forma abrir la ventana de recepción. 1 bit.

  - *FOptsLength*: Longitud de FOpts en bytes. 4 bits.

  - *MACCommand*: Comando MAC. 8 – 40 bits.

  - *CID*: Identificador del comando. 8 bits.

  - *Args*: Argumentos opcionales del comando. 0 – 32 bits.

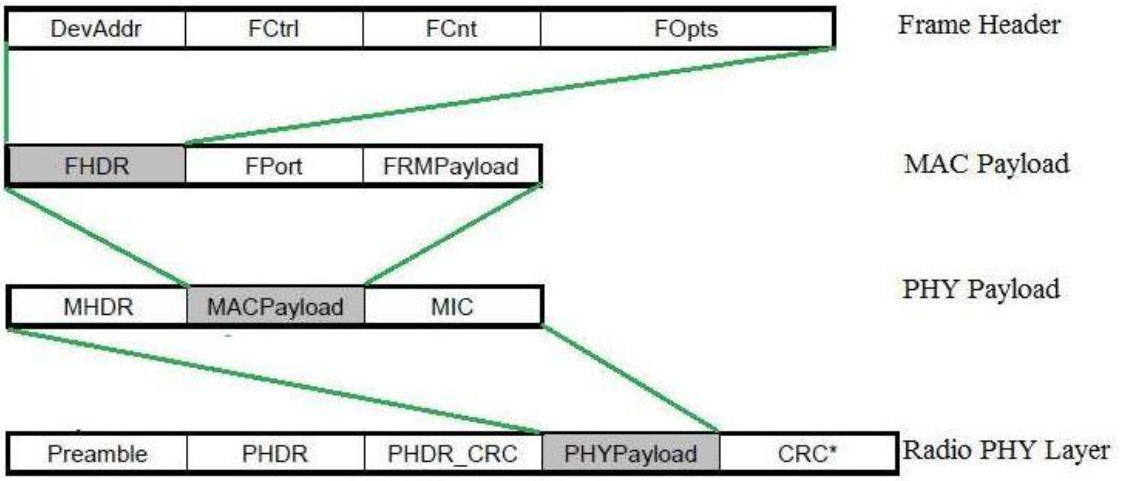


Figura 5.14: Encapsulado de las tramas del paquete LoRaWAN.

En esta descripción de los diferentes campos de los paquetes, se ha nombrado el concepto de comando MAC. *LoRaWAN* define multitud de comandos MAC que permiten configurar los parámetros de los dispositivos finales. Los comandos MAC pueden ser enviados por el nodo final o por el servidor [80]:

- *LinkCheckReq*: Es el único comando MAC que puede ser enviado por un nodo final, y se utiliza para poner a prueba su conectividad.

- *LinkADRReq*: Controla la tasa binaria y la potencia de salida usada por el dispositivo, así como el número de veces que se debe enviar un paquete no confirmado.

- *DutyCycleReq*: Determina el ciclo de trabajo global del dispositivo.

- *RXTimingSetupReq/RXParamSetupReq*: Permite cambiar parámetros de la ventana de recepción.

- *NewChannelReq*: Permite cambiar los canales usados por el dispositivo.

- *DevStatusReq*: Sirve para consultar el nivel de batería y la calidad de recepción de un dispositivo.

Se debe comentar que, debido a la topología de la red, en estrella, la comunicación siempre va a ser nodo final – *gateway*, y viceversa, lo cual conlleva que no se requiera de un campo de dirección del destino en mensajes ascendentes ni dirección de fuente en mensajes descendentes.

5.1.6.4 Acceso a *LoRaWAN*

Para añadir un dispositivo a una red *LoRaWAN,* y poder considerarlo un nodo de ésta, se requiere de una activación en dicha red. Este proceso puede realizarse mediante *Over-The-Air-Activation* (OTAA), o *Activation By Personalization* (ABP). Sea cual sea el proceso elegido para realizar la activación, este ha de proporcionar al nodo final de la información que se indica a continuación. Esta información es imprescindible para garantizar una comunicación correcta y segura dentro de la red.

- *Application Identifier* (*AppEUI*): Identificador global único de aplicación en el espacio de direcciones de *Institute of Electrical and Electronics Engineers* (IEEE) EUI64, que identifica al propietario del nodo final. Longitud de 64 bits.

- *Device Identifier* (*DevEUI*): Identificador global único del dispositivo en el espacio de direcciones IEEE EUI64, que identifica al nodo final. Longitud de 64 bits.

- *Application Key* (*AppKey*): Clave de aplicación de la que derivan las claves de sesión. Longitud de 128 bits. No se usa en el proceso ABP.

- *Device Address* (*DevAddr*): Este es un campo que se incluye siempre en todos los paquetes de comunicación, y consiste en un identificador de 32 bits, de los cuales 7 bits se destinan como identificador de la red, y los 25 restantes para la dirección del dispositivo para esa red. Longitud de 32 bits.

- *Network Session Key* (*NwkSKey*): Clave utilizada por el servidor de la red y el nodo final para calcular y verificar el código de integridad de mensajes para asegurar la integridad de los datos. Longitud de 128 bits.

- *Application Session Key* (*AppSKey*): Clave utilizada por el servidor de la red y el nodo final para encriptar o desencriptar el campo de área de datos de los mensajes. Clave del tipo AES-128. Longitud de 128 bits.

La diferencia fundamental entre el proceso OTAA y ABP reside en la forma con la que se obtienen en cada proceso las claves para una nueva conexión o sesión.

En OTAA, lo primero que se debe hacer es introducir manualmente el identificador global único (*DevEUI*), que como se ha comentado anteriormente identifica de forma única al dispositivo utilizando el esquema IEEE EUI64 [82], el identificador de aplicación (*AppEUI*) y la clave de aplicación específica para el dispositivo (*AppKey*), a partir de la cual se derivarán las claves de sesión y la de sesión de aplicación. El proceso de OTAA realmente comienza cuando se envía por parte del dispositivo final un mensaje MAC denominado *join-request*. Esta petición de unión contiene los parámetros introducidos manualmente con anterioridad, *AppEUI* y *DevEUI*, además de un parámetro adicional denominado *DevNone*. El parámetro *DevNone* es un valor aleatorio utilizado para evitar ataques de reenvío de *join-request* [82]. Si el dispositivo final tiene permiso para unirse a la red, el servidor envía otro mensaje MAC en el que se acepta la petición realizada (*join-accept)*. Esta aceptación está compuesta por un valor aleatorio (*AppNone*), la dirección del dispositivo (*DevAddr*) y el identificador de la red (*NetID*). A partir del parámetro *AppNone* y el valor ya conocido de *AppKey*, se derivan las dos claves de sesión, *AppSKey* y *NwkSKey*. Con estos parámetros almacenados en el dispositivo final, más los introducidos manualmente, ya se pueden realizar transmisiones.

La principal ventaja de usar OTAA como proceso de configuración es que la red genera y envía las claves de encriptación, lo que hace que el sistema resulte más seguro [82]. La desventaja que se puede citar en OTAA, es que se necesita la implementación de un mecanismo de conexión con una complejidad añadida.

En el proceso ABP toda la configuración se realiza de manera manual, debiéndose introducir en el dispositivo final la dirección del dispositivo y las claves de sesión. Una vez hecho esto, ya se puede transmitir.

La principal ventaja de usar ABP es clara, la facilidad y la rapidez con la que se puede conectar a la red *LoRaWAN.* El dispositivo puede estar disponible para transmitir en muy poco tiempo. La desventaja se muestra en que las claves de encriptación ya están preconfiguradas en el dispositivo, y esto resta seguridad.

5.1.6.5 Seguridad

Con todos los campos de las tramas de los paquetes *LoRaWAN*, y con la definición de todas las claves que intervienen en la configuración de los nodos de la red *LoRaWAN*, ya se puede hablar en términos de seguridad.

La seguridad en IoT es un aspecto fundamental, pues los datos que se intercambian muestran explícita o implícitamente información personal o privada. Y aunque no fuera así, en muchos casos se desea guardar la confidencialidad de la información transmitida. *LoRaWAN* tiene esto muy en cuenta, y por eso posee mecanismos de autenticación mutua, de integridad y confidencialidad. Todo centrado en un esquema de claves simétricas.

Ya en el proceso de activación, se requiere de una autenticación mutua entre un dispositivo inicial y el servidor de red. Es decir, solo los dispositivos autorizados pueden activarse en una determinada red. El dispositivo final utiliza la *AppKey* para calcular el campo *MIC* que se recuerda que es el código de integridad del mensaje [82]. En el otro extremo, el servidor tiene almacenada la *AppKey* del dispositivo final, y por tanto comprueba que el mensaje que recibe por parte del dispositivo final es auténtico. Además, para que el dispositivo final se asegure que la respuesta proviene del servidor adecuado, este responde con el *join-accept* cuyo *MIC* ha sido calculado con la *AppKey* que comparten el dispositivo final y el propio servidor. Además, el mismo mensaje de aceptación se cifra con la *AppKey*.

Toda la seguridad en el tráfico normal de paquetes se realiza a través de las dos claves de sesión, la de sesión de red (*NwkSKey*) y la de aplicación (*AppSKey*) [82], que se encargan de

cifrar y firmar los datos. Se recuerda que ambas claves derivan de *AppKey*, y por tanto solo son conocidas por el dispositivo final y el servidor de red.

Así, se usa la misma clave para cifrar y descifrar, por lo que se necesita que tanto el nodo final como el servidor conozcan las claves, que deben ser las mismas. Esto implica que se deben enviar en algún momento esas claves compartidas, poniendo en riesgo la seguridad de la red. *LoRaWAN* soluciona esto compartiendo la *AppKey*, que no hay problema en que sea compartida, y con la clave aleatoria que se genera en el proceso de activación [82]. A partir de estos dos componentes se derivan las clases de sesión que serán comunes para el dispositivo final y para el servidor, sin tener que ser compartidas entre ellos. Por tanto, esta información solo está disponible en estos dos puntos de la red.

La intercepción de la *AppKey* no tiene consecuencias en la seguridad, porque solo serviría para activar fraudulentamente un dispositivo en una red determinada. Se recuerda que el algoritmo de cifrado es AES-128 bits, que mantiene una buena relación entre el coste computacional y la robustez.

A modo de resumen, en la Figura 5.15 [68] se representa el esquema que sigue una red *LoRa* con los tres elementos clave que intervienen en ella, el sensor, el *gateway* y el servidor de la red. También se muestran los diferentes niveles de comunicación que tiene cada elemento, y se remarcan aquellos en los que se establece una conexión segura con cifrado.

Tras realizar un análisis completo de la tecnología *LoRa* se puede concluir que es una tecnología que se adapta perfectamente a las necesidades de IoT, pues su consumo de potencia es muy bajo y su alcance elevado. Otra ventaja que presenta *LoRa* es la capacidad para ajustar la comunicación a las necesidades reales del nodo que quiere transmitir. Todos estos ajustes los establece *LoRaWAN* usando el ya comentado *ADR*. Se debe recordar que el alcance y la tasa binaria dependen del SF que se establezca.

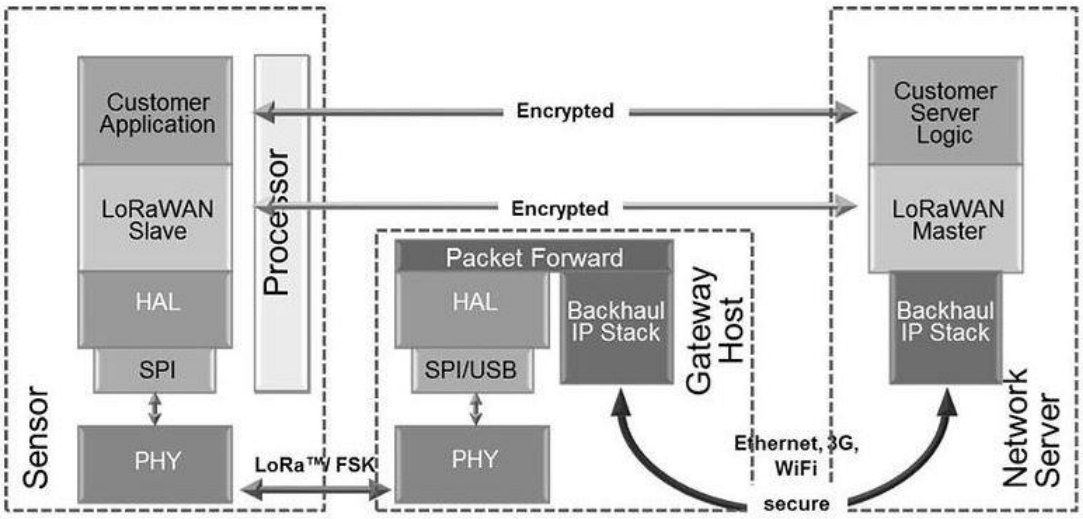


Figura 5.15: Relaciones entre los elementos típicos de una red IoT basada en LoRa.

5.2 Implementación de *LoRa/LoRaWAN* en la plataforma HW/SW

En este apartado se describe la implementación de la comunicación basada en *LoRa/LoRaWAN* para la plataforma HW/SW propuesta en este TFM. Como se ha visto en la descripción teórica de esta tecnología, se pretende que un nodo estación base se conecte a una red *LoRaWAN* para comunicarse con el nodo *gateway*. En este mismo apartado se comentará la codificación requerida por el nodo estación base para comunicarse con el nodo *gateway*. Mientras que en el siguiente capítulo se explicará detalladamente la codificación requerida por el nodo *gateway*. No obstante, en este apartado se muestra también el resultado de una correcta comunicación entre el nodo estación base y el nodo *gateway* usando la tecnología de comunicación *LoRa/LoRaWAN*. La implementación de la comunicación basada en *LoRa/LoRaWAN* se realiza sobre el dispositivo LoPy.

5.2.1 Implementación de *LoRa/LoRaWAN* en el nodo estación base

La estación base debe conectarse a una red *LoRa/LoRaWAN* antes de comenzar con el flujo de información. Para unirse a esta red se debe realizar un proceso de activación, bien mediante ABP o mediante OTAA. En el caso particular de este TFM, el modo de activación utilizado ha sido OTAA, por lo que se debe realizar una asignación de claves para el

dispositivo utilizado. Estas claves deben estar presentes, tanto en la plataforma *The Things Network* (TTN), como en el código del nodo estación base. Las tres claves necesarias son las siguientes:

- *DevEUI*: Es el identificador del dispositivo en la red *LoRaWAN*. Este identificador puede ser editado por el usuario y debe copiarse en el campo *DevEUI* de uno de los dispositivos asociados a la aplicación que se encuentra en la plataforma *The Things Network* (TTN). Se extrae del dispositivo utilizando el comando:

```
binascii.hexlify(network.LoRa().mac())
```

- *AppKEY*: Es la clave de la aplicación que genera automáticamente TTN al crear una aplicación en su plataforma.

- *AppEUI*: Es el identificador de la aplicación, es otro campo que genera automáticamente TTN.

En la Figura 5.16 se puede observar que el primer paso para la configuración de la tecnología de comunicación en el dispositivo LoPy es elegir el modo de funcionamiento, pudiendo ser este, *LoRa* o *LoRaWAN*. Seguidamente, se introducen las claves necesarias para la activación OTAA, indicadas anteriormente. En esta misma Figura 5.16, se observa cómo se añaden los tres canales por defecto que se reservan para la comunicación, todos ellos en la misma frecuencia. También se limitan los posibles valores de *Data Rate*, que pueden utilizar los canales por defecto, de '0' a '5'. El valor escogido está directamente relacionado con el valor de *Spreading Factor*.

```
########################################Configuración LoRa/LoRaWAN####################################
# Se inicializa LoRa en el modo LoRaWAN
# Se pueden añadir más parámteros como la frecuencia, la potencia de transmisión o el Spreading Factor
lora = LoRa(mode=LoRa.LORAWAN)

# Se crean los parámteros de autenticación para la activación OTAA
dev_eui = binascii.unhexlify('70B3D54997783F1E'.replace(' ',')) #Lopy-device-fbt-2
app_key = binascii.unhexlify('0C546F91E25899E5E283A27D5A44BF50'.replace(' ',')) #Lopy-device-fbt-1
app_eui = binascii.unhexlify('70B3D57ED0007139'.replace(' ',''))

# Se dejan los 3 canales por defecto en la misma frecuencia. (Se debe hacer antes del join-request)
lora.add_channel(0, frequency=868100000, dr_min=0, dr_max=5)
lora.add_channel(1, frequency=868100000, dr_min=0, dr_max=5)
lora.add_channel(2, frequency=868100000, dr_min=0, dr_max=5)
```

Figura 5.16: Modo de funcionamiento LoRa/LoRaWAN y claves de activación OTAA.

Una vez que se tiene configurado el modo de funcionamiento, las claves de activación OTAA, y los canales por defecto, se puede pasar al envío de la solicitud de *Join-request*. Con este mensaje que se envía al nodo *gateway*, se comprueba si el nodo estación base que envía la solicitud de conexión está configurado en la plataforma TTN. En caso positivo, el nodo estación base y el nodo *gateway* ya pueden comunicarse de forma continua, mientras que, si las claves no coinciden, el nodo estación base se mantiene enviando solicitudes, aunque no vaya a ser aceptado nunca. Así, tal y como se muestra en la Figura 5.17, el nodo estación base se mantiene en un bucle hasta que se acepta su solicitud. Cabe destacar que se ha realizado el bucle porque debido a problemas de conexión entre el nodo *gateway* y el servidor, puede que no se procesen todas las solicitudes de activación, por lo que se reintenta continuamente el establecimiento de la conexión. No obstante, si las claves introducidas en el nodo estación base y las definidas en la plataforma TTN no coinciden, esta activación nunca llegará a producirse, por lo que la espera infinita del nodo estación base se puede interpretar como un error en el proceso de activación.

En la Figura 5.17 también se puede observar cómo se eliminan los canales que no sean los establecidos por defecto, además de crear un *socket* y configurarlo con un *Data Rate* determinado, en este caso con un valor de 5.

```
# Se envía el join-request a la red vía OTAA
lora.join(activation=LoRa.OTAA, auth=(dev_eui, app_eui, app_key), timeout=0, dr=5)

# Se mantiene en espera hasta que se acepte el join.request
while not lora.has_joined():
    time.sleep(2.5)
    print('Not joined yet...')

print('Joined!')

# Se borran todos los canales que no sean por defecto
for i in range(3, 16):
    lora.remove_channel(i)

# Se crea un socket LoRa
s = socket.socket(socket.AF_LORA, socket.SOCK_RAW)

# Se configura el data rate de LoRaWAN
s.setsockopt(socket.SOL_LORA, socket.SO_DR, 5)
```

Figura 5.17: Join-request, eliminación de canales no prestablecidos y configuración del socket.

En el desarrollo de la funcionalidad del nodo en el dispositivo, se utiliza el *socket* que se acaba de crear y configurar, siendo la manera correcta de utilizarlo la siguiente: si se desea recibir datos procedentes del nodo *gateway*, se debe bloquear el *socket*, mientras que, para poder enviar, se debe desbloquear. El código asociado a las funciones de bloqueo y desbloqueo es el que se muestra en la Figura 5.18.

```
# Se bloquea el socket
s.setblocking(False)
# Se desbloquea el socket
s.setblocking(True)
```

Figura 5.18: Bloqueo y desbloqueo del socket.

5.2.2 Prueba de la comunicación *LoRa/LoRaWAN*

Con el fin de realizar una prueba de comunicación entre el nodo estación base y el nodo *gateway* usando *LoRa/LoRaWAN* como tecnología de comunicación, se añade a la codificación de la funcionalidad de la estación base el código de la Figura 5.19. En esta figura se desbloquea el *socket* para poder realizar un envío y se crea una variable de tipo

bytearray, denominada 'frame', que contiene 5 valores distintos. Finalmente se entra en el bucle infinito que envía continuamente la variable 'frame' con su contenido.

```
50  ##########################################################
51  # Se bloquea el socket
52  s.setblocking(False)  #Para poder recibir
53  # Se desbloquea el socket
54  s.setblocking(True)   #Para poder enviar
55
56  ##########################################################
57  #Se crea un bytearray para la gestión de las tramas
58  frame = bytearray(5)
59  frame[0] = 0x01
60  frame[1] = 0x03
61  frame[2] = 0x05
62  frame[3] = 0x07
63  frame[4] = 0x09
64
65  ##########################################################
66  while 1:
67      s.send(frame)
```

Figura 5.19: Codificación de la funcionalidad de la estación base.

Con la configuración adecuada del *gateway,* que se comentará en el capítulo 6 y con la correcta gestión de la plataforma *The Things Network* (TTN) que se explicará en el capítulo 9, se logra el resultado de comunicación que se muestra en la Figura 5.20. En esta imagen se puede observar cómo se recibe el *array* de bytes esperado, en orden y de forma repetida debido al bucle infinito creado en el nodo estación base.

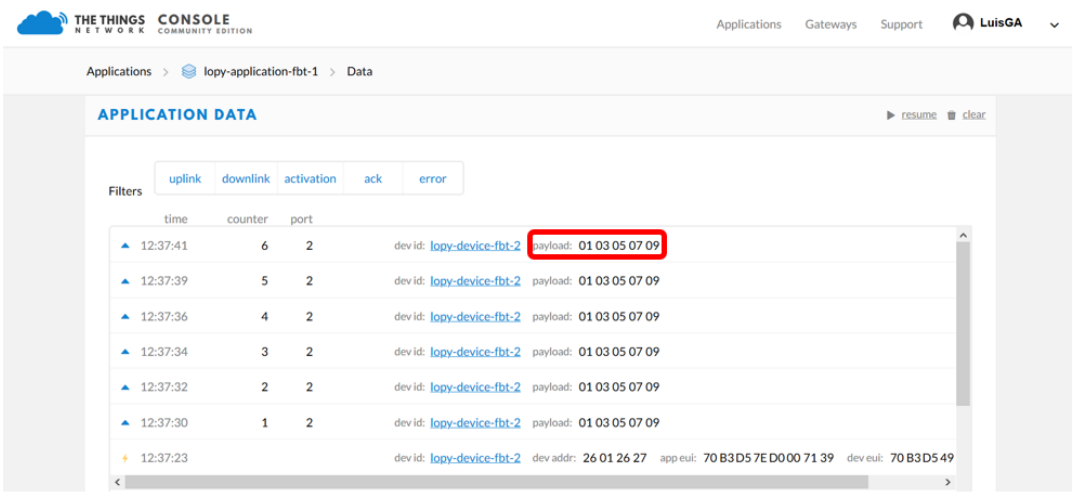


Figura 5.20: Resultado de la prueba de comunicación LoRa/LoRaWAN.

Con esta prueba queda validada la implementación de la comunicación basada en *LoRa/LoRaWAN* sobre dispositivos LoPy para la plataforma HW/SW propuesta en este TFM.

Capítulo 6. Nanogateway

La comunicación mediante la tecnología *LoRa/LoRaWAN* que se trató en el capítulo anterior se implementa entre el nodo estación base y el nodo *gateway*, para el caso de la plataforma HW/SW propuesta en el presente TFM.

El nodo *gateway* es el encargado de gestionar la comunicación con los nodos estación base y con la plataforma *The Things Network*. La comunicación con los nodos estación base se realiza mediante la tecnología *LoRa/LoRaWAN*, mientras que la comunicación con la plataforma TTN se establece vía *WiFi*. En la Figura 6.1 se recuerda la arquitectura completa de la plataforma HW/SW propuesta. Por lo tanto, a diferencia de los nodos finales y los nodos estación base, el dispositivo que actúa como *gateway* se ubica en un punto estratégico para que pueda tener acceso a Internet, que en este caso concreto se materializa en el uso de tecnología *WiFi*. Es decir, debe estar ubicado a una distancia adecuada de los nodos estación base, que como se indicó en capítulos anteriores, en el caso de un par de kilómetros la comunicación es totalmente viable, aun variando el *Spreading Factor* entre todos los valores posibles. Además, debe estar a unos pocos metros de un *router* con acceso a Internet con el fin de poder establecer la comunicación *WiFi* con la plataforma TTN. En definitiva, el *gateway* debe ubicarse en un punto donde ambos tipos de comunicación puedan darse sin ningún problema.

El dispositivo utilizado para implementar la funcionalidad del *gateway* es el mismo que se usa como nodo final y nodo estación base, el dispositivo LoPy de Pycom. Se recuerda que este dispositivo cuenta con tecnología *WiFi*, *Bluetooth* y *LoRa*, por lo que sus características

se adaptan perfectamente a las necesidades de un *gateway* en una solución como la planteada en el presente TFM. Se utilizará la tecnología *WiFi* para la conexión a Internet, y la tecnología *LoRa/LoRaWAN* para la comunicación con el nodo estación base.

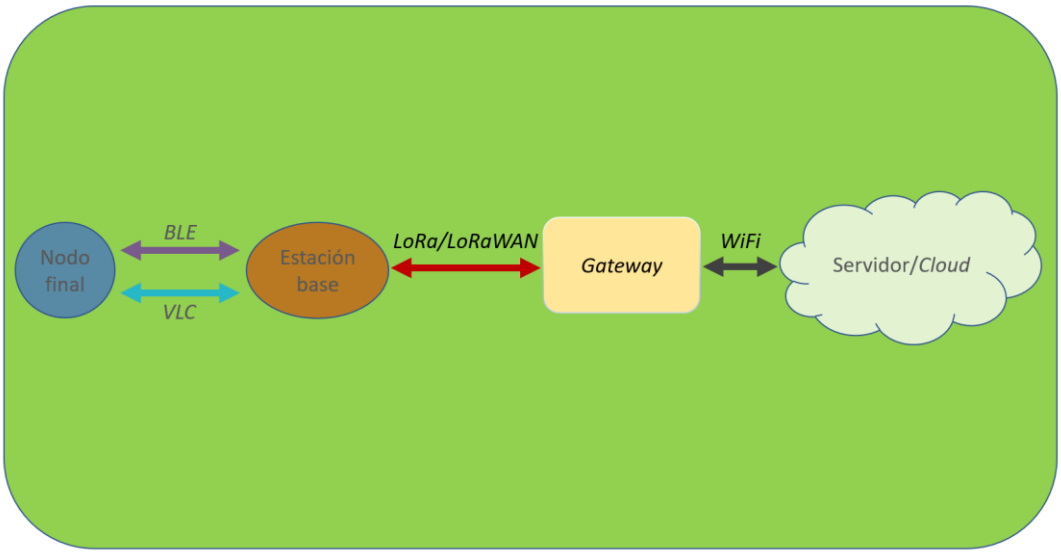


Figura 6.1: Arquitectura de la plataforma HW/SW propuesta: Elemento Gateway.

No obstante, el *gateway* de esta solución es del tipo denominado como *nanogateway* o *gateway* monocanal.

Se denomina *nanogateway* a un *gateway* de un único canal o "*single-channel gateway*". Los *gateways* de canal único solo pueden recibir datos por un canal, y con un único *Spreading Factor* al mismo tiempo, mientras que los *gateways* "completos" pueden recibir por 8 canales diferentes y con 6 valores posibles de *Spreading Factor* simultáneamente [83]. Algunos *gateways* de canal único pueden "saltar" entre frecuencias y valores de S*preading Factor* para simular un *gateway* "completo", pero aun así se mantienen por debajo del 2% de la capacidad real de un *gateway* "completo".

Es importante resaltar que utilidades como *Adaptative Data Rate* no se pueden utilizar con estos *nanogateways*, ya que requieren de múltiples canales para realizar las adaptaciones a las necesidades reales de la red [83].

La principal ventaja que se tiene en los *nanogateways* es su coste, que resulta mucho más asequible, económicamente hablando, que los *gateways* "completos". En resumen, conociendo las limitaciones a nivel de canales y de *Spreading Factor* en las recepciones

simultáneas, se puede hacer uso de este tipo de *gateway* sin generar ningún conflicto en la aplicación concreta propuesta en el presente TFM.

Como se ha comentado anteriormente, se utilizará el dispositivo LoPy para la implementación de la funcionalidad del *nanogateway*, que está certificado por *LoRaWAN*, aunque como nodo, no como *gateway*. Por lo tanto, teniendo claro que no es la solución ideal para una gran red de dispositivos, se utiliza el dispositivo LoPy como *nanogateway* con el fin de realizar pruebas experimentales, y especialmente para pequeñas aplicaciones en las que se controlan las frecuencias utilizadas por los nodos, como por ejemplo en los nodos LoPy. Así, el *nanogateway* queda como se muestra en la Figura 6.2.

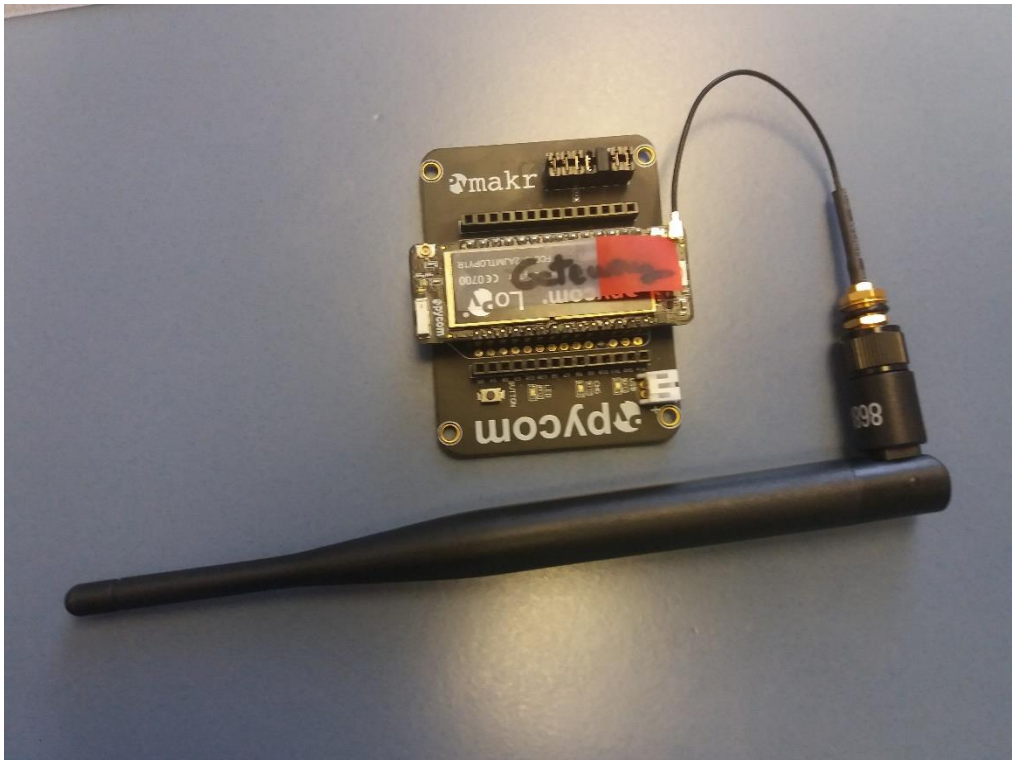


Figura 6.2: Dispositivo LoPy utilizado como nanogateway.

De este modo, se justifica el uso de este dispositivo de Pycom como *nanogateway* por el bajo coste que represente en comparación a un *gateway* "completo", y porque la aplicación propuesta se encuentra dentro de los límites de un *gateway* de canal único.

Pycom, en su documentación [84] facilita el código de un *nanogateway* de referencia, que incorpora todas las funciones requeridas por este, y en el que, en principio, únicamente

hay que configurar los parámetros asociados a la comunicación *WiFi* (*Service Set Identifier* (SSID) y contraseña). El código actualizado se puede consultar en *Github* [85].

En el caso concreto del *nanogateway* utilizado en el presente TFM, consta de 3 archivos en *MicroPython*:

- `Main.py`: Inicialización del funcionamiento del *nanogateway*.
- `Nanogateway.py`: Código que codifica la funcionalidad del *nanogateway*.
- `Config.py`: Destinado a la configuración de las opciones del *nanogateway*.

6.1 Archivo: `nanogateway.py`

Se comienza por el análisis del fichero *nanogateway.py*, pues los otros dos archivos tienen dependencias con este fichero. Las librerías utilizadas por este archivo son las que se muestran en la Figura 6.3.

```
""" LoPy LoRaWAN Nano Gateway. Can be used for both EU868 and US915. """

import errno
import machine
import ubinascii
import binascii
import ujson
import uos
import usocket
import utime
import _thread
from micropython import const
from network import LoRa
from network import WLAN
from machine import Timer
```

Figura 6.3: Librerías utilizadas en nanogateway.py.

Seguidamente, en la Figura 6.4 se adjuntan las constantes utilizadas en el código. Como se puede observar en la Figura 6.4, se incluyen constantes asociadas a diferentes mensajes de error, mostrándose su uso en las funciones definidas.

En esta parte dedicada a la declaración de constantes, también se incluyen estructuras de metadatos, que como se aprecia en la Figura 6.5, se inicializan a un valor por defecto ('0' en la mayoría de los casos). La estructura de la Figura 6.5, STAT_PK, se rellena con campos y valores relacionados con la estadística actual del *nanogateway*, por lo que incluye campos como el tiempo transcurrido desde que se inició su funcionamiento, su ubicación, etc. También se inicializa la estructura RX_PK, cuyos campos y valores se relacionan directamente con el estado de la recepción de paquetes. Tanto la estructura RX_PK como TX_ACK_PK se adjuntan en la Figura 6.6. La estructura TX_ACK_PK contiene información sobre el estado de la transmisión realizada. Su único campo es el de error, por lo que está orientado hacia el aviso de errores en la transmisión. Posteriormente, en las funciones se hará uso de estas estructuras y se rellenará su contenido con los valores oportunos.

```
PROTOCOL_VERSION = const(2)

PUSH_DATA = const(0)
PUSH_ACK = const(1)
PULL_DATA = const(2)
PULL_ACK = const(4)
PULL_RESP = const(3)

TX_ERR_NONE = 'NONE'
TX_ERR_TOO_LATE = 'TOO_LATE'
TX_ERR_TOO_EARLY = 'TOO_EARLY'
TX_ERR_COLLISION_PACKET = 'COLLISION_PACKET'
TX_ERR_COLLISION_BEACON = 'COLLISION_BEACON'
TX_ERR_TX_FREQ = 'TX_FREQ'
TX_ERR_TX_POWER = 'TX_POWER'
TX_ERR_GPS_UNLOCKED = 'GPS_UNLOCKED'

UDP_THREAD_CYCLE_MS = const(10)
```

Figura 6.4: Variables utilizadas en nanogateway.py.

En la Figura 6.7 se crea la clase NanoGateway, que será utilizada en el archivo *main.py*. Dentro de esta clase se definen todas las funciones que permiten que el dispositivo LoPy

se comporte como *nanogateway*. El resto del código de este archivo, por tanto, son funciones que se describirán a continuación. En la Figura 6.7 se muestra un fragmento de la función *__init__()*.

```
STAT_PK = {
    'stat': {
        'time': '',
        'lati': 0,
        'long': 0,
        'alti': 0,
        'rxnb': 0,
        'rxok': 0,
        'rxfw': 0,
        'ackr': 100.0,
        'dwnb': 0,
        'txnb': 0
    }
}
```

Figura 6.5: Estructura STAT_PK.

```
RX_PK = {
    'rxpk': [{
        'time': '',
        'tmst': 0,
        'chan': 0,
        'rfch': 0,
        'freq': 0,
        'stat': 1,
        'modu': 'LORA',
        'datr': '',
        'codr': '4/5',
        'rssi': 0,
        'lsnr': 0,
        'size': 0,
        'data': ''
    }]
}

TX_ACK_PK = {
    'txpk_ack': {
        'error': ''
    }
}
```

Figura 6.6: Estructuras RX_PK y TX_ACK_PK.

```
class NanoGateway:
    """
    Nano gateway class, set up by default for use with TTN, but can be configured
    for any other network supporting the Semtech Packet Forwarder.
    Only required configuration is wifi_ssid and wifi_password which are used for
    connecting to the Internet.
    """

    def __init__(self, id, frequency, datarate, ssid, password, server, port, ntp_server='pool.ntp.org', ntp_period=3600):
        self.id = id
        self.server = server
        self.port = port

        self.frequency = frequency
        self.datarate = datarate

        self.ssid = ssid
        self.password = password

        self.ntp_server = ntp_server
        self.ntp_period = ntp_period
```

Figura 6.7: Creación de la clase NanoGateway y de la función __init__().

La configuración del *gateway* se realiza a partir de los parámetros que se pasan a la función *__init__()*. Estos parámetros, como se verá más adelante, se obtienen del archivo *config.py*. En esta función simplemente se recogen los valores de los parámetros de configuración y se asignan al objeto creado de la clase NanoGateway.

Además de volcar estos parámetros de configuración, también se inicializan valores como el *Spreading Factor* o el ancho de banda, entre otros, tal y como se muestra en la Figura 6.8. Estos dos valores, por ejemplo, se recogen a partir del parámetro de configuración "*datarate*", que como se verá en el archivo *config.py*, incluye en un *String*, tanto el *Spreading Factor* como el ancho de banda. Para extraer la información del *String* se hace uso de dos funciones, que se presentan en la Figura 6.9, denominadas *_dr_to_sf()* y *_dr_to_bw()*. Además, en la misma Figura 6.9 se tiene la conversión inversa a partir de la función *_sf_bw_to_dr().*

El resto de las inicializaciones están relacionadas con las comunicaciones *WiFi* y *LoRa*, pero en esta función no se realiza ninguna acción determinante en su funcionamiento, ya que todas las inicializaciones se mantienen en '*None*'. Por último, se asocia el RTC creado con el RTC propio del objeto NanoGateway.

```
97          self.ntp_period = ntp_period
98
99          self.server_ip = None
100
101         self.rxnb = 0
102         self.rxok = 0
103         self.rxfw = 0
104         self.dwnb = 0
105         self.txnb = 0
106
107         self.sf = self._dr_to_sf(self.datarate)
108         self.bw = self._dr_to_bw(self.datarate)
109
110         self.stat_alarm = None
111         self.pull_alarm = None
112         self.uplink_alarm = None
113
114         self.wlan = None
115         self.sock = None
116         self.udp_stop = False
117         self.udp_lock = _thread.allocate_lock()
118
119         self.lora = None
120         self.lora_sock = None
121
122         self.rtc = machine.RTC()
```

Figura 6.8: Función __init__()_2.

La siguiente función que se analiza es *start()*. No obstante, con antelación se debe comentar brevemente la existencia de la función *_log()* que se encarga de crear mensajes, darles formato, e imprimirlos por pantalla. El código se adjunta en la Figura 6.10.

El código de la Figura 6.11 y la Figura 6.12 muestra la declaración de la función *start()* y su contenido. Lo primero que se hace es imprimir por pantalla un mensaje de inicio con el identificador del *gateway*, haciendo uso de la función *_log()*. Seguidamente, se configura la conexión *WiFi*, para lo cual, se utiliza la función *_connect_to_WiFi()*, cuyo código se adjunta en la Figura 6.13. Una vez configurado y conectado a la red *WiFi*, se sincroniza el RTC con el servidor NTP (servidor horario).

```
def _dr_to_sf(self, dr):
    sf = dr[2:4]
    if sf[1] not in '0123456789':
        sf = sf[:1]
    return int(sf)

def _dr_to_bw(self, dr):
    bw = dr[-5:]
    if bw == 'BW125':
        return LoRa.BW_125KHZ
    elif bw == 'BW250':
        return LoRa.BW_250KHZ
    else:
        return LoRa.BW_500KHZ

def _sf_bw_to_dr(self, sf, bw):
    dr = 'SF' + str(sf)
    if bw == LoRa.BW_125KHZ:
        return dr + 'BW125'
    elif bw == LoRa.BW_250KHZ:
        return dr + 'BW250'
    else:
        return dr + 'BW500'
```

Figura 6.9: Conversiones Datarate-SF/BW.

```
    def _log(self, message, *args):
        """
        Outputs a log message to stdout.
        """

        print('[{:>10.3f}] {}'.format(
            utime.ticks_ms() / 1000,
            str(message).format(*args)
))
```

Figura 6.10: Función _log().

El siguiente paso consiste en acceder al servidor IP, crear un socket UDP *(User Datagram Protocol)*, y dejarlo listo para su uso. Posteriormente, se envía inmediatamente el primer paquete STAT_PK. Para rellenar la estructura de este paquete se utiliza la función

_make_stat_packet(), mientras que para enviar el paquete se hace uso de la función *_push_data()*. Ambas se definen en la Figura 6.14 y la Figura 6.15, respectivamente.

También se crean las interrupciones, haciendo uso de las funciones de alarma del *Timer*. Así, cuando se cumple el tiempo establecido (60 segundos para la primera alarma y 25 para la segunda), y se llama a la función *_push_data()* en el primer caso, y a la función *_pull_data()* en el segundo. El código de la función *_pull_data()* se adjunta en la Figura 6.15.

En la parte final de la función, se comienza un *thread* de recepción UDP haciendo uso de la función *_udp_thread(),* que se detalla en la Figura 6.16, la Figura 6.17 y la Figura 6.18, y se configura *LoRa* en el modo *LoRa* (no *LoRaWAN*), y con los parámetros de configuración indicados, el *socket* asociado.

Por último, si se produce un evento de tipo *TX_PACKET_EVENT* o *RX_PACKET_EVENT,* se llama a la función *_lora_cb()*, cuyo código se adjunta en la Figura 6.19, y que se encarga de dar respuesta al evento acontecido.

Se debe comentar, además, que en una de las llamadas a la función *_log()* se hace uso de la función *_freq_to_float()*, que como su nombre indica convierte el parámetro que se le pasa (una frecuencia) en una variable de tipo *float*, con el fin de poder trabajar con mayor precisión. En la Figura 6.20 se adjunta el código correspondiente a la implementación de dicha función.

En la función *_connect_to_WiFi()* de la Figura 6.13 se ve cómo se introducen el SSID y la contraseña para poder establecer la conexión *WiFi*. A continuación, se comprueba si se está conectado, y cuando lo está, se imprime un mensaje por pantalla indicando esta situación.

En la función *_make_stat_packet()* de la Figura 6.14 se rellena la estructura STAT_PK con los valores actualizados. Se observa cómo para el primer campo, se requiere hacer una lectura del RTC. El resto de los valores se consultan directamente al objeto NanoGateway creado.

```
    def start(self):
        """
        Starts the LoRaWAN nano gateway.
        """

        self._log('Starting LoRaWAN nano gateway with id: {}', self.id)

        # setup WiFi as a station and connect
        self.wlan = WLAN(mode=WLAN.STA)
        self._connect_to_wifi()

        # get a time sync
        self._log('Syncing time with {} ...', self.ntp_server)
        self.rtc.ntp_sync(self.ntp_server, update_period=self.ntp_period)
        while not self.rtc.synced():
            utime.sleep_ms(50)
        self._log("RTC NTP sync complete")

        # get the server IP and create an UDP socket
        self.server_ip = usocket.getaddrinfo(self.server, self.port)[0][-1]
        self._log('Opening UDP socket to {} ({}) port {}...', self.server, self.server_ip[0], self.server_ip[1])
        self.sock = usocket.socket(usocket.AF_INET, usocket.SOCK_DGRAM, usocket.IPPROTO_UDP)
        self.sock.setsockopt(usocket.SOL_SOCKET, usocket.SO_REUSEADDR, 1)
        self.sock.setblocking(False)
```

Figura 6.11: Función start().

```
        self._push_data(self._make_stat_packet())
        # create the alarms
        self.stat_alarm = Timer.Alarm(handler=lambda t: self._push_data(self._make_stat_packet()), s=60, periodic=True)
        self.pull_alarm = Timer.Alarm(handler=lambda u: self._pull_data(), s=25, periodic=True)

        # start the UDP receive thread
        self.udp_stop = False
        _thread.start_new_thread(self._udp_thread, ())

        # initialize the LoRa radio in LORA mode
        self._log('Setting up the LoRa radio at {:.1f} Mhz using {}', self._freq_to_float(self.frequency), self.datarate)
        self.lora = LoRa(
            mode=LoRa.LORA,
            frequency=self.frequency,
            bandwidth=self.bw,
            sf=self.sf,
            preamble=8,
            coding_rate=LoRa.CODING_4_5,
            tx_iq=True
        )
        # create a raw LoRa socket
        self.lora_sock = usocket.socket(usocket.AF_LORA, usocket.SOCK_RAW)
        self.lora_sock.setblocking(False)
        self.lora_tx_done = False

        self.lora.callback(trigger=(LoRa.RX_PACKET_EVENT | LoRa.TX_PACKET_EVENT), handler=self._lora_cb)
        self._log('LoRaWAN nano gateway online')
```

Figura 6.12: Función start()_2.

```
def _connect_to_wifi(self):
    self.wlan.connect(self.ssid, auth=(None, self.password))
    while not self.wlan.isconnected():
        utime.sleep_ms(50)
    self._log('WiFi connected to: {}', self.ssid)
```

Figura 6.13: Función _connect_to_WiFi().

```
def _make_stat_packet(self):
    now = self.rtc.now()
    STAT_PK["stat"]["time"] = "%d-%02d-%02d %02d:%02d:%02d GMT" % (now[0], now[1], now[2], now[3], now[4], now[5])
    STAT_PK["stat"]["rxnb"] = self.rxnb
    STAT_PK["stat"]["rxok"] = self.rxok
    STAT_PK["stat"]["rxfw"] = self.rxfw
    STAT_PK["stat"]["dwnb"] = self.dwnb
    STAT_PK["stat"]["txnb"] = self.txnb
    return ujson.dumps(STAT_PK)
```

Figura 6.14: Función _make_stat_packet().

En la Figura 6.15 se encuentra el código de las funciones *_push_data()* y *_pull_data()*. La primera función se encarga de enviar los datos que se le pasan por parámetro con el *socket* UDP hacia el servidor IP. Mientras que a la segunda función no se le pasa ningún parámetro, y, por tanto, no envía más que el identificador del *gateway*.

```
def _push_data(self, data):
    token = uos.urandom(2)
    packet = bytes([PROTOCOL_VERSION]) + token + bytes([PUSH_DATA]) + ubinascii.unhexlify(self.id) + data
    with self.udp_lock:
        try:
            self.sock.sendto(packet, self.server_ip)
            self._log('push_data packet: {}'.format(binascii.hexlify(packet)))
        except Exception as ex:
            self._log('Failed to push uplink packet to server: {}', ex)

def _pull_data(self):
    token = uos.urandom(2)
    packet = bytes([PROTOCOL_VERSION]) + token + bytes([PULL_DATA]) + ubinascii.unhexlify(self.id)
    with self.udp_lock:
        try:
            self.sock.sendto(packet, self.server_ip)
            self._log('pull_data packet: {}'.format(binascii.hexlify(packet)))
        except Exception as ex:
            self._log('Failed to pull downlink packets from server: {}', ex)
```

Figura 6.15: Función _push_data() y _pull_data().

En la Figura 6.19 se adjunta el código de la función *_lora_cb()*. En esta función, según el evento que se produzca, se lleva a cabo un tipo de acciones u otras. Así, si el evento es de tipo RX_PACKET_EVENT, se incrementan variables asociadas al número de paquetes recibidos, se recibe el paquete y se consulta el estado de la recepción del paquete. Además, los datos recibidos se pasan como parámetro a la función *_make_node_packet()*, que como se muestra en la Figura 6.21, rellena los datos de la estructura de tipo RX_PK. Con la estructura formada, se llama a la función *_push_data()* para que la envíe al servidor IP.

En caso de que el evento sea de tipo TX_PACKET_EVENT, se incrementa la variable que contabiliza el número de transmisiones realizadas, y se inicia la comunicación LoRa con los parámetros configurados.

En la función *_udp_thread()* de la Figura 6.16, la Figura 6.17 y la Figura 6.18, se reciben los datos del servidor y se realizan diferentes acciones sobre ellos. Lo primero que se hace es analizar el tipo de dato recibido, de manera que si es un *Acknowledgement* (ACK) se imprime por pantalla un mensaje que indica que los datos han sido enviados y recibidos correctamente, mientras que si es de tipo PULL_RESP, significa que se ha recibido otro tipo de dato que no es de confirmación.

```
360     def _udp_thread(self):
361         """
362         UDP thread, reads data from the server and handles it.
363         """
364
365         while not self.udp_stop:
366             try:
367                 data, src = self.sock.recvfrom(1024)
368                 _token = data[1:3]
369                 _type = data[3]
370                 if _type == PUSH_ACK:
371                     self._log("Push ack")
372                 elif _type == PULL_ACK:
373                     self._log("Pull ack")
```

Figura 6.16: Función _udp_thread().

```
372                 elif _type == PULL_ACK:
373                     self._log("Pull ack")
374                 elif _type == PULL_RESP:
375                     self.dwnb += 1
376                     ack_error = TX_ERR_NONE
377                     tx_pk = ujson.loads(data[4:])
378                     tmst = tx_pk["txpk"]["tmst"]
379                     t_us = tmst - utime.ticks_us() - 12500
380                     if t_us < 0:
381                         t_us += 0xFFFFFFFF
382                     if t_us < 20000000:
383                         self.uplink_alarm = Timer.Alarm(
384                             handler=lambda x: self._send_down_link(
385                                 ubinascii.a2b_base64(tx_pk["txpk"]["data"]),
386                                 tx_pk["txpk"]["tmst"] - 50, tx_pk["txpk"]["datr"],
387                                 int(tx_pk["txpk"]["freq"] * 1000000)
388                             ),
389                             us=t_us
390                         )
391                     else:
392                         ack_error = TX_ERR_TOO_LATE
393                         self._log('Downlink timestamp error!, t_us: {}', t_us)
394                     self._ack_pull_rsp(_token, ack_error)
395                     self._log("Pull rsp")
```

Figura 6.17: Función _udp_thread()_2.

Ese dato se recoge y se almacena temporalmente, creándose una interrupción para enviar un mensaje *downlink* hacia el nodo final. Dentro de la creación de esa interrupción se hace uso de la función *_send_down_link(),* que se especifica en la Figura 6.22, como acción a ejecutar cuando se produzca la interrupción.

Finalmente, si alguno de los procesos descritos falla, se muestran y envían mensajes de error. El último paso consiste en cerrar el *socket* y finalizar el *thread* UDP.

Como se puede observar en la Figura 6.18, los mensajes de error se envían haciendo uso de la función *_ack_pull_rsp()*, que se muestra en la Figura 6.23.

Por otra parte, en la Figura 6.20 se muestra la función *_freq_to_float(),* que se encarga de transformar la frecuencia que se pasa por parámetro, en una variable de tipo *float* con un mecanismo que evita imprecisiones.

En la función *_make_node_packet()* de la Figura 6.21 se rellena la estructura RX_PK con todos los valores relacionados con la recepción de un paquete.

```
                        else:
                            ack_error = TX_ERR_TOO_LATE
                            self._log('Downlink timestamp error!, t_us: {}', t_us)
                        self._ack_pull_rsp(_token, ack_error)
                        self._log("Pull rsp")
                except usocket.timeout:
                    pass
                except OSError as ex:
                    if ex.errno != errno.EAGAIN:
                        self._log('UDP recv OSError Exception: {}', ex)
                except Exception as ex:
                    self._log('UDP recv Exception: {}', ex)

                # wait before trying to receive again
                utime.sleep_ms(UDP_THREAD_CYCLE_MS)

            # we are to close the socket
            self.sock.close()
            self.udp_stop = False
            self._log('UDP thread stopped')
```

Figura 6.18: Función _udp_thread()_3.

```
def _lora_cb(self, lora):
    """
    LoRa radio events callback handler.
    """

    events = lora.events()
    if events & LoRa.RX_PACKET_EVENT:
        self._log('*** LoRa.RX_PACKET_EVENT')
        self.rxnb += 1
        self.rxok += 1
        rx_data = self.lora_sock.recv(256)
        stats = lora.stats()
        packet = self._make_node_packet(rx_data, self.rtc.now(), stats.rx_timestamp,
        stats.sfrx, self.bw, stats.rssi, stats.snr)
        self._push_data(packet)
        self._log('Received packet: {}', packet)
        self._log('Received packet rx_data: {}'.format(binascii.hexlify(rx_data)))
        self.rxfw += 1
    if events & LoRa.TX_PACKET_EVENT:
        self._log('*** LoRa.TX_PACKET_EVENT')
        self.txnb += 1
        lora.init(mode=LoRa.LORA,frequency=self.frequency,bandwidth=self.bw,
            sf=self.sf,preamble=8,coding_rate=LoRa.CODING_4_5,tx_iq=True)
```

Figura 6.19: Función _lora_cb().

```
def _freq_to_float(self, frequency):
    """
    MicroPython has some inprecision when doing large float division.
    To counter this, this method first does integer division until we
    reach the decimal breaking point. This doesn't completely elimate
    the issue in all cases, but it does help for a number of commonly
    used frequencies.
    """

    divider = 6
    while divider > 0 and frequency % 10 == 0:
        frequency = frequency // 10
        divider -= 1
    if divider > 0:
        frequency = frequency / (10 ** divider)
    return frequency
```

Figura 6.20: Función _freq_to_float().

```
def _make_node_packet(self, rx_data, rx_time, tmst, sf, bw, rssi, snr):
    RX_PK["rxpk"][0]["time"] = "%d-%02d-%02dT%02d:%02d:%02d.%dZ" % (rx_time[0],
    rx_time[1], rx_time[2], rx_time[3], rx_time[4], rx_time[5], rx_time[6])
    RX_PK["rxpk"][0]["tmst"] = tmst
    RX_PK["rxpk"][0]["freq"] = self._freq_to_float(self.frequency)
    RX_PK["rxpk"][0]["datr"] = self._sf_bw_to_dr(sf, bw)
    RX_PK["rxpk"][0]["rssi"] = rssi
    RX_PK["rxpk"][0]["lsnr"] = snr
    RX_PK["rxpk"][0]["data"] = ubinascii.b2a_base64(rx_data)[:-1]
    RX_PK["rxpk"][0]["size"] = len(rx_data)
    return ujson.dumps(RX_PK)
```

Figura 6.21: Función _make_node_packet().

En la Figura 6.22 se muestra la función *_send_down_link(),* que se encarga de enviar datos al nodo final. Para llevar a cabo esta tarea, en primer lugar, inicializa la comunicación *LoRa* con los parámetros configurados, y posteriormente envía los datos.

En la función *_ack_pull_rsp(),* mostrada en la Figura 6.23, se rellena la estructura TX_ACK_PK con el error que se ha producido, y se envía al servidor IP.

```
def _send_down_link(self, data, tmst, datarate, frequency):
    """
    Transmits a downlink message over LoRa.
    """

    self.lora.init(
        mode=LoRa.LORA,
        frequency=frequency,
        bandwidth=self._dr_to_bw(datarate),
        sf=self._dr_to_sf(datarate),
        preamble=8,
        coding_rate=LoRa.CODING_4_5,
        tx_iq=True
        )
    while utime.ticks_us() < tmst:
        pass
    self.lora_sock.send(data)
    self._log(
        '!!! Sent downlink packet scheduled on {:.3f}, at {:.1f} Mhz using {}: {}',
        tmst / 1000000,
        self._freq_to_float(frequency),
        datarate,
        data
    )
```

Figura 6.22: Función _send_down_link().

```
def _ack_pull_rsp(self, token, error):
    TX_ACK_PK["txpk_ack"]["error"] = error
    resp = ujson.dumps(TX_ACK_PK)
    packet = bytes([PROTOCOL_VERSION]) + token + bytes([PULL_ACK]) + ubinascii.unhexlify(self.id) + resp
    with self.udp_lock:
        try:
            self.sock.sendto(packet, self.server_ip)
        except Exception as ex:
            self._log('PULL RSP ACK exception: {}', ex)
```

Figura 6.23: Función _ack_pull_rsp().

Tras haber descrito dos de las funciones principales, *__init__()* y *start()*, a continuación, se explica una tercera función principal, denominada *stop()*. En la Figura 6.24 se adjunta el código correspondiente a esta función.

En la función *stop()* de la Figura 6.24 se detienen todos los procesos que se están ejecutando. Se indica la parada de *LoRa*, con la opción de *Sleep*, se cancelan todas las

interrupciones, se detiene el sincronismo del RTC, se indica al *thread* UDP que se detenga, y se deshabilita la conexión *WiFi*.

Con esta función se finaliza el análisis del archivo *nanogateway.py*

```
def stop(self):
    """
    Stops the LoRaWAN nano gateway.
    """

    self._log('Stopping...')

    # send the LoRa radio to sleep
    self.lora.callback(trigger=None, handler=None)
    self.lora.power_mode(LoRa.SLEEP)

    # stop the NTP sync
    self.rtc.ntp_sync(None)

    # cancel all the alarms
    self.stat_alarm.cancel()
    self.pull_alarm.cancel()

    # signal the UDP thread to stop
    self.udp_stop = True
    while self.udp_stop:
        utime.sleep_ms(50)

    # disable WLAN
    self.wlan.disconnect()
    self.wlan.deinit()
```

Figura 6.24: Función stop().

6.2 Archivo: `config.py`

En este apartado se procede a comentar el archivo *config.py*, cuyo código se muestra en la Figura 6.25. En la Figura 6.25 se pueden observar los parámetros de configuración que se pueden editar en función de las particularidades de cada *nanogateway*. Es fundamental modificar las variables WiFi_SSID y WiFi_PASS que vienen por defecto para poder establecer la conexión *WiFi* con la red que se desee. Por otra parte, también es importante

introducir en la variable GATEWAY_ID el identificador del *gateway* registrado en la plataforma *The Things Network*. Además, los datos se envían desde el *gateway* hacia el servidor, por lo que se debe indicar a qué servidor se enviarán los datos. La dirección del servidor, que se recoge en la variable SERVER, pertenece a TTN.

```
""" LoPy LoRaWAN Nano Gateway configuration options """

GATEWAY_ID = '70B3D54999D4C26B'

SERVER = 'router.eu.thethings.network'
PORT = 1700

NTP = "pool.ntp.org"
NTP_PERIOD_S = 3600

WIFI_SSID = 'ULPGC'
WIFI_PASS = ''

LORA_FREQUENCY = 868100000
LORA_GW_DR = "SF7BW125" # DR_5
LORA_NODE_DR = 5
```

Figura 6.25: config.py.

6.3 Archivo: `main.py`

Por último, en la Figura 6.26 se adjunta el código del archivo *main.py*, en el que se crea un objeto de la clase NanoGateway y se le asignan los parámetros de configuración establecidos en el archivo *config.py*. Seguidamente, se llama a la función *start()* para iniciar así el funcionamiento del *nanogateway*.

```
""" LoPy LoRaWAN Nano Gateway example usage """

import config
from nanogateway import NanoGateway

if __name__ == '__main__':
    nanogw = NanoGateway(
        id=config.GATEWAY_ID,
        frequency=config.LORA_FREQUENCY,
        datarate=config.LORA_GW_DR,
        ssid=config.WIFI_SSID,
        password=config.WIFI_PASS,
        server=config.SERVER,
        port=config.PORT,
        ntp_server=config.NTP,
        ntp_period=config.NTP_PERIOD_S
        )

    nanogw.start()
    nanogw._log('You may now press ENTER to enter the REPL')
input()
```

Figura 6.26: main.py.

Capítulo 7. Estación Base

El nodo estación base realiza la función de intermediario entre el nodo final y el nodo *gateway* o concentrador. Es decir, la información que reciba desde el nodo final mediante *BLE* o *VLC* se retransmite hacia el nodo *gateway* usando la tecnología de comunicación *LoRa/LoRaWAN*. Del mismo modo, pero en sentido contrario de la comunicación, este nodo puede recibir información desde el nodo *gateway* mediante *LoRa/LoRaWAN* y retransmitirla hacia el nodo final utilizando *BLE* o *VLC*. En la Figura 7.1 se muestra la ubicación de este nodo en la arquitectura de la plataforma propuesta. En este caso, el nodo puede ser alimentado de forma ininterrumpida. No obstante, se utilizan tecnologías de bajo consumo de potencia con el fin de hacer un uso energéticamente eficiente del nodo.

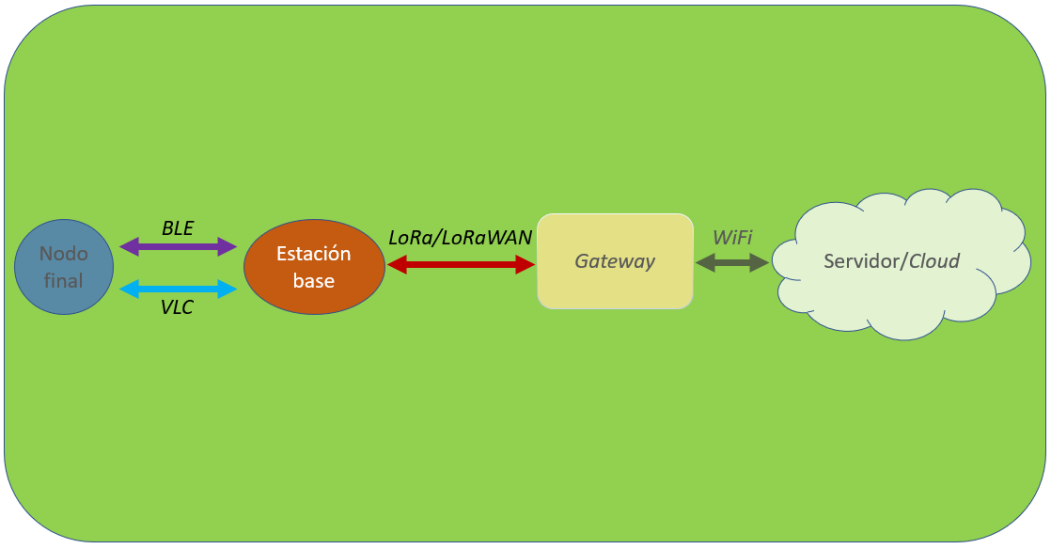


Figura 7.1: Arquitectura de la plataforma HW/SW propuesta: Elemento Estación Base.

La implementación de la funcionalidad del nodo estación base, al igual que en el caso del nodo *gateway*, se realizará sobre el dispositivo LoPy, de la empresa Pycom. Como se pudo ver en el capítulo 3, este dispositivo permite integrar las tres tecnologías de comunicación simultáneamente, y con un planteamiento de ahorro de consumo de potencia. A continuación, se procede a explicar la codificación de esta funcionalidad.

Primeramente, se deben incluir las librerías requeridas para el correcto desarrollo de la funcionalidad. En la Figura 7.2 se muestran las librerías utilizadas en el nodo estación base.

```
2  from network import LoRa
3  import socket
4  import machine
5  from machine import Timer
6  import time
7  from machine import UART
8  from network import Bluetooth
9  import pycom
10 import binascii
11 from machine import RTC
12 from array import array
```

Figura 7.2: Librerías utilizadas en el nodo estación base.

Tal y como se explicó en el capítulo 5, antes de realizar comunicaciones con la tecnología *LoRa/LoRaWAN* se debe iniciar un proceso de activación para introducir al nodo en la red *LoRaWAN*. Para ello, se recuerda que se requerían tres claves: *DevEUI*, *AppEUI* y *AppKey*. La primera de ellas se obtiene con el comando:

```
binascii.hexlify(network.LoRa().mac()),
```

mientras que las dos siguientes se obtienen en la plataforma TTN al registrar el dispositivo que se desea conectar.

En la Figura 7.3 se muestra la configuración que se explicó en el capítulo 5, pero con las claves correspondientes al nodo estación base. En la misma Figura 7.3, se realiza la activación y se espera hasta que se confirme la operación. Seguidamente, como se puede ver en la Figura 7.4, se borran los canales que no pertenecen a los denominados “por defecto” y además de crea un *socket* de comunicación. Con estas acciones se completa la configuración de la comunicación *LoRa/LoRaWAN*.

```
16  ########################################Configuración LoRa/LoRaWAN#####################################
17  # Se inicializa LoRa en el modo LoRaWAN
18  # Se pueden añadir más parámteros como la frecuencia, la potencia de transmisión o el Spreading Factor
19  lora = LoRa(mode=LoRa.LORAWAN)
20
21  # Se crean los parámteros de autenticación para la activación OTAA (nueva App)
22  dev_eui = binascii.unhexlify('70B3D5499A472710'.replace(' ','')) #estacion base
23  app_key = binascii.unhexlify('9F710CB8AE2C64C4AB1B269FBCB11D20'.replace(' ','')) #estacion base -Lopy tfm
24  app_eui = binascii.unhexlify('70B3D57ED002974D'.replace(' ','')) #Lopy-tfm2
25
26  # Se dejan los 3 canales por defecto en la misma frecuencia. (Se debe hacer antes del join-request)
27  lora.add_channel(0, frequency=868100000, dr_min=0, dr_max=5)
28  lora.add_channel(1, frequency=868100000, dr_min=0, dr_max=5)
29  lora.add_channel(2, frequency=868100000, dr_min=0, dr_max=5)
30
31  # Se envía el join-request a la red vía OTAA
32  lora.join(activation=LoRa.OTAA, auth=(dev_eui, app_eui, app_key), timeout=0, dr=5)
33
34  # Se mantiene en espera hasta que se acepte el join.request
35  while not lora.has_joined():
36      time.sleep(2.5)
37      print('Codigo actualizado-11')
38      print('Not joined yet...')
39
40  print('Joined!')
```

Figura 7.3: Configuración de LoRa/LoRaWAN en el nodo estación base.

```
42  # Se borran todos los canales que no sean por defecto
43  for i in range(3, 16):
44      lora.remove_channel(i)
45
46  # Se crea un socket LoRa
47  s = socket.socket(socket.AF_LORA, socket.SOCK_RAW)
48
49  # Se configura el data rate de LoRaWAN
50  s.setsockopt(socket.SOL_LORA, socket.SO_DR, 5)
```

Figura 7.4: Configuración de socket LoRa/LoRaWAN en el nodo estación base.

En la Figura 7.5, se muestra el punto del código en el que se fija el tipo de comunicación que se desea en la transmisión y en la recepción de datos para el nodo en cuestión. Tal y como se aprecia en la Figura 7.5, existen tres posibilidades de configuración para cada sentido de la comunicación. Las tres posibilidades en cada uno de los dos sentidos de comunicación (transmisión y recepción) son:

- *VLC*: Código 0x01
- *BLE*: Código 0x02
- *BLE y VLC* simultáneamente: Código 0x03

En función del código seleccionado se realiza la configuración e inicialización *VLC*, *BLE* o ambas. En la misma Figura 7.5, se aprecia la configuración *VLC* en la que como se vio en el capítulo 3, se trata de configurar la UART en los pines 'P3' y 'P4' con 9600 baudios, 8 bits de datos, sin bit de paridad y 1 bit de stop. Por otra parte, en la configuración *BLE*, acorde a lo estudiado en el capítulo 4, simplemente se crea un objeto *Bluetooth*. Además de las configuraciones anteriores, también se muestra un mensaje en el terminal que indique la tecnología de comunicación escogida tanto para la transmisión como para la recepción.

```
#Selección tecnología de trasmisión
techselectedTX = 0x02 #BLE          0x01:VLC; 0x02: BLE; 0x03: VLC+BLE simultáneo
#Selección tecnología de recepción
techselectedRX = 0x01 #VLC          0x01:VLC; 0x02: BLE; 0x03: VLC+BLE simultáneo

########################################Configuración UART-VLC#################################
#Si se selecciona la tecnología VLC o ambas (BLE también)
if((techselectedRX == 0x01) | (techselectedRX == 0x03)):
    print('VLC configurado como receptor')
    #Configuración UART
    uart = UART(1, baudrate=9600, pins=('P3','P4'))   # Inicialización baudrate y pines físicos
    uart.init(9600, bits=8, parity=None, stop=1) #96008N1
if((techselectedTX == 0x01) | (techselectedTX == 0x03)):
    print('VLC configurado como transmisor')
    #Configuración UART
    uart = UART(1, baudrate=9600, pins=('P3','P4'))   # Inicialización baudrate y pines físicos
    uart.init(9600, bits=8, parity=None, stop=1) #96008N1
########################################Configuración BLE#####################################
#Si se selecciona la tecnología BLE o ambas (VLC también)
if((techselectedRX == 0x02) | (techselectedRX == 0x03)):
    print('BLE configurado como receptor')
    #Configuración BLE
    bt = Bluetooth()
if((techselectedTX == 0x02) | (techselectedTX == 0x03)):
    print('BLE configurado como transmisor')
    #Configuración BLE
    bluetooth = Bluetooth()
```

Figura 7.5: Configuración de comunicación VLC y BLE en el nodo estación base.

Una vez que se han configurado las tecnologías de comunicación para la transmisión y la recepción, se procede a la creación de variables de tipo *bytearray* que almacenarán los datos que se van a comunicar. En la Figura 7.6 se muestra la declaración de dos variables de este tipo. La variable 'frame' será la que almacene los datos que se van a comunicar siempre en sentido ascendente en la arquitectura (desde el nodo final hasta TTN), mientras que la variable 'bytearr' está destinada al almacenamiento de los datos que se van a comunicar en sentido descendente en la arquitectura (desde TTN hasta el nodo final). Se declaran estos *array* de bytes con un tamaño de 11 bytes, pero en función de la aplicación específica que se le vaya a dar a la plataforma genérica este tamaño puede ser modificado.

En la Figura 7.6 también se puede ver la declaración de dos funciones idénticas que permiten borrar los datos de estas variables y poner cada byte a un valor nulo. La función *clearFrame()* se encarga de limpiar la variable 'frame', y la función *clear_bytearr()* se encarga de limpiar la variable 'bytearr'.

Con respecto a la transmisión hacia el nodo final, esta se podía realizar mediante *VLC* o mediante *BLE*. Según la configuración escogida se envía con una tecnología o con otra, o incluso se da la posibilidad de enviar con las dos simultáneamente con el fin de realizar pruebas, ya que no sería óptimo salvo en aplicaciones que requieran de mayor robustez. En la Figura 7.7 se muestra la decisión de enviar un dato por la tecnología de comunicación escogida. Simplemente, en la función *send()*, se redirige el flujo del programa hacia la función *send_BLE()* o hacia *send_VLC()*.

```
86  ##################################################Contenido de la trama######
87  #Se crea un bytearray para la gestión de las tramas
88  frame = bytearray(11)   #Recibido VLC/BLE y transmitido con LoRa/LoRaWAN
89  bytearr = bytearray(11) #Recibido desde LoRa/LoRaWAN y transmitido con BLE/VLC
90
91  ##################################################Funciones##################
92  #Función para borrar los campos de las tramas
93  def clearframe():
94      global frame
95      for i in range(0, 11):
96       frame[i] = frame[i] & 0x00
97
98  #Función para borrar los campos de las tramas
99  def clear_bytearr():
100     global data
101     for i in range(0, 11):
102      bytearr[i] = bytearr[i] & 0x00
103
```

Figura 7.6: Declaración de bytearrays y de funciones clearframe() y clear_bytearr().

La función *send_VLC()* de la Figura 7.7 llama al objeto UART creado y escribe en él la variable correspondiente. En esa misma Figura 7.7 se muestra la función *send_BLE()*, que como se estudió en el capítulo 4, comienza un proceso de *advertisement* y seguidamente envía el Servicio y la Característica correctamente identificados y con el valor de la variable que almacena los datos que se desean enviar, la variable 'bytearr'.

Del mismo modo, en la Figura 7.8 se realiza la misma acción, pero para la recepción de datos. En función de la tecnología de comunicación escogida, en la función *receive()* se

redirige el flujo del programa hacia la función *receive_VLC()* o hacia la función *receive_BLE()*.

La función *receive_VLC()* de la Figura 7.8, se encarga de realizar una lectura de la UART y se almacena el valor leído en la variable de comunicación ascendente 'frame'. Además, muestra por pantalla en el terminal el contenido de esta variable.

La función *receive_BLE()* de la Figura 7.9, realiza una fase de escaneo, en la que se encuentra en modo escucha esperando por la recepción de un *advertisement*, y una vez que lo recibe y lo identifica como el *advertisement* esperado, comienza con el proceso de conexión. Siguiendo el mismo procedimiento que se explicó en el capítulo 4, se leen los Servicios y Características, y de este último se almacenan los datos recibidos en la variable de comunicación ascendente 'frame'.

```
104 #Función que selecciona la tecnología de transmisión
105 def send():
106     if((techselectedTX == 0x01) | (techselectedTX == 0x03)): #VLC
107         send_VLC()
108     if((techselectedTX == 0x02) | (techselectedTX == 0x03)): #BLE
109         send_BLE()
110
111 #Función que realiza transmisiones vía VLC
112 def send_VLC():
113     global bytearr
114     uart.write(bytearr)
115
116
117 #Función que realiza transmisiones vía BLE
118 def send_BLE():
119     global bluetooth
120     global bytearr
121
122     #bt = Bluetooth()
123     bluetooth = Bluetooth()
124     bluetooth.set_advertisement(name='LoPy_Server', service_uuid=b'1234567890123456')
125
126     bluetooth.advertise(True)
127
128     srv1 = bluetooth.service(uuid=b'1234567890123456', isprimary=True)
129     print('no enviada caracteristica')
130
131     chr1 = srv1.characteristic(uuid=b'ab34567890123456', value=bytearr)
132     print('enviado')
133
134     char1_read_counter = 0
```

Figura 7.7: Funciones send(), send_VLC() y send_BLE().

```
#Función que selecciona la tecnología de transmisión
def receive():
    if((techselectedRX == 0x01) | (techselectedRX == 0x03)): #VLC
       receive_VLC()
    if((techselectedRX == 0x02) | (techselectedRX == 0x03)): #BLE
       receive_BLE()

#Función que realiza recepiones vía VLC
def receive_VLC():
    global frame
    uart.readinto(frame) #Se almacenan los datos en un bytearray
    print("- RECEIVED data from end node = {}".format(binascii.hexlify(frame)))

```

Figura 7.8: Funciones receive() y receive_VLC().

```
#Función que realiza recepiones vía BLE
def receive_BLE():
    global data
    global frame
    #global flag_scan
    #if(flag_scan):
    scan_BLE()
    adv = bt.get_adv()  #se almacenan hasta 16 advertisements, se llama a uno de ellos
    if adv and bt.resolve_adv_data(adv.data, Bluetooth.ADV_NAME_CMPL) == 'LoPy_Server':
        try:
            conn = bt.connect(adv.mac)
            services = conn.services()
            print(services)
            for service in services:
                print(service.uuid())
                time.sleep(0.050)
                if type(service.uuid()) == bytes:
                    print('Reading chars from service = {}'.format(service.uuid()))
                else:
                    print('Reading chars from service = %x' % service.uuid())
                chars = service.characteristics()
                for char in chars:
                    if (char.properties() & Bluetooth.PROP_READ):
                        print('char {} value = {}'.format(char.uuid(), char.read()))
                        data = char.read()
                        print(data)
                        #flag_scan=1
                        #Se almacenan los datos en un bytearray
                        frame = bytearray(data)
            conn.disconnect()
            #break
        except:
            print("Error while connecting or reading from the BLE device")
            #break
    else:
        time.sleep(0.050)
```

Figura 7.9: Función receive_BLE().

Ya se han visto las funciones de transmisión y recepción que permiten la comunicación con el nodo final, así que sólo queda presentar las funciones de transmisión y recepción que

permiten al nodo estación base comunicarse con el nodo *gateway*. Esta comunicación, como se explicó anteriormente, está basada en *LoRa/LoRaWAN*.

```
192  #Función que realiza una transmisión LoRa/LoRaWAN
193  def send_LoRa():
194      # Se desbloquea el socket
195      s.setblocking(True)   #Para poder enviar
196      s.send(frame)
197  #Función que realiza una recepción LoRa/LoRaWAN
198  def receive_LoRa():
199      #Recepción de la trama de configuración
200      #Se bloquea el socket para poder recibir
201      s.setblocking(False)
202      #Se realiza la recepción
203      data = s.recv(64)
204      #Se almacenan los datos en un bytearray
205      bytearr = bytearray(data)
206       #Se imprime el bytearray recibido en la pantalla
207      print("- RECEIVED data from TTN = {}".format(binascii.hexlify(bytearr)))
```

Figura 7.10: Funciones send_LoRa() y receive_LoRa().

En la Figura 7.10 se muestran las funciones asociadas a este tipo de comunicación. Por una parte, la función *send_LoRa()* simplemente abre el *socket* creado anteriormente en la configuración *LoRa/LoRaWAN* y envía el dato por este. Por otra parte, la función *receive_LoRa()* bloquea el *socket* y recibe por este. El valor leído se almacena en la variable de comunicación descendente 'bytearr'. Además, se muestra por pantalla en el terminal el valor recibido.

Llegados a este punto del *firmware*, con todas las funciones declaradas y las configuraciones de las distintas tecnologías de configuración completadas, queda desarrollar el uso de todas estas posibilidades en un bucle infinito. Este desarrollo dependerá de la aplicación específica que se le dé a la plataforma genérica presentada, pero su funcionamiento imprescindible, aunque optimizable, se presenta en la Figura 7.11.

```
210  while 1:
211      receive()
212      if((frame[5] == 0)): #poner byte del id que siempre esta ocupado
213          print(" Recibido frame vacio, no se reenvía por LoRa ")
214      else:
215          send_LoRa()
216          clearframe() #Se limpia el frame
217      receive_LoRa()
218      if(len(bytearr) > 0):
219          send()
220          clear_bytearr() #Se limpia el bytearray
```

Figura 7.11: Bucle infinito del nodo estación base.

El flujo seguido en la Figura 7.11 es el siguiente: Primero se deben recibir datos desde el nodo final (usando la función *receive()*), después se debe valorar si el contenido de estos datos es nulo o incorrecto, y en caso de que contenga información, se reenvía mediante la función *send_LoRa()* hacia el nodo *gateway*. Seguidamente se borra el contenido de la variable 'frame' para que pueda volver a ser utilizada. En caso de que los datos recibidos sean de valor nulo simplemente se muestra un aviso por pantalla en el terminal. La valoración de que los datos contengan valor nulo o incorrecto se ha realizado en este ejemplo, mediante la comprobación del byte 5, ya que según las pruebas realizadas en el capítulo 3 y en el capítulo 4, este valor nunca es nulo. No obstante, dependiendo de la aplicación específica que se implemente en la plataforma genérica, este tipo de valoración puede cambiar.

Del mismo modo, y tras realizar la comunicación en sentido ascendente en la arquitectura, a continuación se realiza una recepción *LoRa/LoRaWAN* mediante la función *receive_LoRa()*, si el contenido de este envío es no nulo, pues se reenvía el dato recibido hacia el nodo final llamando a la función *send()*. En caso de que se haya realizado un envío con esta función, se procede a borrar el contenido de la variable 'bytearr' con el fin de poder utilizarla de nuevo.

Capítulo 8. Nodo final

El nodo final, mediante el uso de un sensor integrado recoge información del entorno, y haciendo uso de *BLE* o *VLC* la transfiere al nodo estación base. Del mismo modo, pero en sentido contrario de la comunicación, este nodo puede recibir información desde la estación base mediante *BLE* o *VLC*. En la Figura 8.1 se muestra la ubicación de este nodo en la arquitectura de la plataforma propuesta. Este nodo es alimentado por batería, por lo que su funcionalidad debe acotarse a restricciones de consumo de potencia.

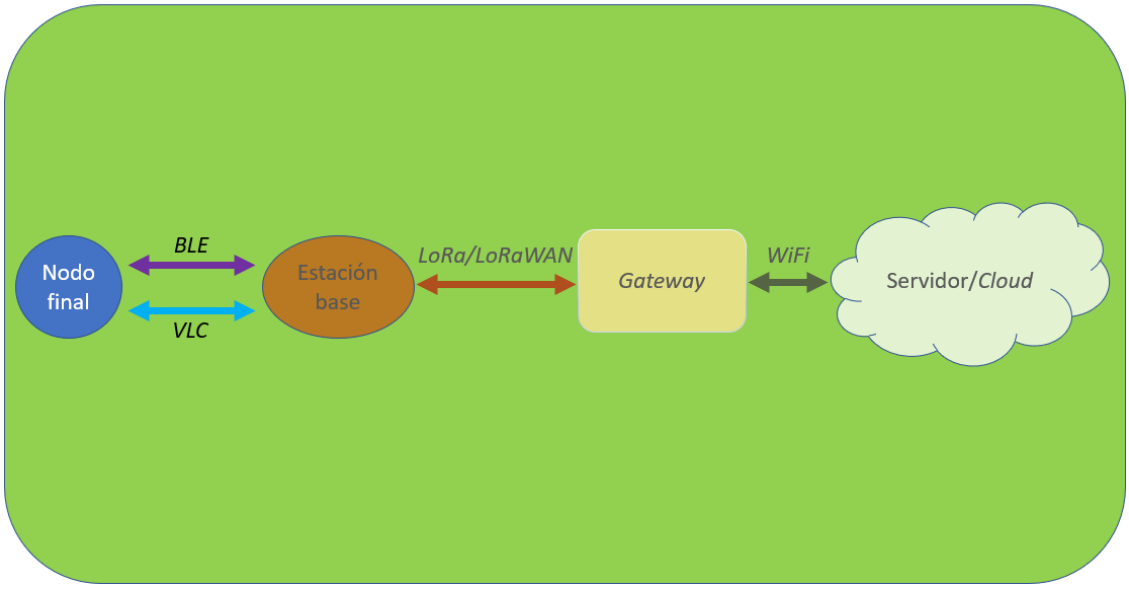


Figura 8.1: Arquitectura de la plataforma HW/SW propuesta: Elemento Nodo Final.

La implementación de la funcionalidad del nodo final, al igual que en el caso del nodo *gateway* y del nodo estación base, se realizará sobre el dispositivo LoPy, de la empresa

Pycom. Se debe aclarar que el nodo final debe contar con un sensor para recoger información del entorno y trasmitir esa información al resto de la plataforma IoT. No obstante, la selección del sensor depende de cada aplicación específica que se vaya a implementar en la plataforma genérica, por lo que en la descripción de la funcionalidad no se presenta información relativa a este elemento del nodo final. En el capítulo 10, en el que se trata un *use of case* de esta plataforma genérica, se mostrará un ejemplo de integración del sensor en el nodo final. Del mismo modo, en relación con la batería que alimenta a este nodo final, la selección de una batería determinada también depende de la aplicación concreta que se vaya a implementar, por lo que su selección queda abierta en este punto de la memoria del TFM. Al igual que en el caso del sensor, en el capítulo 10, en el *use of case* planteado se propondrá una batería acorde a las características de la aplicación planteada.

A continuación, se procede a explicar la codificación de esta funcionalidad. Primeramente, se deben incluir las librerías requeridas para el correcto desarrollo de la funcionalidad. En la Figura 8.2 se muestran las librerías usadas en el nodo estación base.

```
2  from network import LoRa
3  import socket
4  import machine
5  from machine import Timer
6  import time
7  from machine import UART
8  from network import Bluetooth
9  import pycom
10 import binascii
11 from machine import RTC
12 from array import array
```

Figura 8.2: Librerías utilizadas en el nodo final.

Tanto en el nodo final como en el nodo estación base se implementa la misma funcionalidad con respecto a las tecnologías de comunicación *BLE* y *VLC*.

En la Figura 8.3 se muestra el punto del código en el que se fija el tipo de comunicación que se desea en la transmisión y en la recepción de datos para el nodo en cuestión. Tal y como se aprecia en la Figura 8.3, existen tres posibilidades de configuración para cada sentido de la comunicación. Las tres posibilidades en cada uno de los dos sentidos de comunicación (transmisión y recepción) son:

- *VLC*: Código 0x01
- *BLE*: Código 0x02
- *BLE* y *VLC* simultáneamente: Código 0x03

En función del código seleccionado se realiza la configuración e inicialización *VLC*, *BLE* o ambas. En la misma Figura 8.3 se aprecia la configuración *VLC,* en la que como se vio en el capítulo 3, se trata de configurar la UART en los pines 'P3' y 'P4' con 9600 baudios, 8 bits de datos, sin bit de paridad y 1 bit de stop. Por otra parte, en la configuración *BLE*, acorde a lo estudiado en el capítulo 4, simplemente se crea un objeto *Bluetooth*. Además de las configuraciones anteriores, también se muestra un mensaje en el terminal que indique la tecnología de comunicación escogida tanto para la transmisión como para la recepción.

```
#Selección tecnología de trasmisión
techselectedTX = 0x02 #BLE          0x01:VLC; 0x02: BLE; 0x03: VLC+BLE simultáneo
#Selección tecnología de recepción
techselectedRX = 0x01 #VLC          0x01:VLC; 0x02: BLE; 0x03: VLC+BLE simultáneo

########################################Configuración UART-VLC#################################
#Si se selecciona la tecnología VLC o ambas (BLE también)
if((techselectedRX == 0x01) | (techselectedRX == 0x03)):
    print('VLC configurado como receptor')
    #Configuración UART
    uart = UART(1, baudrate=9600, pins=('P3','P4'))   # Inicialización baudrate y pines físicos
    uart.init(9600, bits=8, parity=None, stop=1) #96008N1
if((techselectedTX == 0x01) | (techselectedTX == 0x03)):
    print('VLC configurado como transmisor')
    #Configuración UART
    uart = UART(1, baudrate=9600, pins=('P3','P4'))   # Inicialización baudrate y pines físicos
    uart.init(9600, bits=8, parity=None, stop=1) #96008N1
########################################Configuración BLE#####################################
#Si se selecciona la tecnología BLE o ambas (VLC también)
if((techselectedRX == 0x02) | (techselectedRX == 0x03)):
    print('BLE configurado como receptor')
    #Configuración BLE
    bt = Bluetooth()
if((techselectedTX == 0x02) | (techselectedTX == 0x03)):
    print('BLE configurado como transmisor')
    #Configuración BLE
    bluetooth = Bluetooth()
```

Figura 8.3: Configuración VLC y BLE en el nodo final.

Una vez que se han configurado las tecnologías de comunicación para la transmisión y la recepción, se procede a la creación de variables de tipo *bytearray* que almacenarán los datos que se van a comunicar. En la Figura 8.4 se muestra la declaración de dos variables de este tipo. La variable 'frame' será la que almacene los datos que se van a comunicar siempre en sentido ascendente en la arquitectura (desde el nodo final hasta TTN), mientras

que la variable 'bytearr' está destinada al almacenamiento de los datos que se van a comunicar en sentido descendente en la arquitectura (desde TTN hasta el nodo final). Se declaran estos *array* de bytes con un tamaño de 11 bytes, pero en función de la aplicación específica que se le vaya a dar a la plataforma genérica este tamaño puede ser modificado.

En la Figura 8.4 también se puede ver la declaración de dos funciones idénticas que permiten borrar los datos de estas variables y poner cada byte a un valor nulo. La función *clearFrame()* se encarga de limpiar la variable 'frame', y la función *clear_bytearr()* se encarga de limpiar la variable 'bytearr'.

```
86  ##########################################################Contenido de la trama#######
87  #Se crea un bytearray para la gestión de las tramas
88  frame = bytearray(11)   #Recibido VLC/BLE y transmitido con LoRa/LoRaWAN
89  bytearr = bytearray(11) #Recibido desde LoRa/LoRaWAN y transmitido con BLE/VLC
90
91  ##########################################################Funciones##################
92  #Función para borrar los campos de las tramas
93  def clearframe():
94      global frame
95      for i in range(0, 11):
96       frame[i] = frame[i] & 0x00
97
98  #Función para borrar los campos de las tramas
99  def clear_bytearr():
100     global data
101     for i in range(0, 11):
102      bytearr[i] = bytearr[i] & 0x00
103
```

Figura 8.4: Declaración de bytearrays y de funciones clearframe() y clear_bytearr().

Con respecto a la transmisión hacia el nodo estación base, esta se podía realizar mediante *VLC* o mediante *BLE*. Según la configuración escogida se envía con una tecnología o con otra, o incluso se da la posibilidad de enviar con las dos simultáneamente con el fin de realizar pruebas, ya que no sería óptimo salvo en aplicaciones que requieran de mayor robustez. En la Figura 8.5 se muestra la decisión de enviar un dato con la tecnología de comunicación escogida. Simplemente, en la función *send()*, se redirige el flujo del programa hacia la función *send_BLE()* o hacia *send_VLC()*.

La función *send_VLC()* de la Figura 8.5 llama al objeto UART creado y escribe en él la variable correspondiente. En esa misma Figura 8.5 se muestra la función *send_BLE()*, que como se

estudió en el capítulo 4, comienza un proceso de *advertisement* y seguidamente envía el Servicio y la Característica correctamente identificados y con el valor de la variable que almacena los datos que se desean enviar, la variable 'bytearr'.

```
#Función que selecciona la tecnología de transmisión
def send():
    if((techselectedTX == 0x01) | (techselectedTX == 0x03)): #VLC
       send_VLC()
    if((techselectedTX == 0x02) | (techselectedTX == 0x03)): #BLE
       send_BLE()

#Función que realiza transmisiones vía VLC
def send_VLC():
    global bytearr
    uart.write(bytearr)


#Función que realiza transmisiones vía BLE
def send_BLE():
    global bluetooth
    global bytearr

    #bt = Bluetooth()
    bluetooth = Bluetooth()
    bluetooth.set_advertisement(name='LoPy_Server', service_uuid=b'1234567890123456')

    bluetooth.advertise(True)

    srv1 = bluetooth.service(uuid=b'1234567890123456', isprimary=True)
    print('no enviada caracteristica')

    chr1 = srv1.characteristic(uuid=b'ab34567890123456', value=bytearr)
    print('enviado')

    char1_read_counter = 0
```

Figura 8.5: Funciones send(), send_VLC() y send_BLE().

Del mismo modo, en la Figura 8.6 se realiza la misma acción, pero para la recepción de datos. En función de la tecnología de comunicación escogida, en la función *receive()* se redirige el flujo del programa hacia la función *receive_VLC()* o hacia la función *receive_BLE()*.

La función *receive_VLC()* de la Figura 8.6, se encarga de realizar una lectura de la UART y se almacena el valor leído en la variable de comunicación ascendente 'frame'. Además, muestra por pantalla en el terminal el contenido de esta variable.

La función *receive_BLE()* de la Figura 8.7, realiza una fase de escaneo, dónde se encuentra en modo escucha esperando por la recepción de un *advertisement* y una vez que lo recibe

y lo identifica como el *advertisement* esperado, comienza con el proceso de conexión. Siguiendo el mismo procedimiento que se explicó en el capítulo 4, se leen los Servicios y Características, y de este último se almacenan los datos recibidos en la variable de comunicación ascendente 'frame'.

```
#Función que selecciona la tecnología de transmisión
def receive():
    if((techselectedRX == 0x01) | (techselectedRX == 0x03)): #VLC
       receive_VLC()
    if((techselectedRX == 0x02) | (techselectedRX == 0x03)): #BLE
       receive_BLE()

#Función que realiza recepiones vía VLC
def receive_VLC():
    global frame
    uart.readinto(frame) #Se almacenan los datos en un bytearray
    print("- RECEIVED data from end node = {}".format(binascii.hexlify(frame)))

```

Figura 8.6: Funciones receive() y receive_VLC().

```
#Función que realiza recepiones vía BLE
def receive_BLE():
    global data
    global frame
    #global flag_scan
    #if(flag_scan):
    scan_BLE()
    adv = bt.get_adv()  #se almacenan hasta 16 advertisements, se llama a uno de ellos
    if adv and bt.resolve_adv_data(adv.data, Bluetooth.ADV_NAME_CMPL) == 'LoPy_Server':
        try:
            conn = bt.connect(adv.mac)
            services = conn.services()
            print(services)
            for service in services:
                print(service.uuid())
                time.sleep(0.050)
                if type(service.uuid()) == bytes:
                    print('Reading chars from service = {}'.format(service.uuid()))
                else:
                    print('Reading chars from service = %x' % service.uuid())
                chars = service.characteristics()
                for char in chars:
                    if (char.properties() & Bluetooth.PROP_READ):
                        print('char {} value = {}'.format(char.uuid(), char.read()))
                        data = char.read()
                        print(data)
                        #flag_scan=1
                        #Se almacenan los datos en un bytearray
                        frame = bytearray(data)
            conn.disconnect()
            #break
        except:
            print("Error while connecting or reading from the BLE device")
            #break
    else:
        time.sleep(0.050)
```

Figura 8.7: Función receive_BLE().

Llegados a este punto del *firmware*, con todas las funciones declaradas y las configuraciones de las distintas tecnologías de configuración completadas, queda desarrollar el uso de todas estas posibilidades en un bucle infinito. Este desarrollo dependerá de la aplicación específica que se le dé a la plataforma genérica presentada, pero su funcionamiento imprescindible, aunque optimizable, se presenta en la Figura 8.8.

```
393  while 1:
394
395   consulta_sensor()
396   proceso_datos()
397   if(necesario_enviar):
398       send()
399   receive()
400   if(len(bytearr) > 0):
401       actualizar_config()
402       sleep(2)
```

Figura 8.8: Bucle infinito del nodo final.

El flujo seguido en la Figura 8.8 es el siguiente: en primer lugar se realiza una consulta al sensor y se procesan e interpreta el dato ofrecido por este. En función del valor leído en el sensor puede corresponder enviar los datos al nodo estación base o no. En caso de ser necesaria dicha transmisión se llama a la función *send()*, que enviará los datos correspondientes usando *BLE* o *VLC*.

Después se realiza un intento de recepción mediante la función *receive()*, después se debe valorar si el contenido de estos datos es nulo o incorrecto. En caso de que sí se contenga información, se procesa esa información en función de lo requerido por la aplicación concreta que ha sido implementada sobre la plataforma genérica. Un ejemplo de este procesado de información puede ser actualizar la configuración del nodo final. No obstante, en el capítulo 10, en el *use of case* presentado, se muestra un ejemplo de este tipo con más detalle.

Capítulo 9. The Things Network

En el presente TFM, se ha elegido *The Things Network* (TTN) como plataforma para desarrollar la aplicación que permitirá gestionar la información recibida/enviada desde/hacia los nodos finales. En la Figura 9.1 se recuerda la arquitectura de la plataforma HW/SW propuesta, en la que TTN se comunica continuamente con el *gateway* mediante *WiFi*.

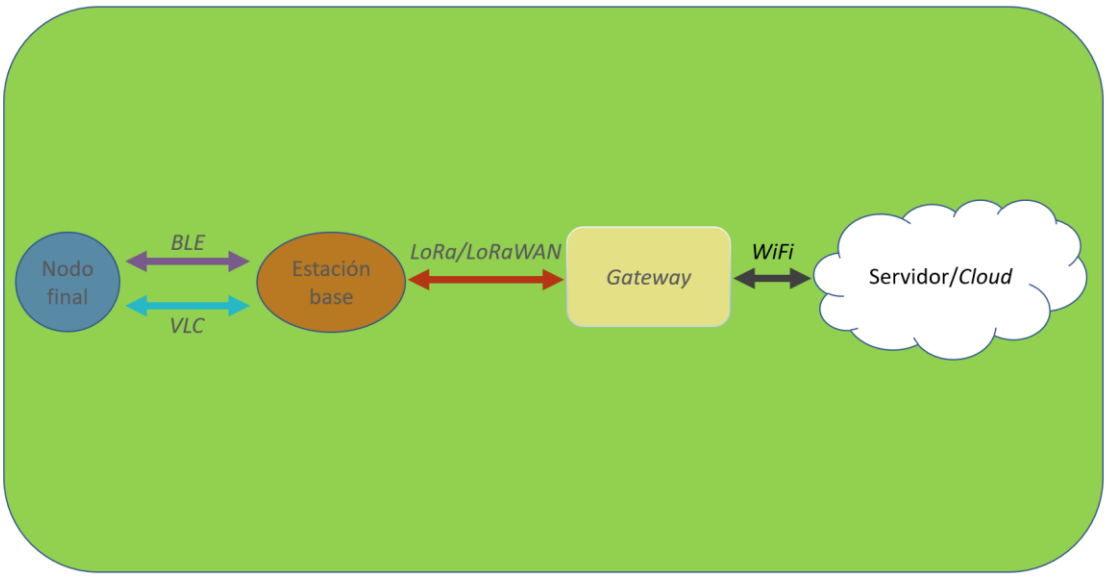


Figura 9.1: Arquitectura de la plataforma HW/SW propuesta: Elemento The Things Network.

En esta plataforma se cuenta con un foro dedicado exclusivamente a IoT, en el que existe código abierto y consultas de otros desarrollos ya completados. También existe la posibilidad de ubicar geográficamente un *gateway*, de consultar tutoriales e información

sobre integraciones. Y por último cuenta con una consola que integra toda la funcionalidad de la plataforma. Si se consulta su portal web [86] se visualiza una interfaz como la que se muestra en la Figura 9.2.

Para poder trabajar en esta plataforma se debe crear en primer lugar, una cuenta de usuario. Con la cuenta creada, se puede acceder a todas las funcionalidades de la plataforma, siendo las más utilizadas en el presente TFM, la consola, para el desarrollo de la solución, y el foro, para la resolución de diferentes dudas e inconvenientes que han ido surgiendo en el desarrollo de la solución.

Si se inicia la consola, aparece la opción de acceder a los *gateways* registrados o a las aplicaciones, tal y como se muestra en la Figura 9.3. El primer paso, consiste en registrar el *gateway*, por lo que se selecciona la opción GATEWAYS. Una vez dentro de esta sección, se da la opción de registrar un nuevo *gateway*.

Figura 9.2: Portal web The Things Network.

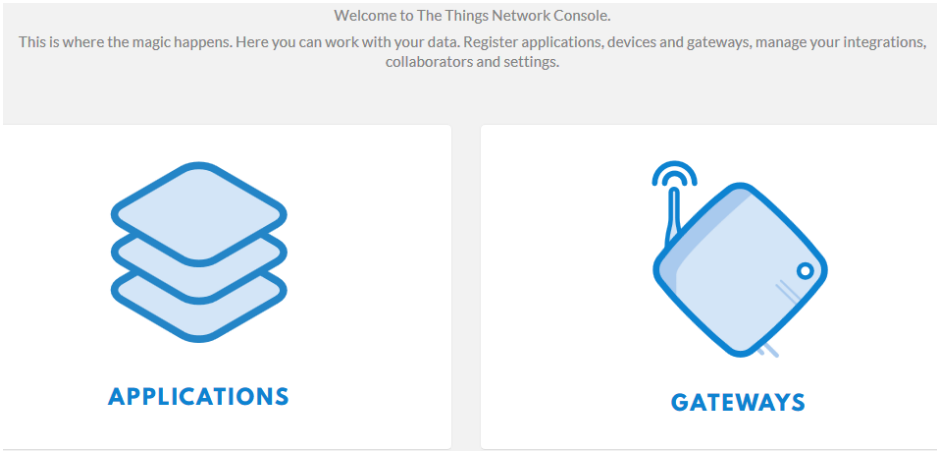


Figura 9.3: Consola de TTN.

En el registro del *gateway* se debe introducir, en primer lugar, el ID del *gateway*, configurado en el archivo *config.py* del *nanogateway*. También se debe seleccionar el plan de frecuencia adecuado, que para el caso de esta aplicación debe ser la frecuencia europea de 868 MHz. También es conveniente señalar si la antena estará ubicada dentro de un habitáculo (*indoor*) o en el exterior (*outdoor*). Además, se puede indicar la ubicación física del *gateway* a partir de un mapa. Con el proceso de registro completado se pueden consultar las características del *gateway* en el espacio "*Gateway Overview*", que presenta la disposición que se muestra en la Figura 9.4.

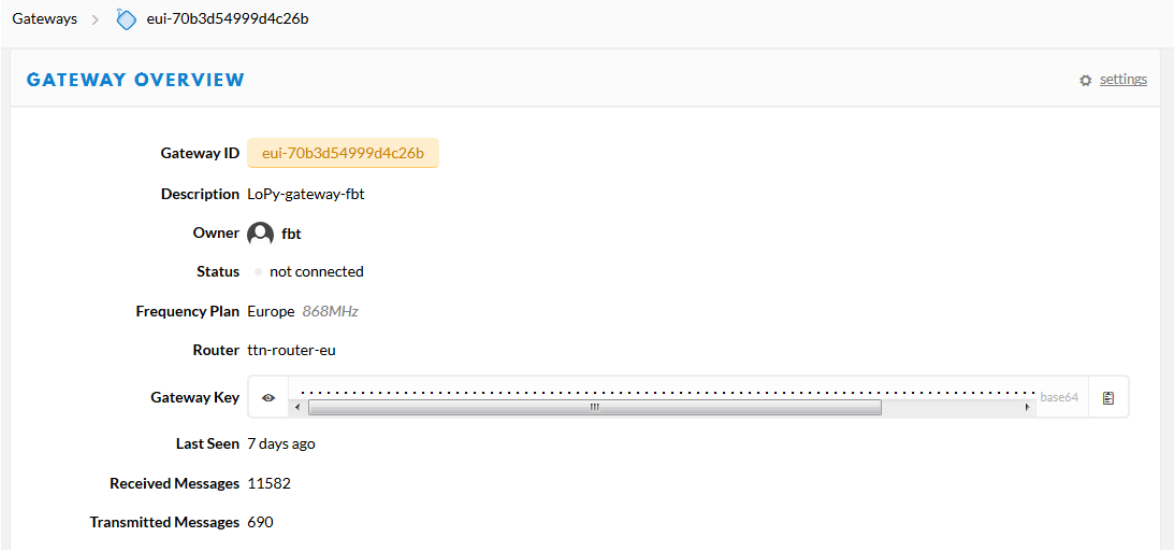


Figura 9.4: Gateway Overview.

Teniendo el *gateway* registrado, ya se puede proceder a la creación de una aplicación en la plataforma TTN.

Los pasos que se deben seguir son similares a los del caso del *gateway*. Así, en primer lugar, se debe registrar la aplicación. En este registro se requiere de un ID de la aplicación, la selección del gestor, que para el caso del presente TFM es el europeo, y el EUI asociado a la aplicación, que autogenerará la propia plataforma TTN. Introduciendo los datos y pulsando en registrar, finaliza el proceso de creación de la aplicación.

Con la aplicación creada se puede acceder al campo denominado "*Application Overview*", que se muestra en la Figura 9.5 y en la Figura 9.6. En ellas se describen las principales características de la aplicación, como su ID, su EUI (que será necesario utilizar en el nodo estación base), su clave de acceso, los colaboradores que pueden editar la aplicación, y los dispositivos asociados en la aplicación.

Si se accede a los dispositivos, se pueden visualizar los detalles y características de aquellos que se tengan registrados, además de disponer de la posibilidad de registrar nuevos dispositivos. El registro de un dispositivo es similar al del *gateway* y al de la aplicación.

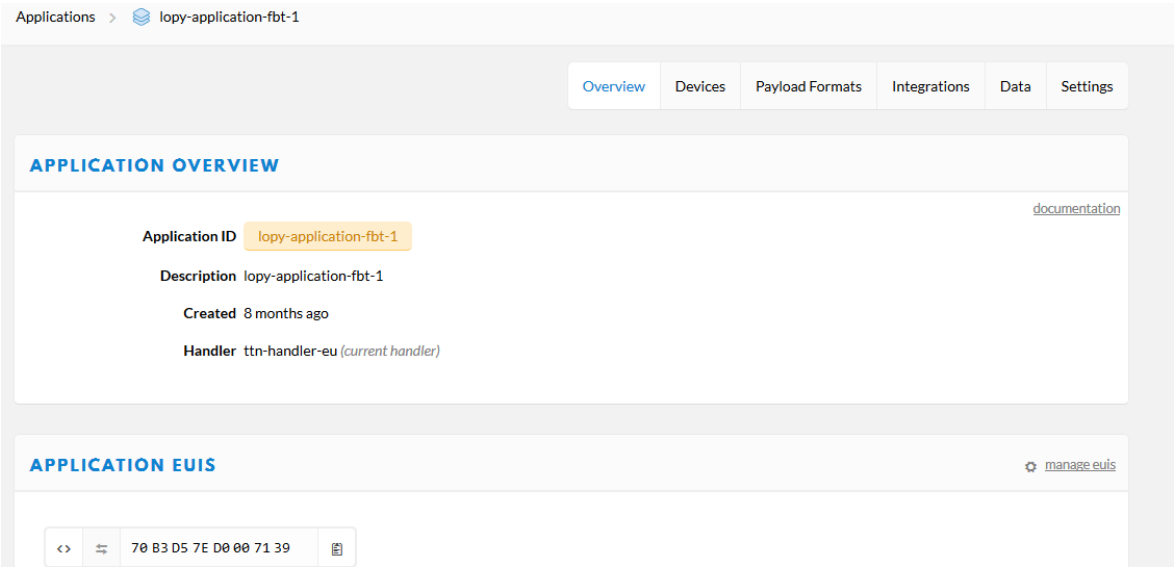


Figura 9.5: Application Overview.

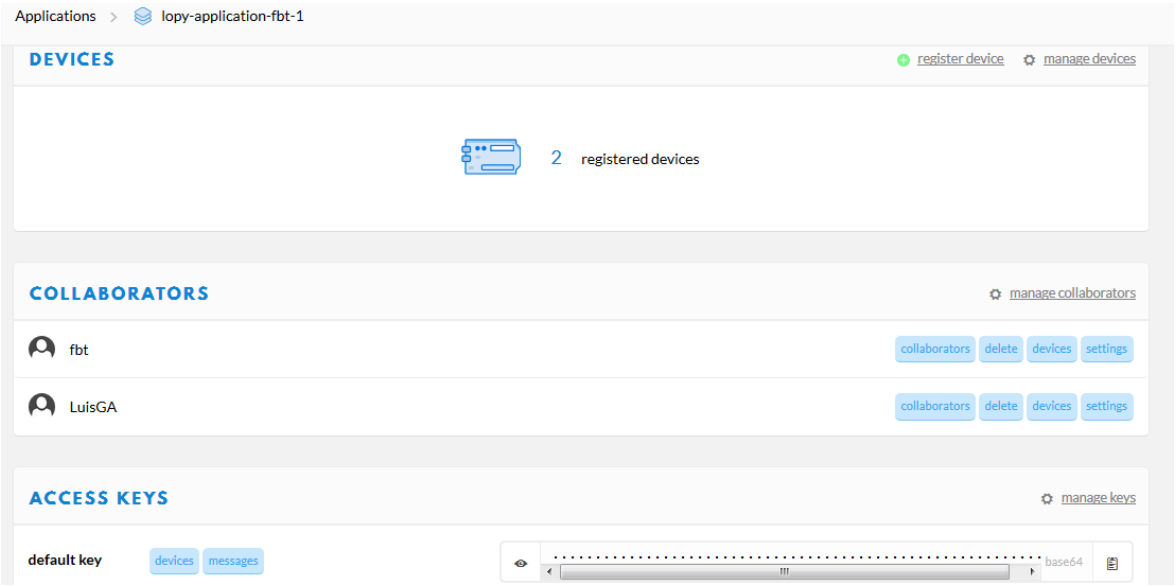


Figura 9.6: Application Overview_2.

Así, en el registro de un dispositivo se requiere la introducción de un ID, el EUI del dispositivo (que debe coincidir con el *dev_EUI* del nodo estación base) y el EUI de la aplicación. El ID del dispositivo se introduce manualmente por el usuario y el EUI de la aplicación se genera automáticamente en la creación de ésta. Por otra parte, el EUI del dispositivo se obtiene ejecutando la siguiente sentencia en el dispositivo que se desea registrar.

```
binascii.hexlify(network.LoRa().mac())
```

Con el registro del dispositivo se genera una clave específica para el dispositivo, denominada *app_key*, que debe incluirse en el código del nodo estación base. Si se consultan las características de uno de los dispositivos registrados, se encuentra una interfaz como la mostrada en la Figura 9.7 y en la Figura 9.8. En estas figuras se puede ver también el tipo de activación *LoRaWAN* que se utiliza para el dispositivo, que para el caso del presente TFM será OTAA. También se generan automáticamente los campos *Device Adress*, *Network Session Key* y *App Session Key*.

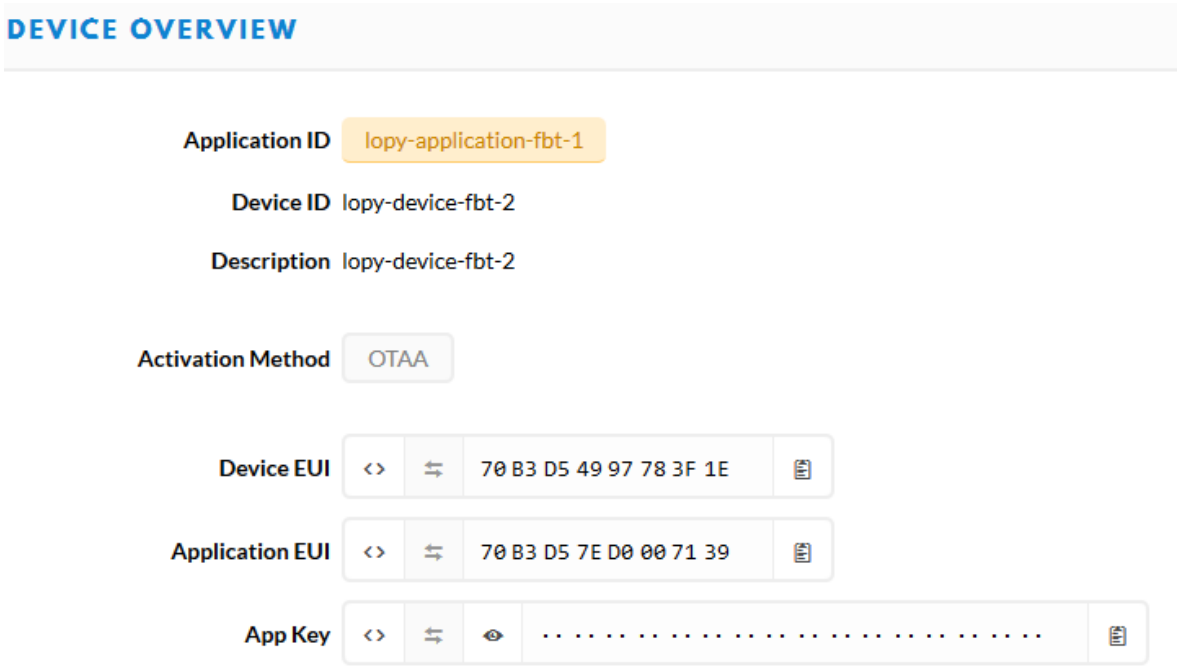


Figura 9.7: Device Overview.

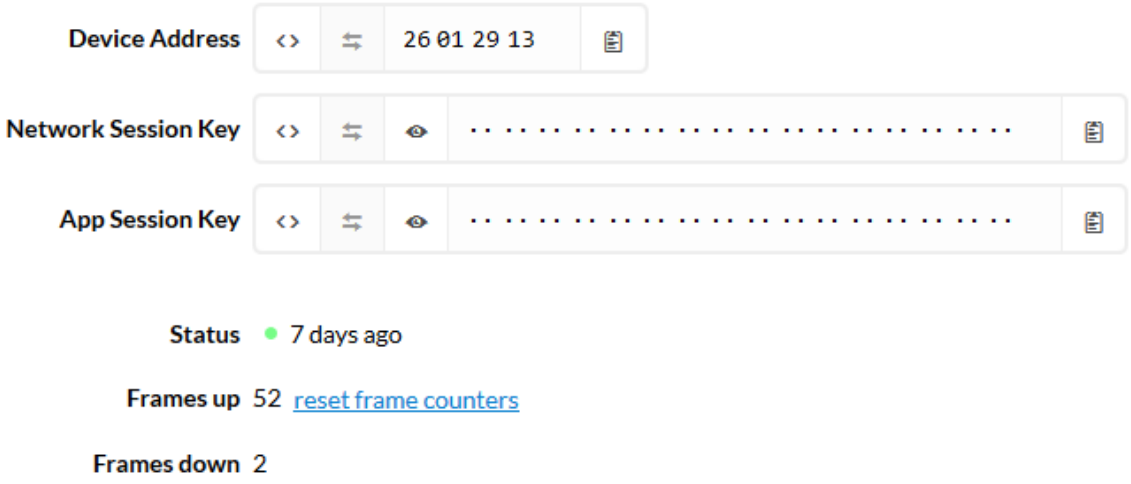


Figura 9.8: Device Overview_2.

Capítulo 10. Use of Case: Smart Parking

En los capítulos anteriores se han presentado los diferentes elementos que constituyen la arquitectura completa de la plataforma IoT de carácter genérico que se planteaba como objetivo principal del presente TFM. Sin embargo, como se ha podido comprobar en las respectivas explicaciones de cada bloque de la arquitectura, esta plataforma genérica queda abierta a modificaciones con el fin de ser fácilmente adaptable a soluciones específicas y concretas. En este capítulo se presenta un caso de uso o *use of case* sobre esta plataforma genérica.

El caso de uso planteado es un tipo de aplicación que se encuentra en el contexto de las *Smart Cities*, concretamente en el campo de la movilidad. La solución escogida como *use of case* para la plataforma genérica versa sobre *Smart Parking*.

La solución Smart Parking que se aborda en el presente TFM trata de detectar el estado de las plazas de aparcamiento en un *parking* privado y transmitir la información de dicho estado a través de la plataforma HW/SW propuesta. De esta forma, en el elemento de la arquitectura TTN, se espera la recepción de esa información de estado. Se ha acotado la funcionalidad de esta solución a la visualización del estado de la plaza de aparcamiento en TTN, sin embargo, como futura ampliación, se podría desarrollar adicionalmente, y mediante las técnicas de exportación de datos que ofrece TTN, una aplicación móvil o una

plataforma web en la que se tenga una interfaz cómoda y sencilla en la que el usuario pueda consultar el estado de las plazas del *parking*.

Existen diferentes escenarios en los que esta solución tiene cabida. Inicialmente se plantea para un parking privado, y como se trata de desarrollar un prototipo que valide la funcionalidad de la plataforma, se tendrá únicamente un nodo final, un nodo estación base y un nodo *gateway*. Se parte de la idea de que existe una infraestructura suficiente para dotar de alimentación ininterrumpida a los nodos estación base y al nodo *gateway*, así como de conexión *WiFi* en un punto concreto del *parking* para que el nodo *gateway* pueda conectarse con la plataforma TTN.

Por lo tanto, la distribución física de los elementos de la plataforma es:

- Nodo final: Ubicado encima de la plaza de aparcamiento con el sensor orientado hacia esta. Este nodo es alimentado mediante batería.

- Nodo estación base: Ubicado cerca de la plaza de estacionamiento con visión directa hacia el nodo final. En este caso, el nodo puede ser alimentado de forma ininterrumpida.

- Nodo *gateway*: Ubicado en una zona con acceso a una red *WiFi* y con alimentación ininterrumpida.

El uso de esta plataforma genérica en una aplicación de este tipo puede justificarse por diversas razones, todas ellas soportadas desde el punto de vista de las tecnologías de comunicación implementadas.

Por ejemplo, una razón puede ser la inexistencia de una infraestructura *WiFi* en un *parking* privado. Otra puede ser la poca cobertura de comunicaciones móviles debido a la atenuación que sufren las señales en las paredes y techo de un *parking*, y en ambos casos, el elevado consumo de potencia que suponen estos dos tipos de tecnologías de comunicación para el nodo final hace que estas, se conviertan en opciones difícilmente asumibles.

Además, en algunos casos, la utilización de una tecnología que facilite la descongestión del espectro radioeléctrico como *VLC* es bien considerada. A esto se debe sumar que se puede aprovechar la iluminación empleada en la comunicación *VLC* para alumbrar ciertas estancias del *parking*.

Para casos de uso como este, la plataforma genérica brinda diferentes opciones de comunicación. Un ejemplo puede ser, que para ciertos nodos, se puede desear una comunicación *VLC* y para otros, una comunicación *BLE*, ya sea por motivos de interferencia o porque energéticamente, según la funcionalidad requerida, una tecnología convierte a la plataforma en más eficiente que usando la otra. En definitiva, con unas adaptaciones mínimas de la funcionalidad requerida se logra una solución adecuada a las necesidades del *use of case* propuesto.

En la Figura 10.1 se muestra la adaptación resultante a nivel de arquitectura de la plataforma genérica. Se puede ver que la adaptación en este sentido es mínima, ya que sólo es necesario seleccionar el tipo de tecnología que se desea utilizar para la comunicación entre el nodo final y el nodo estación base. Se ha decidido que la comunicación ascendente entre estos dos nodos se realice a través de un enlace *VLC*, mientras que para la comunicación descendente, para esos dos mismos nodos se utilizará *BLE*. El resto de la arquitectura se mantiene igual que la plataforma genérica explicada en capítulos anteriores.

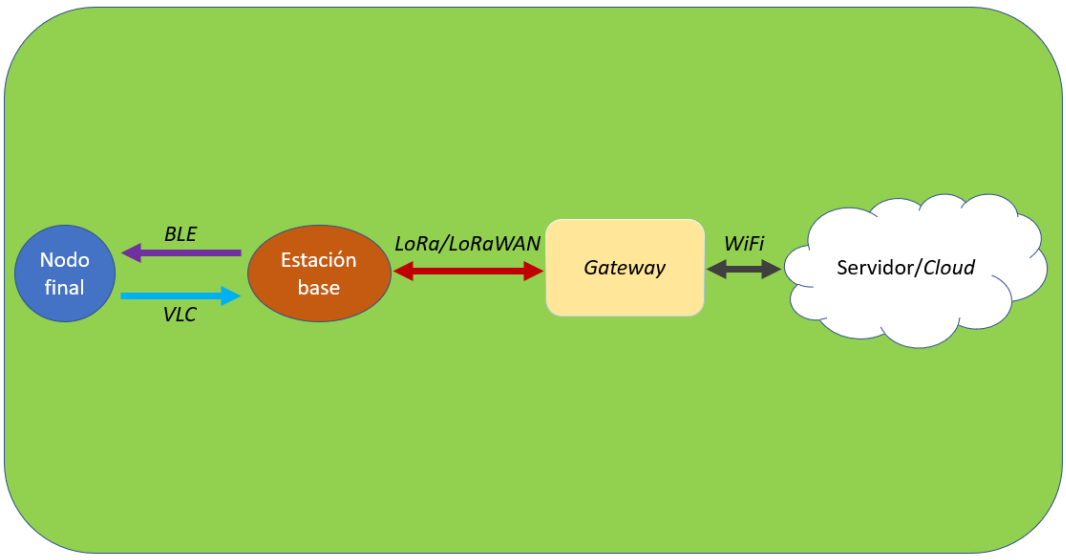


Figura 10.1: Adaptación de la plataforma HW/SW para la solución Smart Parking.

A continuación, se explicarán aquellos aspectos que se deben tener en cuenta para la puesta en marcha de la solución específica sobre la plataforma genérica desarrollada en cada uno de los elementos de la arquitectura, empezando por el nodo final y terminando por la plataforma TTN.

10.1 Adaptación del Nodo Final.

El nodo final para la aplicación específica que se presenta en este capítulo está formado por los siguientes elementos:

- Dispositivo LoPy: Dispositivo que gestiona y hace uso de los demás elementos del nodo final.

- Sensor: Se encarga de detectar el estado de la plaza de estacionamiento. Se comunica de forma cableada con la *Expansion Board* del dispositivo LoPy.

- Batería: Elemento que provee de energía eléctrica al nodo final. Se conecta directamente a la *Expansion Board* del dispositivo LoPy.

- Módulo emisor óptico: Elemento de transmisión *VLC* para comunicarse con el nodo estación base. Se conecta también a la *Expansion Board* del dispositivo LoPy.

En la Figura 10.2 se muestra el nodo final con todos sus elementos ya conectados. Tanto el módulo emisor óptico como el dispositivo LoPy y su *Expansion Board* se han explicado con detalle en el capítulo 3, al ser elementos fijos de la plataforma genérica propuesta en este TFM. No obstante, el sensor y la batería son elementos que se deben seleccionar en función de la aplicación específica que se vaya a implementar, por lo que corresponde presentar ambos elementos en este capítulo.

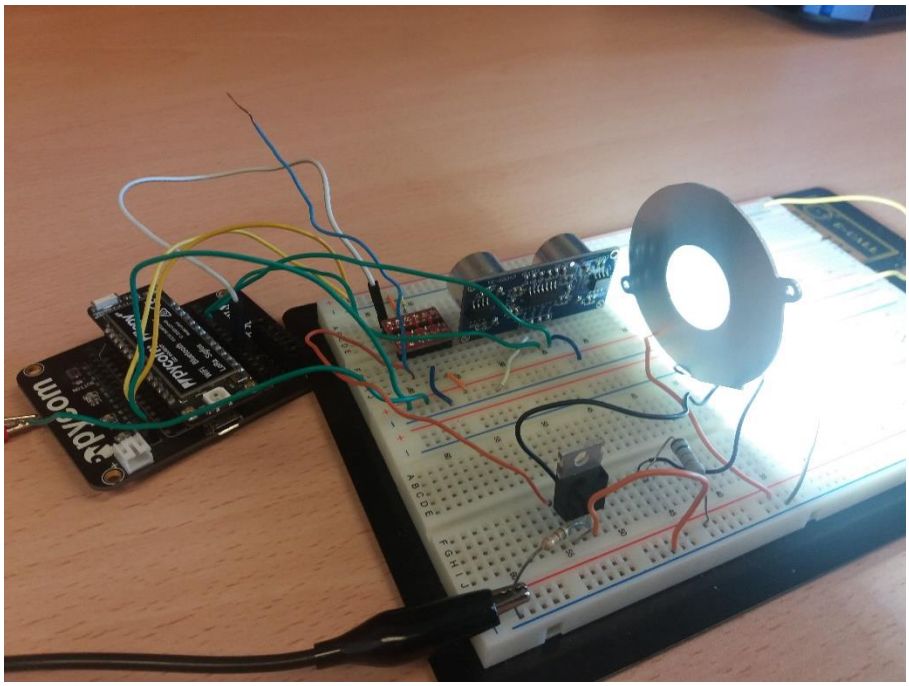


Figura 10.2: Montaje completo del Nodo Final.

10.1.1 Integración del sensor

El sensor escogido en este caso es el sensor HC-SR04 cuyo aspecto se muestra en la Figura 10.3.

Figura 10.3: Sensor HC-SR04.

Se trata de un sensor de ultrasonidos que permite medir distancias. Al tratarse de tecnología ultrasonora, la señal transmitida es inaudible por el oído humano, pero permite enviar un pulso y detectar la onda reflejada a partir de un micrófono diseñado para la

frecuencia de funcionamiento del sensor (40 KHz). Su funcionamiento se basa en medir el tiempo existente entre el instante en el que se envía el pulso, y el instante en el que se recibe la señal reflejada. De esta forma, y sabiendo que la velocidad de propagación en el sonido para condiciones normales (20ºC, 50% humedad, presión atmosférica a nivel del mar) es de 343 m/s, se puede obtener la distancia que hay desde el sensor hasta el objeto que ha producido la reflexión del pulso.

Este sensor es de baja calidad, y su uso está limitado para pruebas o aplicaciones muy concretas en las que exista un entorno de propagación multitrayecto prácticamente nula. Sin embargo, debido a su bajo coste y a la simple funcionalidad que va a desarrollar en el prototipo de esta solución *Smart Parking*, su uso está justificado. En la Tabla 10.1 se adjuntan sus características y prestaciones.

Características HC-SR04	
Alimentación	5 V
Frecuencia de trabajo	40 KHz
Consumo (en suspensión)	<2 mA
Consumo (en activo)	15 mA
Ángulo efectivo	<15º
Rango de medida	2 cm - 400 cm
Resolución	0.3 cm

Tabla 10.1: Características del sensor HC-SR04.

El conexionado del sensor con la Expansion Board del dispositivo LoPy se realiza a través de pines digitales más los pines de alimentación para dotar de energía al sensor. En concreto, los pines digitales del dispositivo LoPy utilizados son 'P9' y 'P10', que se corresponden con los pines 'G16' y 'G17' de la *Expansion Board* respectivamente. La alimentación se realiza desde los pines de '5V' y 'GND' del dispositivo LoPy.

Con respecto a su implementación a nivel de *firmware*, se ha desarrollado una librería denominada 'Ultrasonic.py' que permite obtener la distancia entre el sensor y el objeto

reflector. Esta librería se ha desarrollado en lenguaje *MicroPython* con el fin de poder integrarla en la funcionalidad del nodo final sin ningún tipo de conflicto.

En la Figura 10.4 se declara la clase principal de la librería, denominada 'Ultrasonic'. En esta clase se inicializa el objeto con los pines digitales que se pasan por parámetros y se configuran como pin de salida (para el pin 'Trigger' del sensor HC-SR04) y como pin de entrada (para el pin 'Echo' del sensor HC-SR04). El pin 'Trigger' se encarga de enviar el pulso cambiando su estado digital y el pin 'Echo' señaliza el estado de recepción de la señal reflejada.

```
5  import machine
6  import time
7
8  class Ultrasonic:
9      def __init__(self, tPin, ePin):
10         self.triggerPin = tPin  #Se asocia el pin Trigger
11         self.echoPin = ePin #Se asocia el pin Echo
12
13         #Se inicializa el pin Trigger como salida
14         self.trigger = machine.Pin(self.triggerPin)
15         self.trigger.init(machine.Pin.OUT, pull=None, value=0)
16
17         #Se inicializa el pin Echo como entrada
18         self.echo = machine.Pin(self.echoPin)
19         self.echo.init(machine.Pin.IN, pull=None)
20
```

Figura 10.4: Declaración de la clase 'Ultrasonic'.

En la Figura 10.5 se muestra la función que permite medir la distancia que existe entre el sensor y el objeto reflector en unidades de centímetro (cm). El procedimiento que sigue esta función es inicialmente poner a nivel lógico alto el pin 'Trigger' durante 10 µs y después devolverlo a nivel lógico bajo con el fin de crear un pulso de 10 µs de duración. Tras enviar el pulso, se almacena en la variable 'start' el instante exacto en el que este fue enviado, y al detectar un nivel lógico alto del pin 'Echo' se almacena en la variable 'end' el instante de detección. Seguidamente se realiza la diferencia entre ambos tiempos (en unidad de µs), y aplicando la Ecuación 10.1 se obtiene la distancia que hay entre el sensor y el objeto reflejado. La ecuación se obtiene a partir de la relación física entre velocidad, distancia y tiempo, en la que la velocidad es el cociente entre distancia y tiempo. Si se despeja la distancia de la ecuación y se tiene en cuenta que la distancia recorrida por la señal emitida

es el doble de la que hay entre el sensor y el objeto reflector, y se convierte la velocidad de propagación del sonido de 343 m/s a 29 cm/µs se obtiene la expresión de la Ecuación 10.1.

$$Distancia\ (cm) = \frac{\frac{end-start}{2}}{29} \qquad (10.1)$$

```
#Se declara la función que mide la distancia que hay desde el sensor hasta el objeto reflector
def distance_in_cm(self):
    start = 0
    end = 0

    # Se envía un pulso de 10 us
    self.trigger.value(1)
    time.sleep_us(10)
    self.trigger.value(0)

    # Se espera hasta que se envía el pulso
    while self.echo.value() == 0:
        start = time.ticks_us() #Comienza la cuenta

    # Se espera hasta que se reciba el pulso
    while self.echo.value() == 1:
        end = time.ticks_us() #Finaliza la cuenta

    #Se restan ambos tiempos
    differ = time.ticks_diff(start, end)

    #Se calcula la distancia sabiendo que la velocidad de propagación del
    #sonido es de 343 m/s o de 29 us/cm, y que la señal realiza dos veces
    #el mismo camino
    dist_in_cm = (differ / 2) / 29

    return dist_in_cm
```

Figura 10.5: Función distance_in_cm().

Así, con esta librería se ofrece la posibilidad de calcular la distancia que hay entre el sensor y el objeto reflector, por lo que para la aplicación que se desea implementar basta con definir un valor umbral a partir del cual se pueda interpretar si el estado de la plaza de estacionamiento es de libre u ocupado. La definición de este valor umbral depende del entorno donde se vaya a instalar el nodo final, y realizar un estudio previo de los valores medidos por el sensor para fijar ese valor umbral es totalmente aconsejable.

En definitiva, en la funcionalidad del nodo final se pedirá el valor de la distancia entre el sensor y el objeto reflector haciendo uso de la librería 'Ultrasonic.py' y en función de si el valor recibido es menor o mayor que el umbral, se determinará si la plaza está libre u ocupada.

10.1.2 Integración de la batería

La batería debe alimentar al nodo final, y en aplicaciones de este tipo siempre es recomendable prolongar su vida útil para evitar su sustitución en cortos períodos de tiempo. Toda la funcionalidad y el uso de las tecnologías de comunicación seleccionadas está enfocado a reducir el consumo de potencia con el fin de evitar continuas sustituciones o recargas de baterías.

La *Expansion Board* asociada al dispositivo LoPy cuenta con un conector JST de batería LiPo. Haciendo uso de este conector se puede alimentar el dispositivo con una batería de Polímero de Litio (LiPo). La batería seleccionada para la aplicación planteada se referencia con el modelo BAT573 de Electronic NIMO [87]. La tensión de las baterías LiPo depende del número de celdas de la batería. Una celda aporta una tensión de 3.7V, de forma que una batería de 4 celdas proporcionaría una tensión de 14.8V. La batería del modelo seleccionado cuenta con una única celda, por lo que aporta una tensión nominal de 3.7V y su capacidad es de 180 mAh. La batería es recargable por lo que su ciclo de vida es alto. En la Figura 10.6 se muestra la batería BAT573.

Figura 10.6: Batería LiPo BAT573.

10.1.3 Diseño y desarrollo de la funcionalidad del nodo final

Antes de desarrollar el diagrama de flujo del *firmware* que se implementará en el nodo final, es necesario presentar sus tareas y restricciones con el fin de obtener un funcionamiento adecuado a las especificaciones de la solución planteada.

El nodo final, como se ha explicado con anterioridad, debe decidir el estado de la plaza de aparcamiento en función de la distancia medida por el sensor ultrasónico entre él y el objeto reflector. Una vez que se conoce el estado de la plaza, se debe comunicar esa información al nodo estación base mediante *VLC*. También se debe tener en cuenta que haciendo uso de *BLE*, el nodo final puede recibir mensajes desde la estación base. En este caso, si recibiera un mensaje, el nodo final debe interpretarlo y valorar si le corresponde realizar una actualización de su configuración. A lo largo del desarrollo del *firmware* del nodo final se deben considerar los siguientes aspectos.

- Consumo de potencia: Se debe minimizar el consumo de potencia en todas las acciones del nodo final. Esto se debe a que el nodo se alimenta mediante el uso de baterías, y por el enfoque de la aplicación se requiere que la duración de la batería sea lo más prolongada posible.

- Inactividad: En los intervalos de tiempo en los que no se envían datos desde el nodo final, se puede entender que se ha perdido la conexión, o que se ha producido algún error en el funcionamiento del nodo. Por esto, resulta interesante enviar con un período constante, pero modificable, un tipo de trama que indique que el nodo final sigue operando correctamente.

- Configuración remota del nodo: Se debe considerar la posibilidad de que se tengan que reajustar los parámetros de configuración del nodo final, dependiendo de las características particulares de cada aplicación de *Smart Parking*. El envío de esta configuración se realiza mediante la recepción *BLE* de los datos enviados desde la estación base.

- Período de poco uso: En una aplicación como la de tipo *Smart Parking* propuesta en este caso de uso, existen períodos en los que la exigencia de detectar el estado de la plaza se relaja debido a la leve afluencia de vehículos en horas determinadas. Por ejemplo, el tráfico de vehículos en un período nocturno puede que sea menor que en horas punta durante el día. Aprovechando esta circunstancia, se puede crear un modo "nocturno" en el que se evalúe el estado de la plaza de aparcamiento con menos frecuencia, favoreciendo así el ahorro en el consumo de potencia del nodo final. Las franjas horarias que delimitan el modo nocturno del modo normal pueden ser definidas en la configuración remota del nodo final.

Considerando la funcionalidad principal del nodo, así como estos últimos requisitos, se diseña el diagrama de flujo del *firmware* del nodo final que se muestra en la Figura 10.7.

En el diagrama de flujo de la Figura 10.7 se puede observar cómo la primera acción que se realiza desde el nodo final, y que de hecho sólo se realiza una vez, es el envío de una trama de inicialización denominada '*Start Frame*'. En esta trama, cuyo contenido exacto se detallará más adelante, se envía información acerca del nodo final. Tras el envío de esta trama se procede a definir el modo de funcionamiento. Existen dos modos de funcionamiento, el modo normal y el modo nocturno. La diferencia entre estos dos modos radica en el tiempo que permanece dormido el nodo final. En el modo normal el tiempo de *sleep* es menor que en el modo nocturno, por lo que, en este último, el ahorro de consumo de potencia es mayor.

Tras decidir el modo de funcionamiento que se sigue en la iteración, se realiza una consulta al sensor de ultrasonidos. Este sensor devuelve la distancia que existe entre este y el objeto reflectante. Con este valor de distancia se realiza una comparación en el siguiente proceso del diagrama de flujo, evaluándose el estado de la plaza. En este proceso se compara el valor de distancia recibido del sensor con el valor umbral definido previamente en el *firmware*. Si la distancia es mayor significa que no hay un vehículo posicionado en la plaza y por lo tanto se determina que la plaza se encuentra en estado libre. Por el contrario, si la distancia es menor que el valor umbral, se determina que la plaza está ocupada por un vehículo.

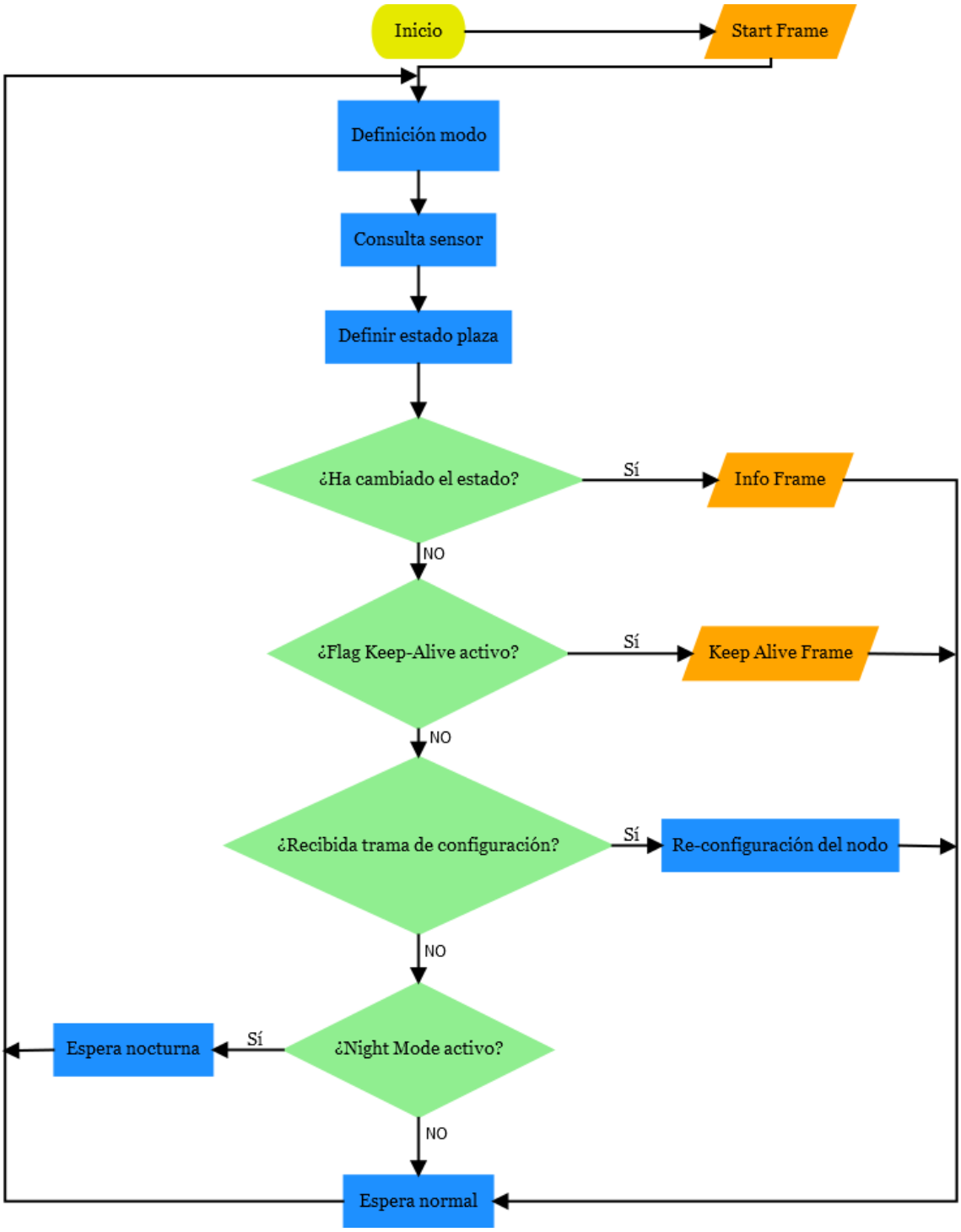


Figura 10.7: Diagrama de flujo del funcionamiento del nodo final.

Una vez que se define el estado de la plaza, se evalúa si ha cambiado el estado de la plaza con respecto a la iteración anterior. En caso afirmativo, se envía una trama de información denominada '*Info Frame*' en la que se actualiza al nodo estación base el estado de la plaza de estacionamiento. Tras el envío de la trama de información, se realiza un periodo de espera en bajo consumo, y posteriormente se inicia de nuevo el mismo flujo partiendo

desde el proceso de definición del modo de funcionamiento. En caso negativo, se prosigue avanzando en el diagrama de flujo hacia el siguiente evento, en el que se trata de evaluar si se debe enviar una trama de '*Keep Alive*' que indique que el nodo sigue funcionando. El período con el que se envía esta trama es configurable y su misión es informar de que el nodo sigue funcionando correctamente. Su utilidad se muestra al no cambiar el estado de la plaza de estacionamiento en un largo período de tiempo, y no haber enviado en ese período ninguna trama hacia el nodo estación base. Como en el caso anterior, tras el envío de la trama, se realiza una espera de bajo consumo de potencia antes de iniciar nuevamente una nueva iteración de funcionamiento. En caso de que no se haya cumplido aún el período de trama '*Keep Alive*', se sigue avanzando en el diagrama de flujo hacia la siguiente cuestión.

Como se explicó anteriormente, existe la posibilidad de recibir una trama de configuración procedente desde el nodo estación base. En este caso, se valora si se ha recibido una trama de configuración y en caso de ser así se interpreta dicha trama y se reconfigura el nodo final. Tras la configuración se realiza una espera de bajo consumo de potencia antes de iniciar una nueva iteración de funcionamiento. En caso de que no se haya recibido ningún mensaje desde el nodo estación base, se sigue avanzando en el diagrama de flujo hacia el siguiente paso, en el que simplemente se evalúa si se debe realizar una espera de bajo consumo más o menos larga, dependiendo del modo de funcionamiento. En caso de estar en modo nocturno, esta espera es más prolongada que la del modo normal. Este tiempo, en ambos modos, es configurable.

10.1.4 Formato de los paquetes

A continuación, se describe detalladamente el contenido de cada tipo de trama, tanto las que envía el nodo final como la que envía el nodo estación base.

En el formato de los diferentes tipos de trama se ha establecido que los 2 primeros bytes de cada una de ellas sean comunes a todos los tipos de tramas. Así, el Byte 0 en todas las tramas contiene la información mostrada en la Tabla 10.2.

Bit	Nombre	Descripción
7	Estado de la plaza de aparcamiento (solo para *Info Frame*)	'0' indica que la plaza está libre.
		'1' indica que la plaza está ocupada.
6-4	Reservados	Bits no utilizados en esta versión del nodo final.
3	Tipo de trama	0d(0000b) – *Info frame* 1d(0001b) – *Keep-Alive frame* 5d(0101b) – *Start frame* Valores del 6 al 15 están reservados.
2		
1		
0		

Tabla 10.2: Contenido del Byte 0 de las tramas.

Por otra parte, el Byte 1 representa un contador de tramas que va de 0 a 255, cuyo propósito es servir de mecanismo para detectar que no se hayan perdido tramas en la comunicación.

En concreto, estos dos *bytes* indican el tipo de trama que se envía (Byte 0), y el orden en el que son enviadas (Byte 1).

Una vez conocidos los dos *bytes* que siempre estarán en todas las tramas enviadas, se puede describir por separado cada uno de los tipos de tramas que se pueden enviar desde el nodo final hacia el nodo estación base.

Como se comentó anteriormente, al iniciar el funcionamiento de un nodo final se envía una trama de inicio o *Start Frame*. Esta trama contiene información sobre el estado y la configuración inicial del nodo. El contenido de esta trama se detalla a continuación.

En la trama de inicio, o *Start Frame*, se incluyen los campos mostrados en la Tabla 10.3.

Byte	Nombre	Descripción
0	Datos de la trama	Información de la trama.
1	Contador de tramas	Número de trama.
2	Versión del *firmware*	Número de versión del *firmware*.
3	NM Start	Hora de inicio del modo nocturno.

4	NM Period	Duración del modo nocturno en horas.
5	ID	Identificador del nodo.
6	NM Keep Alive	Periodo de tiempo entre dos tramas *Keep-Alive* durante el modo nocturno.
7	NM Sleep Time	Tiempo de espera (dormido) del nodo durante el modo nocturno.
8	Sleep Time	Tiempo de espera (dormido) del nodo durante el modo normal.
9	Keep Alive	Periodo de tiempo entre dos tramas *Keep-Alive* durante el modo normal.
10	Reservado	Bit no utilizado en esta versión del nodo final.

Tabla 10.3: Contenido de las tramas de tipo Start Frame.

Por otro lado, cuando se detecta un cambio en el estado de la plaza de estacionamiento, se envía una trama de información o *Info Frame*, cuyo contenido se muestra en la Tabla 10.4.

Byte	Nombre	Descripción
0	Datos de la trama	Información de la trama.
1	Contador de tramas	Número de trama.
2-4	Reservados	Bits no utilizados en esta versión del nodo final.
5	ID	Identificador del nodo.
6-10	Reservados	Bits no utilizados en esta versión del nodo final.

Tabla 10.4: Contenido de las tramas de tipo Info Frame.

Para indicar que el nodo final sigue funcionando correctamente, se envían periódicamente tramas de tipo *Keep-Alive*, que además del significado implícito a su recepción, recogen el tiempo transcurrido desde que se inició el funcionamiento del nodo final. El contenido de las tramas de tipo *Keep-Alive* es el mostrado en la Tabla 10.5.

Byte	Nombre	Descripción
0	Datos de la trama	Información de la trama.

1	Contador de tramas	Número de trama.
2	Tiempo transcurrido (hh)	Horas transcurridas desde que se inició el funcionamiento del nodo.
3	Tiempo transcurrido (mm)	Minutos transcurridos desde que se inició el funcionamiento del nodo. Se acumulan, es decir, si lleva 2h y 10min funcionando aparecerá en este campo el valor: 130.
4	Reservado	Bit no utilizado en esta versión del nodo final.
5	ID	Identificador del nodo.
6-10	Reservados	Bits no utilizados en esta versión del nodo final.

Tabla 10.5: Contenido de las tramas de tipo Keep-Alive Frame.

Otro tipo de trama definida en la aplicación desarrollada en este TFM es la trama de configuración. A diferencia de las tramas anteriores, esta trama se envía desde el nodo estación base, y el nodo receptor es el nodo final. En esta trama se incluyen valores de configuración que debe adoptar el nodo final.

Para indicar qué nodo se desea configurar, se debe introducir en el campo *NodoToConfig* el ID del nodo que se desea configurar. En el nodo final, se filtran las tramas, y solo se atienden las que lleven en el campo *NodoToConfig* el identificador asociado al nodo final.

En la Tabla 10.6 se muestran los campos definidos en una trama de tipo *Configuration*.

Byte	**Nombre**	**Descripción**
0	Datos de la trama	Información principal de la trama de configuración.
1	ID nodo	Identificador que se va a asignar al nodo (En caso de que el bit 6 del Byte 0 esté a '1').
2	Versión del *firmware*	Número de versión del *firmware* actualizado.
3	NM Start	Hora de inicio del modo nocturno.

4	NM Period	Duración del modo nocturno en horas.
5	NM Sleep Time	Tiempo de espera (dormido) del nodo durante el modo nocturno.
6	NM Keep Alive	Periodo de tiempo entre dos tramas *Keep-Alive* durante el modo nocturno.
7	NodoToConfig	0xXX- ID del nodo que se desea configurar.
8	Sleep Time	Tiempo de espera (dormido) del nodo durante el modo normal.
9	Keep Alive	Periodo de tiempo entre dos tramas *Keep-Alive* durante el modo normal.
10	Reservado	Byte no utilizado en esta versión del nodo final.

Tabla 10.6: Contenido de las tramas de tipo Configuration Frame.

En el caso de las tramas de tipo *Configuration Frame*, el Byte 0 es diferente al del resto de las tramas, ya que se requiere del uso de varios bits para identificar el tipo de trama, es decir, que el receptor de la trama pueda diferenciar una trama de configuración de otros paquetes recibidos. Además, al tratarse de una trama que se envía en el sentido inverso al de la comunicación utilizada en el envío de las otras tramas, no se adjunta en el paquete de configuración información del estado del nodo, como si ocurría en el Byte 0 del resto de tramas. El Byte 0 de la trama de configuración contiene la información mostrada en la Tabla 10.7.

Bit	Nombre	Descripción
7	Reservado	Bit no utilizado en esta versión del nodo final.
6	Asignar ID	'1' indica que se desea asignar un ID al nodo, '0' que no. El ID a asignar se ubica en el Byte 1.
5	Código de configuración	Se fija el código a '110011'. Este código se comprueba en el nodo final, de manera que, si se recibe este código, se
4		

3		interpreta esta trama como de configuración, y en caso contrario, se descarta.
2		
1		
0		

Tabla 10.7: Contenido del Byte 0 de las tramas de tipo Configuration Frame.

10.1.5 Adaptación del *firmware* del nodo final

En el capítulo 8 ya se describió el *firmware* del nodo final en aquellos aspectos que son genéricos en todas las soluciones. Por lo que a continuación se van a describir únicamente aquellos aspectos que han sido adaptados o incluidos específicamente para la solución *Smart Parking* propuesta como caso de uso de la plataforma desarrollada en el presente TFM .

La primera y más importante adaptación que se realiza es seleccionar las tecnologías de comunicación que se van a utilizar para transmitir datos al nodo estación base, así como para recibirlos desde este. En este caso, como se expuso al comienzo de este capítulo, la comunicación *uplink* (hacia el nodo estación base) se realiza mediante *VLC*, mientras que la comunicación *downlink* (desde el nodo estación base) se realiza mediante *BLE*.

En la Figura 10.8 se muestra la selección de estas tecnologías para cada sentido de la comunicación. Además, se declaran variables importantes para el funcionamiento del nodo final, como pueden ser el identificador del nodo, el contador de tramas, el modo de funcionamiento o variables temporales. Por otro lado, en la Figura 10.9 se muestran las variables relacionadas con la configuración inicial del nodo final.

También se crean diferentes objetos temporales como *timers* o *Real Time Clocks* (RTC) que permiten llevar la gestión temporal de las tramas. En la Figura 10.10 se muestran los objetos de este tipo que han sido creados. En esta misma Figura 10.10 se inicializa la cuenta de los dos *timers* creados para gestionar el período de envío de tramas '*Keep-Alive*', así como el tiempo transcurrido desde que se inició el nodo final. También se crea un objeto de la librería '*Ultrasonic.py*' que permite realizar lecturas del sensor.

```
#Selección tecnología de trasmisión
techselectedTX = 0x01 #VLC          0x01:VLC; 0x02: BLE; 0x03: VLC+BLE simultáneo
#Selección tecnología de recepción
techselectedRX = 0x02 #BLE          0x01:VLC; 0x02: BLE; 0x03: VLC+BLE simultáneo

#Estado de la plaza
slot = 0

#ID del nodo
id1=0x05

#Contador para el frameCounter
contador = 0

#Variables auxiliares para el cálculo de current hours and minutes
total = 0
totalh = 0
totalm = 0

#Modo nocturno(NM=1) o normal(NM=0)
NM = 0

#Variables iniciales del RTC
year = 2020
month = 1
day = 15
hour = 10
minutes = 00
seconds = 0
usec = 0
```

Figura 10.8: Selección de las tecnologías de comunicación y variables declaradas.

```
#configuración inicial del nodo final
#versión del firmware
version = 01
#hora a la que arranca el modo nocturno. Tipo:24horas
NMStart = 24
#Duración del modo nocturno. (En horas)
NMPeriod = 6
#Intervalo de tiempo entre dos tramas Keep-Alive en el modo nocturno. 4h --> 14400s
NMKeepAlive = 14400
#Intervalo de tiempo en el que el nodo está "dormido" en el modo nocturno. 5min --> 300s
NMSleepTime = 300
#Intervalo de tiempo entre dos tramas Keep-Alive en el modo normal. 2h --> 7200s
KeepAlive = 30 #Provisional para pruebas
#Intervalo de tiempo en el que el nodo está "dormido" en el modo normal. 2min --> 120s
SleepTime = 10 #Provisiona para pruebas
```

Figura 10.9: Variables configurables.

```
93  ########################################Inicializaciones##############
94
95  #Creamos objeto rtc y lo inicializamos con un valor determinado
96  rtc = RTC()
97  rtc.init((year, month, day, hour, minutes, seconds, usec, 0))
98  rtc.ntp_sync("pool.ntp.org") #Sincronización
99
100 #Creamos dos objetos Timer
101 chrono = Timer.Chrono()
102 k = Timer.Chrono()
103
104 #Se inicia la cuenta para ambos
105 chrono.start()
106 k.start()
107
108 #Inicialización sensor
109 sensor= ultrasonic.Ultrasonic('P9','P10')
```

Figura 10.10: Timers, RTC y objeto sensor utilizados.

Una vez que se ha creado un objeto que pueda utilizar funciones de la librería '*Ultrasonic.py*', se define la función *consulta_sensor()* en la que se solicita al sensor la distancia entre este y el objeto reflector, evaluándose a continuación si la plaza está libre u ocupada. En este caso, el umbral establecido es de 35 cm. En la Figura 10.11 se muestra la definición de esta función.

```
124 #Función que valora el estado de la plaza
125 def consulta_sensor():
126     global slot
127     dist = sensor.distance_in_cm()
128     print("Dist = {}".format(dist))
129     if(dist>35): #estudiar entorno primero para fijar el umbral
130         slot = 0
131         print('plaza libre')
132     else:
133         slot = 1
134         print('plaza ocupada')
```

Figura 10.11: Función consulta_sensor().

Como se explicó anteriormente, existe un byte que se encarga de contar las tramas que se han enviado, con el fin de detectar en el destino, si han llegado todos las tramas correctamente, o se ha extraviado alguna. En la Figura 10.12 se define la función *increase_frameCounter()* que incrementa la cuenta cada vez que se va a enviar una trama.

En la Figura 10.12 se define también la función *actualizar_config()* que recoge los valores recibidos en la trama de configuración que se envía desde la estación base, y actualiza la configuración del nodo final con dichos valores.

```
260  def increase_frameCounter():
261      global contador
262      contador = (contador + 1) & 0xFF
263
264  #Función para actualizar la configuración actual del nodo por los
265  #valores recibidos de la trama de configuración
266  def actualizar_config():
267      print('actualizo')
268      global version
269      global NMStart
270      global NMPeriod
271      global NMKeepAlive
272      global NMSleepTime
273      global SleepTime
274      global KeepAlive
275      global id1
276      global threhold
277      version = data[2]
278      NMStart = data[3]
279      NMPeriod = data[4]
280      NMKeepAlive = data[5]
281      NMSleepTime = data[6]
282      SleepTime = data[8]
283      KeepAlive = data[9]
284      if(data[0] == 0x73):
285        id1= data[1]
```

Figura 10.12: Funciones increase_frameCounter() y actualizar_config().

En la Figura 10.13 se muestra la definición de la función *define_mode()*, que según la hora actual medida por el objeto RTC, así como de la hora de inicio, fin, y del período establecido como período nocturno, se define si el nodo final debe funcionar en modo normal o en modo nocturno. En la Figura 10.13 se muestran también la función *clear_frame()* y *clear_bytearr(),* que como se explicó en el capítulo 8, se encargan de borrar el contenido de las variables utilizadas para la comunicación ascendente o descendente, respectivamente. Por último, en la Figura 10.13 se muestra la función *estadoPlaza()*, que tras realizar la consulta del sensor, incluye la información de estado de la plaza en la trama que se va a enviar hacia el nodo final. Tras incluir este valor, se llama a la función *trama()* y se pasa por parámetro el texto 'info' que indica que se debe enviar una trama de tipo '*Info Frame*'.

```
#Función para comprobar si se debe cambiar de modo (normal o nocturno)
def define_mode():
    global NM
    #Se captura el estado del RTC.
    m = rtc.now()
    #Si se llega a la hora NMStart se pasa al modo nocturno, y no se cambia de
    if((m[3] >= NMStart) and (m[3] <= NMStart + NMPeriod)):
      NM = 1
    #Cuando termina el período establecido para el modo nocturno, se inicia el
    if((m[3] > NMStart + NMPeriod) and (m[3] < NMStart)):
      NM = 0

#Función para borrar los campos de las tramas
def clearframe():
    global frame
    for i in range(0, 11):
     frame[i] = frame[i] & 0x00

#Función para borrar los campos de las tramas
def clear_bytearr():
    global data
    for i in range(0, 11):
     bytearr[i] = bytearr[i] & 0x00

#Función para evaluar el estado de la plaza de aparcamiento
def estadoPlaza():
    #global slot
    global frame
    print(slot)
    if(slot == 1):
         print('plaza ocupada')
         frame[0]=0x80
    if(slot == 0):
         print('plaza libre')
         frame[0]=0x00
    trama('info')
```

Figura 10.13: Funciones define_mode(), clearframe(), clearbytearr() y estadoPlaza().

En la siguiente función se codifica el proceso de formación de las tramas. El código de la función *formarTrama()* se adjunta en la Figura 10.14 y la Figura 10.15. Así, en la Figura 10.14 se muestra el código común a todas las tramas con las que trabaja esta función. Estas tramas son: *Info Frame*, *Keep-Alive Frame* y *Start Frame*. Todas las tramas llevan en el Byte 1 el valor del campo *frameCounter* que corresponda, además del identificador del nodo final, que se encuentra siempre en el Byte 5. El resto de Bytes se rellenan con un valor u otro, dependiendo del formato de cada tipo de trama. Por ejemplo, en el Byte 0, los primeros 4 bits se dedican a identificar el tipo de trama, por lo que como se observa en la Figura 10.14 y la Figura 10.15, se realiza una operación lógica 'OR' con el Byte 0 para incluir el tipo de trama, mientras que se evita la modificación de otros bits ya asignados.

En el caso de la trama de información no hay que añadir ninguna información adicional a la del estado de la plaza de aparcamiento, que ya fue añadida en la función *estadoPlaza()*.

En la Figura 10.14 se puede observar el proceso realizado para obtener el tiempo transcurrido desde que se inició el funcionamiento del nodo final, necesario para la trama de tipo '*Keep-Alive*'. En primer lugar, se realiza una lectura del estado del *Timer*, y posteriormente se evalúa el número de horas y minutos transcurridos según el valor de la lectura. Por último, se transfieren los valores a los Bytes 2 y 3.

```
#Función para rellenar las tramas con el contenido apropiado
def formarTrama(tipo):
    global frame
    #Se incrementa el valor del FrameCounter y se guarda en el Byte 1.
    increase_frameCounter()
    frame[1] = contador
    #Se guarda ID del nodo final en el Byte 5.
    frame[5] = id1

    #Según el tipo de trama que se pase por parámetro en la función se rellena
    #un tipo de trama u otro
    if(tipo == 'info'):
        frame[0] = frame[0] | 0x00 # Identificador del tipo de trama: Info Frame

    if(tipo == 'keep'):
        frame[0] = frame[0] | 0x01 # Identificador del tipo de trama: Keep-Alive Frame
        total = chrono.read() #Se realiza una lectura del tiempo transcurrido
        #desde que se inició el funcionamiento del nodo
        if(total >= 3600): #Si el tiempo transcurrido supera los 3600 segundos,
        #se guardan el número de horas transcurridas en la variable totalh
         totalh = total/3600
        else:   #Si no, se deja este valor a 0.
         totalh = 0
        if(total >=60): #Si el tiempo transcurrido supera los 60 segundos,
        #se guardan los minutos transcurridos en la variable totalm
         totalm = total/60
        else:   #Si no, se deja este valor a 0.
         totalm = 0
        #Se guardan las horas y los minutos transcurridos en los Bytes 2 y 3 respectivamente.
        frame[2] = int(totalh)
        frame[3] = int(totalm)

```

Figura 10.14: Función formarTrama().

En la Figura 10.15 se muestra el contenido principal de la trama de inicio. En la trama de tipo *Start Frame 2* se almacenan en los Bytes todos los parámetros de configuración. De esta manera se informa al usuario de la configuración inicial que utiliza el nodo final.

```
356     if(tipo == 'start2'):
357         frame[0] = frame[0] | 0x05 # Identificador del tipo de trama: Start Frame 2
358         #En la Start Frame 2 se informa d ela configuración inicial del nodo.
359         frame[2] = version
360         frame[3] = NMStart
361         frame[4] = NMPeriod
362         frame[6] = NMKeepAlive
363         frame[7] = NMSleepTime
364         frame[8] = SleepTime
365         frame[9] = KeepAlive
```

Figura 10.15: Función formarTrama()_2.

En la Figura 10.16 se define la función *trama()*, que se encarga de llamar a su vez a la función *formarTrama()* para posteriormente enviar la trama correspondiente a través de la función *send()*. Además, tras este envío, muestra por pantalla del terminal dicha trama. En esta función *trama()* se pasa como parámetro el tipo de trama que se va a enviar. Una vez que se envía la trama, se realiza una limpieza del *array* de bytes que contenía el mensaje con el fin de volver a utilizar la misma variable.

```
367 #Función que engloba otras funciones con el objetivo de simplificar el loop infinito
368 def trama(tipo):
369     global frame
370     if(tipo == 'info'):
371      formarTrama('info')
372     if(tipo == 'keep'):
373      formarTrama('keep')
374     if(tipo == 'start2'):
375      formarTrama('start2')
376     #Se visualiza la trama por pantalla
377     print("- SENT data = {}".format(binascii.hexlify(frame)))
378     #Se envía la trama
379     send()
380     #Se borran los campos de la trama(Se limpia el bytearray)
381     clearframe()
```

Figura 10.16: Función trama().

En este punto del *firmware* ya se han definido todas las funciones requeridas, así como las variables u objetos necesarios. Por lo tanto, se puede iniciar el flujo de funcionamiento del nodo final. En la Figura 10.17 y en la Figura 10.18 se muestra el código asociado a dicho funcionamiento. En la Figura 10.17 se puede comprobar que la primera acción que se realiza es enviar la trama de inicio '*Start Frame*', y que una vez que se ha enviado se comienza al funcionamiento cíclico dentro de un bucle infinito.

```
383  ########Arranque del funcionamiento del nodo#####################################
384  #En el arranque del nodo se envían las dos tramas de inicio
385  trama('start2')
386  #recibirUnaVez = 0
387  #Loop infinito
388  while 1:
389   #Se define en qué modo se está actualmente
390   define_mode()
391   consulta_sensor()
392   estadoPlaza()
393
394   #Se realiza una acptura del timer 'k'
395   lap = k.read()
396   #Cada 'KeepAlive' segundos se manda una trama de keep alive en modo normal
397   if(lap >= KeepAlive and NM == 0):
398    #se reinicia la cuenta y se envía la trama de Keep-Alive
399    k.reset()
400    trama('keep')
401   #Cada 'KeepAlive' segundos se manda una trama de keep alive en modo nocturno
402   if(lap >= NMKeepAlive and NM == 1):
403    #Se reincia la cuenta y se envía la trama de Keep-Alive
404    k.reset()
405    trama('keep')
406
```

Figura 10.17: Codificación del funcionamiento del nodo final.

Dentro del bucle infinito se sigue el funcionamiento especificado en el diagrama de flujo expuesto. Se define el modo de funcionamiento, se consulta la distancia entre el sensor y el objeto reflectante, y se determina el estado de la plaza de aparcamiento. Seguidamente se evalúa si se ha cumplido el tiempo de envío de trama '*Keep-Alive*' y en caso afirmativo se envía dicha trama. En caso negativo, no se envía.

En la Figura 10.18 se muestra la recepción de datos, así como si la trama recibida es correcta y el nodo final que la ha recibido es el destino de dicha trama, realizándose la reconfiguración del nodo final. Finalmente, según el modo de operación en el que se encuentra el nodo, se realiza un tipo de espera más largo o corto.

Con estas modificaciones y adaptaciones concluye la configuración del nodo final para la aplicación específica *Smart Parking*.

```
413     receive()
414     #Se imprime el bytearray recibido en la pantalla
415     print("- RECEIVED data from TTN = {}".format(binascii.hexlify(bytearr)))
416    ##################################################################################################
417     #Si el bytearray tiene contenido se analiza la trama recibida
418     if(len(bytearr) > 0):
419     #Se comprueba que la trama recibida es de configuración
420      if((bytearr[0] == 0x73) or (bytearr[0] == 0x33)): #Código-filtro = 0x33 en el Byte 0.
421      #0x33--> Campo asignarID ='no', 0x73--> Campo asignarID ='si'
422      #Se analiza si el mensaje es para este nodo: variable ID1 o para cualquier nodo receptor (0x01)
423       if(bytearr[7] == id1 or bytearr[7] == 0x01): #Si la trama recibida es para este nodo,
424       #se actualizan los parámetros configuradores del nodo
425        actualizar_config()
426        #Se imprime por pantalla los nuevos parámetros configuradores
427        #print("Nuevos parámetros: Versión: " + version + ", ID1: " + id1 + ", NMStart: " + NMStart +
428      clear_bytearr()
429
430    #Según el modo de funcionamiento el tiempo de espera(dormido)
431    # del nodo será un valor u otro
432    if(NM == 1): #Modo nocturno
433     time.sleep(NMSleepTime)
434    else: #Modo normal
435     time.sleep(SleepTime)
436
```

Figura 10.18: Codificación del funcionamiento del nodo final.

10.2 Adaptación del nodo Estación Base y del nodo *Gateway*

En este punto ya se han analizado las adaptaciones requeridas en el nodo final, por lo que en este apartado se procede a describir las adaptaciones de los otros dos nodos de la plataforma genérica, correspondientes al nodo estación base y el nodo *gateway*. Con respecto a este último, este no sufre ninguna modificación ni adaptación, ya que tal y como se ha implementado su *firmware,* cumple completamente con la funcionalidad requerida en la aplicación especifica de *Smart Parking*.

Por otra parte, el nodo estación base si requiere de varias adaptaciones que se comentarán a continuación.

Al igual que en el nodo final, se debe establecer qué tecnología de comunicación va a usar en la transmisión hacia el nodo final y en la recepción de los mensajes que envíe este. Según lo especificado al comienzo de este capítulo el nodo estación base recibe datos desde el nodo final mediante *VLC*, y transmite datos hacia el nodo final mediante *BLE*. En la Figura 10.19 se muestra esta selección de tecnología de comunicación a nivel de *firmware*.

```
#Selección tecnología de trasmisión
techselectedTX = 0x02 #BLE          0x01:VLC; 0x02: BLE; 0x03: VLC+BLE simultáneo
#Selección tecnología de recepción
techselectedRX = 0x01 #VLC          0x01:VLC; 0x02: BLE; 0x03: VLC+BLE simultáneo
```

Figura 10.19: Selección de la tecnología de comunicación para el nodo estación base.

Con respecto a la funcionalidad del nodo estación base, tal y como se indicó en el capítulo 7, se reciben datos continuamente del nodo final, y en función de si son correctos o no, se reenvían a través de la tecnología *LoRa/LoRaWAN* hacia el nodo *gateway,* o bien se muestra un mensaje de error por pantalla en el terminal, respectivamente.

Con el objetivo de realizar una prueba o simulación de envío de trama de configuración, se fuerza el envío *BLE* desde el nodo estación base al nodo final. Tras entrar en el bucle infinito 8 veces, se envía una trama de configuración, que como se verá en el apartado de validación de este mismo capítulo pretende modificar el identificador a modo de prueba de funcionamiento. El hecho de que se espere a la octava iteración se debe a que se desea dar tiempo para poder visualizar el funcionamiento del nodo previamente al envío de la trama de configuración.

```
count = 0
while 1:
    receive()
    if((frame[5] == 0)): #poner byte del id que siempre esta ocupado #falta coger de los 10 mensajes, uno solo
        print(" Recibido frame desde VLC vacio, no se reenvía por LoRa ")
    else:
        send_LoRa()
        clearframe() #Se limpia el frame
    #receive_LoRa()
    #if(len(bytearr) > 0):
    #    send()
    #    clear_bytearr() #Se limpia el bytearray
#Fuerzo el envío BLE downlink con este código
    count = count + 1
    if(count==8):
       bytearr[0] = 0x73
       bytearr[1] = 0x34
       bytearr[2] = 0x02
       bytearr[3] = 0x06
       bytearr[4] = 0x07
       bytearr[5] = 0x1E
       bytearr[6] = 0x0A
       bytearr[7] = 0x01
       bytearr[8] = 0x0A
       bytearr[9] = 0x3C
       bytearr[10] = 0x00
       #count=0
       print("- RECEIVED data from TTN = {}".format(binascii.hexlify(bytearr)))
       send_BLE()
       clear_bytearr()
```

Figura 10.20: Adaptación del firmware del nodo estación base.

Con esta modificación del *firmware* se concluye la adaptación del nodo estación base para la aplicación específica *Smart Parking* sobre la plataforma genérica propuesta en este TFM.

10.3 Adaptación del servicio *The Things Network*

El último elemento de la plataforma genérica que queda por adaptar a la solución específica es TTN. En este caso, la única modificación que se debe añadir con respecto a la configuración que se explicó en el capítulo 9, es desarrollar funciones que permitan decodificar la información recibida, y codificar las sentencias que se quieran enviar al nodo final. Es decir, con el fin de evitar el trato directo a nivel de bytes y poder visualizar el significado de las informaciones transmitidas, se desarrollan las denominadas *Payload Functions*. Estas funciones se incluyen en la pestaña *Payload Formats*, dentro de la aplicación creada en la plataforma TTN, ya que son específicas para cada aplicación. Esta es la razón por la que se deben desarrollar en función de la aplicación y no de forma genérica.

Las funciones desarrolladas para la aplicación del presente TFM son el decodificador o *decoder*, que se utiliza para convertir los bytes en los valores asociados a determinados campos (o *fields*), y el codificador o *encoder*, que se utiliza para convertir los valores de cada campo en los bytes correspondientes. De esta manera, el *decoder* se utilizará cuando se reciba una trama desde el nodo estación base, y el *encoder* se utilizaría en caso de recibir en TTN desde una aplicación externa la petición de envío *downlink* hacia el nodo estación base y por lo tanto también hacia el nodo final. Esta última *payload function* se ha desarrollado, pero no se ha llegado a implementar.

La función *Decoder* permite, a partir de la información de cada campo, mostrar los bytes recibidos por la aplicación desde el nodo estación base, a través del *nanogateway*. Hay campos asignados directamente a ciertos bits o bytes que no dependen del tipo de trama recibida, por lo que estos son los primeros en extraerse, como se muestra en la Figura 10.21.

El resto de los campos deben rellenarse en función del tipo de trama que se haya recibido. Para ello se hace uso de la estructura *switch*, como se muestra también en la Figura 10.21,

con el fin de poder determinar qué tipo de trama es la que se está recibiendo. Como se explicó en el apartado correspondiente a la implementación de la funcionalidad del nodo final, las tramas incluyen un identificador de tipo de trama en el Byte 0, y de ahí es donde se extrae la información que determina el tipo de trama. A continuación, se debe evaluar para cada trama si hay que actualizar campos, o no. Por ejemplo, para el tipo de trama *Info Frame*, se debe comprobar un bit del Byte 0 para ver si la plaza de aparcamiento está libre u ocupada. Según el valor de este bit, se actualiza el campo *ParkingSlotStatus* con el valor "Libre" u "Ocupado".

```
function Decoder(bytes, port) {
// Decode an uplink message from a buffer
// (array) of bytes to an object of fields.
 var tipoTrama = (bytes[0] & 0x0F);
 var FrameCounter = bytes[1];
 var IDnodo = bytes[5];
 var tipo;
 var ParkingSlotStatus;
 var error;

 switch (tipoTrama){
  case (0x00):
   tipo = 'Info';
   if (bytes[0] && 0x80){
    ParkingSlotStatus = 'Ocupado';
   }else{
    ParkingSlotStatus = 'Libre';
   }
   break;

  case (0x01):
   tipo = 'KeepAlive';
   var currentHours = bytes[2];
   var currentMinutes = bytes[3];
   break;

  case (0x05):
   tipo = 'Start2';
   var version = bytes[2];
   var NMStart = bytes[3];
   var NMPeriod = bytes[4];
   var NMKeepAlive = bytes[6];
   var NMSleepTime = bytes[7];
   var SleepTime = bytes[8];
   var KeepAlive = bytes[9];
   break;
  default:
   error = 'Tipo de trama errónea';
 }
```

Figura 10.21: Función Decoder.

En las tramas de tipo *Keep-Alive*, la información que se tiene que extraer de los bytes es el tiempo transcurrido desde que se inició el funcionamiento del nodo final, que se encuentran separados en horas y minutos.

En la Figura 10.21 se puede ver que para las tramas de tipo *Start Frame 2,* se extraen múltiples valores de los bytes, ya que prácticamente cada uno de ellos contiene el valor de un campo.

En caso de que la trama recibida no se corresponda con ninguna de las enunciadas anteriormente, no se extrae ningún valor más.

Por último, como se observa en la Figura 10.22, se devuelven las variables en las que se han almacenado los valores extraídos de los bytes recibidos.

```
41  return{
42   tipo: tipo,
43   FrameCounter: FrameCounter,
44   IDnodo: IDnodo,
45   ParkingSlotStatus: ParkingSlotStatus,
46   currentHours: currentHours,
47   currentMinutes: currentMinutes,
48   version: version,
49   error: error,
50   NMStart: NMStart,
51   NMPeriod: NMPeriod,
52   NMKeepAlive: NMKeepAlive,
53   NMSleepTime: NMSleepTime,
54   SleepTime: SleepTime,
55   KeepAlive: KeepAlive
56   }
57 }
```

Figura 10.22: Función Decoder_2.

Con respecto a la función *Encoder*, en ella se realiza la conversión de los valores de cada campo a bytes, con el fin de poder enviar una trama de configuración al nodo final, y que este pueda interpretarla correctamente.

Para ello se crea un vector de 11 bytes que será rellenado con los valores de los campos que se tengan en la trama de configuración. Como se puede ver en la Figura 10.23, hay valores que se pueden introducir directamente en los bytes correspondientes, mientras que otros dependen de si un campo contiene un “si” o un “no”, como es el caso del campo “*asignarID*”.

```
function Encoder(object, port) {

  // Encode downlink messages sent as
  // object to an array or buffer of bytes.
  var bytes = [00, 00, 00, 00, 00, 00, 00, 00, 00, 00, 00];

  if (object.asignarid === "si"){
    bytes[0] = 0x73;
  }else if (object.asignarid === "no"){
    bytes[0] = 0x33;
  }

  bytes[1] = object.nodeid;
  bytes[2] = object.firmwareversion;
  bytes[3] = object.nmstart;
  bytes[4] = object.nmperiod;
  bytes[5] = object.nmsleeptime;
  bytes[6] = object.nmkeepalive;
  bytes[7] = object.id2config;
  bytes[8] = object.sleeptime;
  bytes[9] = object.keepalive;

return bytes;
}
```

Figura 10.23: Función Encoder.

Una vez que se han rellenado todos los bytes de la trama de configuración, la función devuelve el *array* de *bytes* con su contenido actualizado.

Si se deseara probar inicialmente el funcionamiento de la función *Encoder*, se puede enviar una sentencia *JavaScript Object Notation* (JSON), tal y como se recibiría desde una aplicación externa. Una sentencia JSON válida podría ser la que se muestra en la Figura 10.24.

```
{ "asignarid": "no", "broadcast": "si", "id2config":2, "nodeid":2, "firmwareversion":1, "nmstart":45,
"nmperiod":30, "nmsleeptime":15, "nmkeepalive":25, "sleeptime":2, "keepalive":5}
```

Figura 10.24: Sentencia JSON.

En esta sentencia se pueden observar todos los campos con sus respectivos valores, que se pasarían a formato bytes en la función E*ncoder*.

10.4 Validación de la plataforma HW/SW sobre el *Use of Case*

En este apartado se documentan las pruebas de validación experimental de la aplicación específica *Smart Parking* que ha sido implementada sobre la plataforma genérica propuesta en el presente TFM.

En las siguientes imágenes se muestra el montaje realizado para las pruebas de validación. En la Figura 10.25 se muestran el nodo final y el nodo estación base conjuntamente. En la Figura 10.26 se muestra con mayor detalle el montaje del nodo final, del mismo modo que en la Figura 10.27 se muestra únicamente el nodo estación base y en la Figura 10.28 el nodo *gateway*.

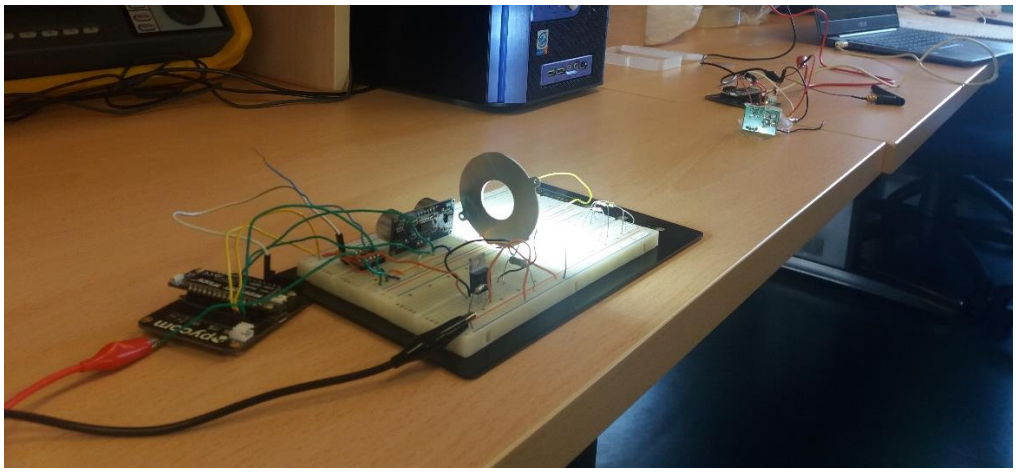


Figura 10.25: Montaje del nodo final y del nodo estación base.

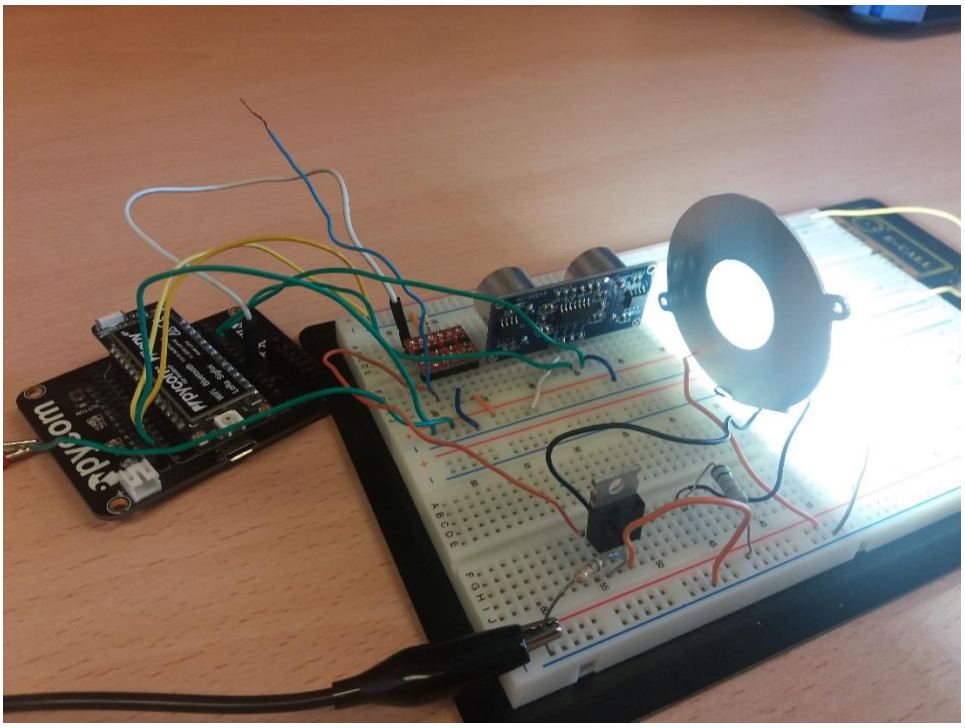


Figura 10.26: Nodo final.

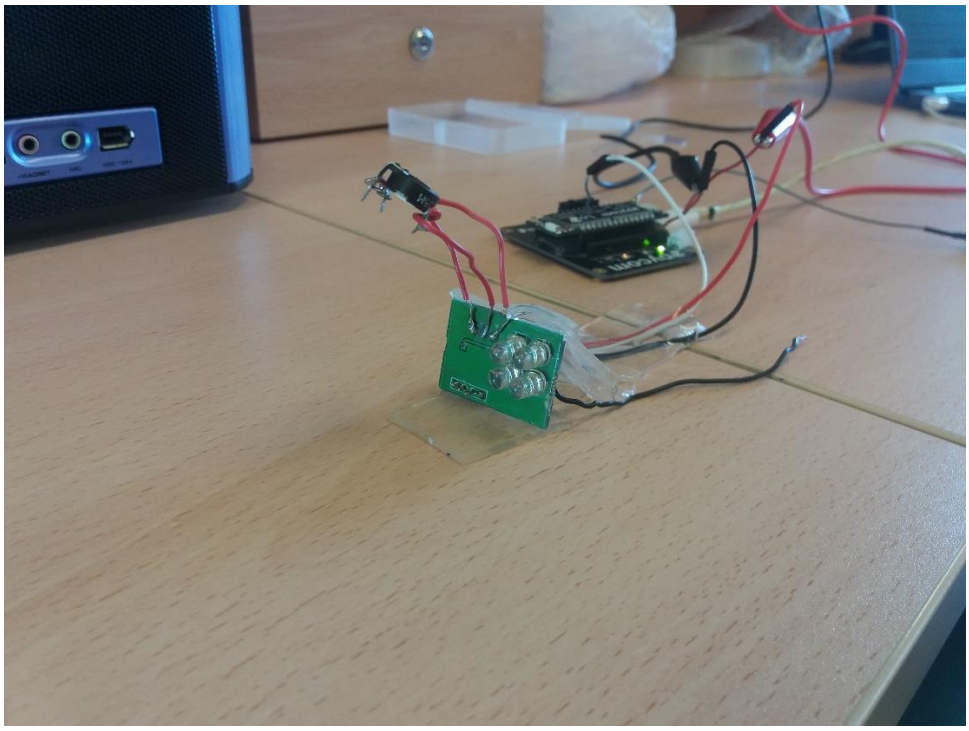


Figura 10.27: Nodo estación base.

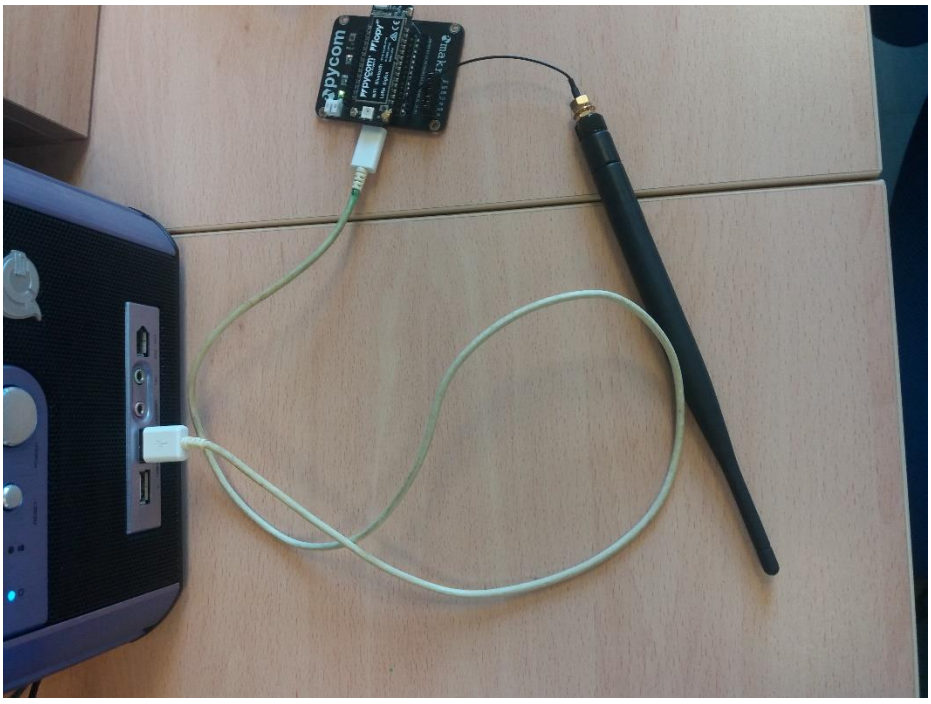


Figura 10.28: Nodo gateway.

El procedimiento seguido para realizar la prueba de funcionamiento final comienza por la preparación del proceso experimental. En esta preparación se realizan las siguientes tareas:

- Alimentación de los nodos. En la prueba realizada el nodo final, el nodo estación base y el nodo gateway son alimentados externamente mediante fuentes de alimentación o en algunos casos mediante la alimentación que provee la conexión USB nodo-PC. Realmente, el nodo final puede ser alimentado por batería, pero por comodidad en el montaje se ha decidido alimentarlo externamente.

- Posicionamiento: Se posiciona el nodo final encima de una mesa y con el sensor orientado hacia el fondo de esta. El módulo óptico transmisor del nodo final está orientado hacia el módulo óptico receptor del nodo estación base. Por lo tanto, el nodo estación base también se encuentra orientado hacia el nodo final. Se encuentran en estado de visión directa (LoS) y a unos 50 cm de separación. El nodo *gateway* se encuentra a 1 metro aproximadamente del nodo estación base.

- En esta prueba de funcionamiento se mantiene un período de envío de trama 'Keep-Alive' bajo con el fin de poder visualizar esta trama en un corto espacio de tiempo de prueba. Además, con el objetivo de conocer el estado de la plaza de aparcamiento en todo momento, no se valora si se ha producido un cambio de estado, sino que directamente se envía el estado detectado en la plaza, haya cambiado o no.

- Inicio de la aplicación en TTN. Se accede a la cuenta creada y configurada en TTN, explicada en el capítulo 9. Dentro de la cuenta se accede a la aplicación desarrollada para el presente TFM y se muestra la pestaña de 'DATA'.

- Inicio de la prueba. Se inicia el nodo *gateway* en primer lugar. Seguidamente se conecta el nodo estación base, que debe conectar mediante *LoRa/LoRaWAN* con el *gateway* para tener acceso a la red *LoRaWAN*. Seguidamente se inicia el funcionamiento del nodo final.

Con este procedimiento de preparación completado se puede dar comienzo a la prueba de validación experimental.

Para simular la ocupación de la plaza de estacionamiento se posicionará una mano enfrente del sensor durante un período de tiempo. Y para volver al estado de plaza de estacionamiento libre se quitará la mano del campo de visión del sensor.

Inicialmente los datos recibidos en TTN, sin que haya ningún objeto delante del sensor, son los que se muestran en la Figura 10.29. En ella se pueden apreciar las últimas 5 tramas recibidas. Haciendo uso del decodificador se puede ver que el ID del nodo es 52 (en byte – círculo rojo de la Figura 10.29: 0x34) y que inicialmente al tener el bit más significativo del byte 0 a '0' (círculo amarillo) el estado de la plaza es de libre. En caso de que esté ocupado este byte 0 debe valer 0x80 en vez de 0x00. En color azul se marca el campo de contador de tramas, que como se puede ver, se va incrementando de uno en uno.

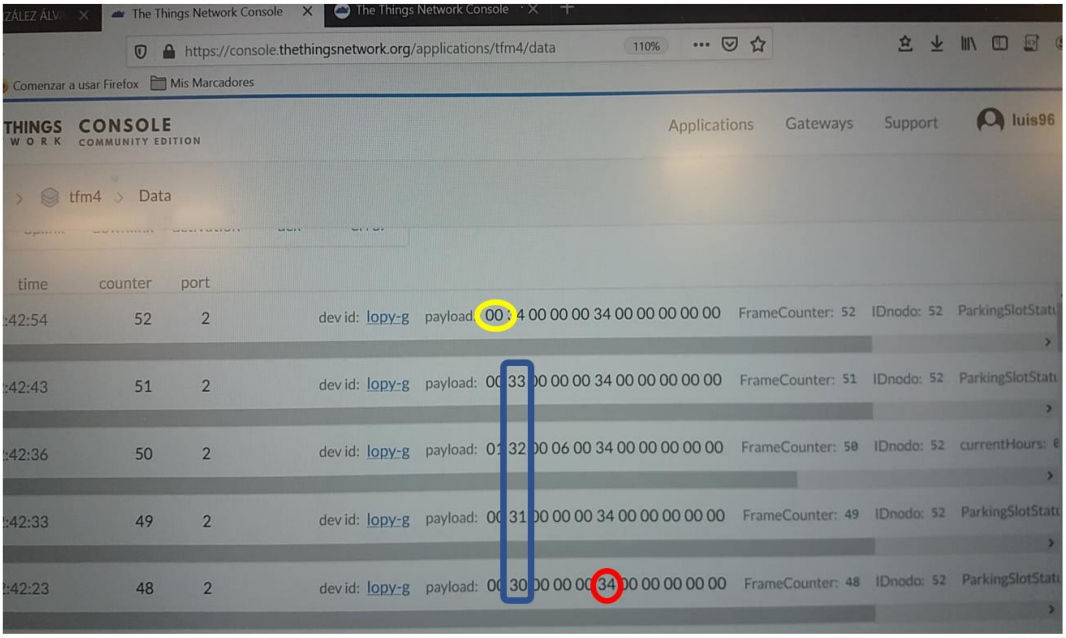


Figura 10.29: Datos recibidos en TTN sin obstáculo en el sensor.

Seguidamente, se posiciona la mano delante del sensor ultrasónico, tal y como se muestra en la Figura 10.30, y se obtiene la nueva trama de la Figura 10.31 en la que se indica que la plaza está ocupada.

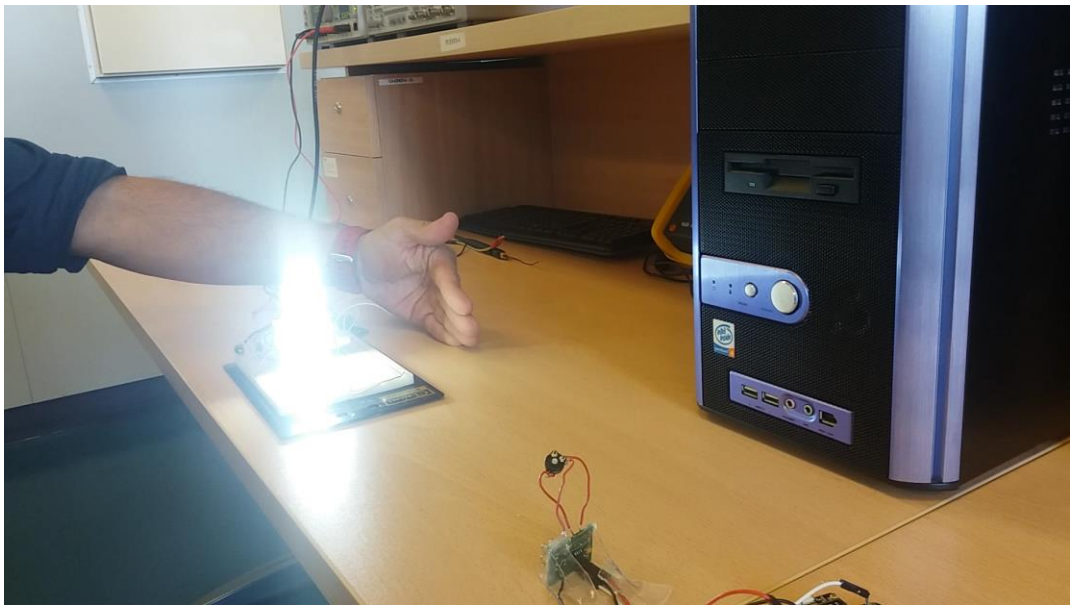


Figura 10.30: Simulación de ocupación de la plaza de estacionamiento.

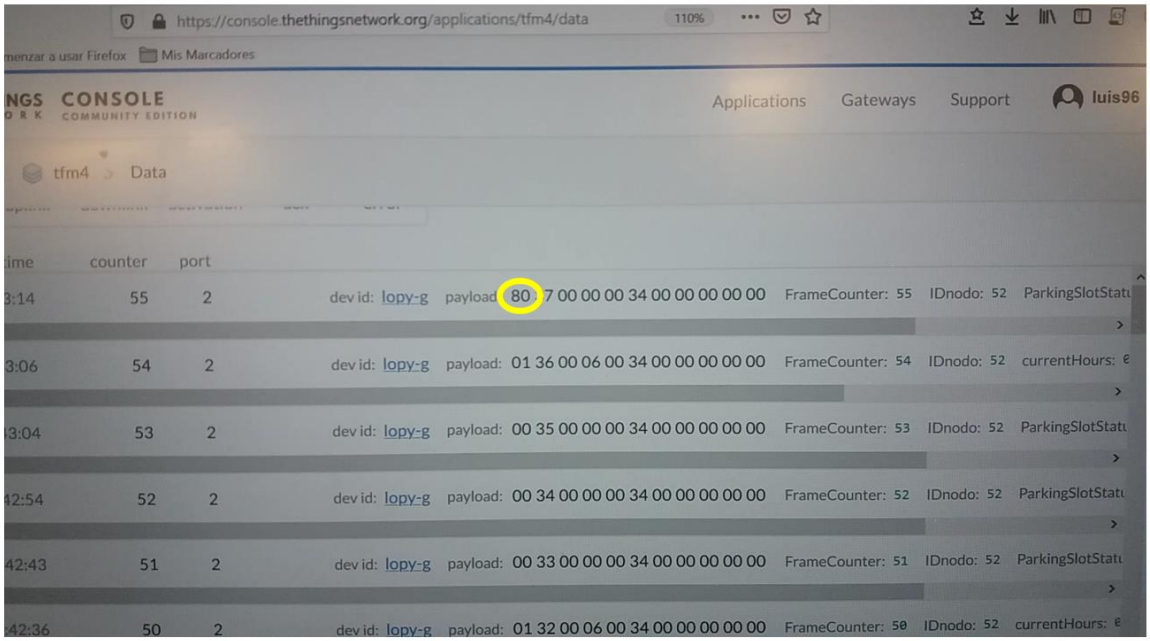


Figura 10.31: Recepción de nueva trama indicando ocupación de la plaza de estacionamiento.

De la misma forma, se retira la mano de la zona de visión del sensor, tal y como se muestra en la Figura 10.32, y se obtiene la trama de la Figura 10.33, en la que se indica que el estado de la plaza de estacionamiento vuelve a ser libre.

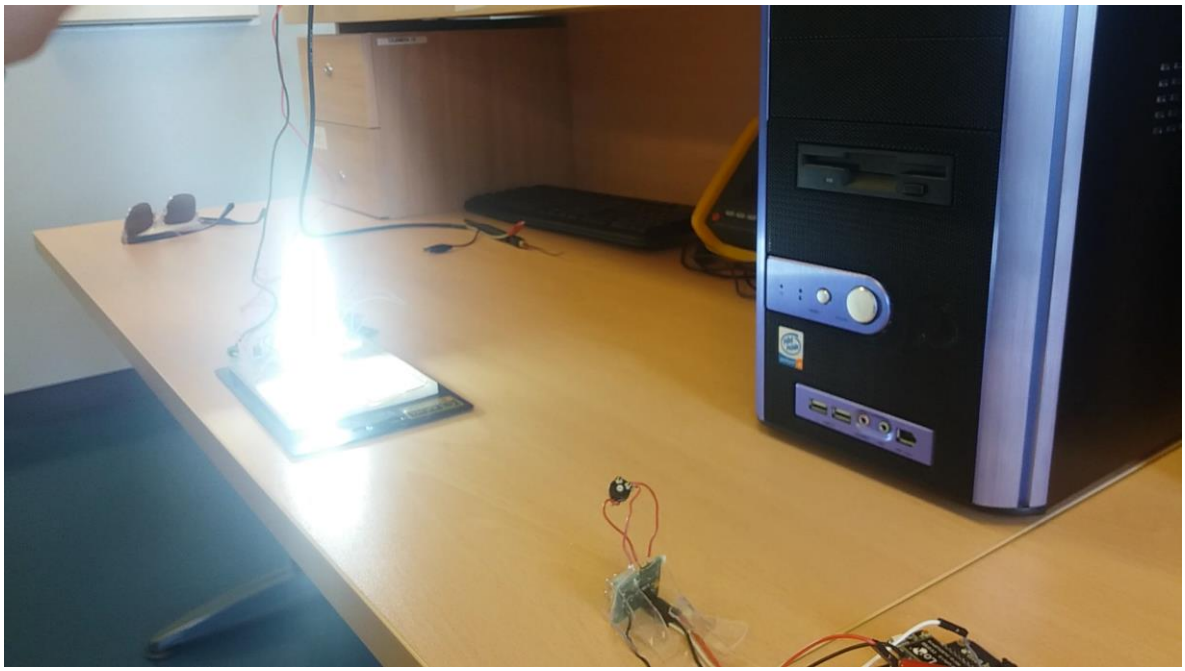


Figura 10.32: Simulación de liberación de la plaza de estacionamiento.

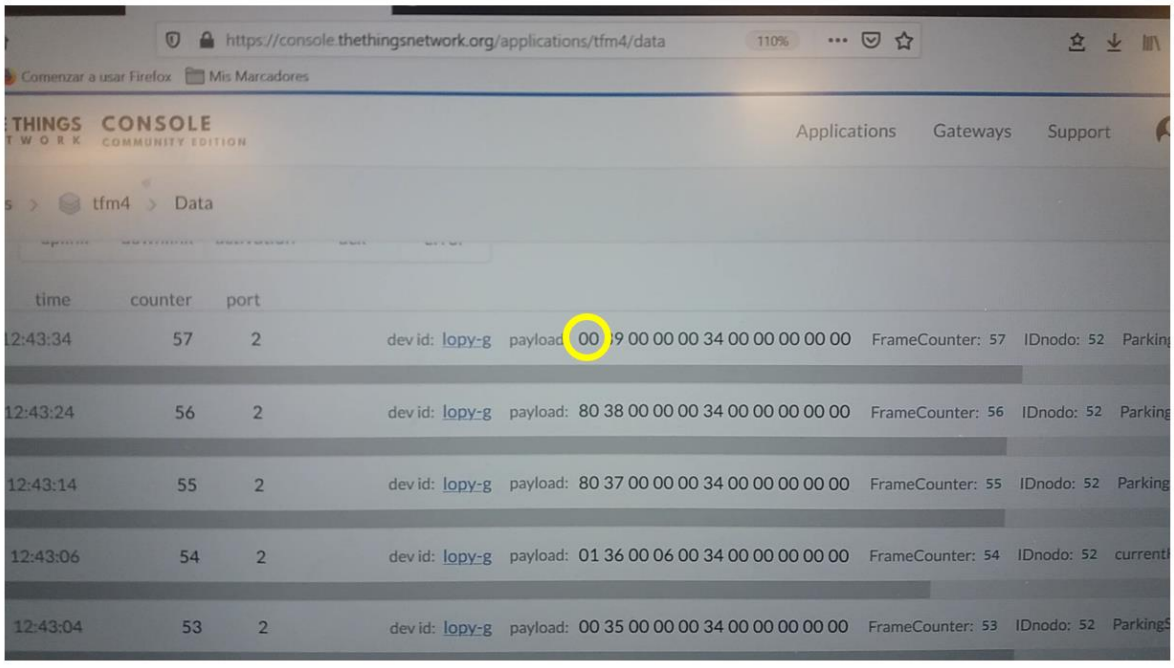


Figura 10.33: Recepción de nueva trama indicando liberación de la plaza de estacionamiento.

Con esta demostración se considera validada experimentalmente la plataforma HW/SW para el *use of case* de *Smart Parking* en el sentido ascendente de la comunicación.

Con el fin de realizar una validación de comunicación en sentido descendente, se reinicia el funcionamiento de todos los nodos y se envía una trama de configuración al nodo final

con el objetivo de modificar alguno de sus parámetros de configuración. En este caso se pretende cambiar el identificador del nodo. En concreto, se va a modificar su valor inicial de 0x05 por un valor de 0x34. En la Figura 10.34 se muestran las tramas enviadas por el nodo final antes de que el nodo final reciba la trama de configuración. Después de un tiempo, el nodo recibe la trama de configuración y se actualiza. Las nuevas tramas con el identificador del nodo actualizado al valor 0x34 se muestra en la Figura 10.35.

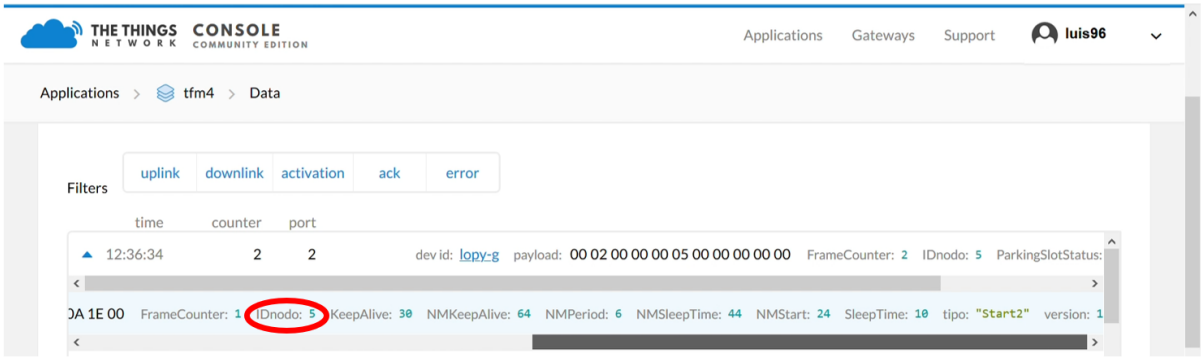


Figura 10.34: Tramas enviadas por el nodo final original.

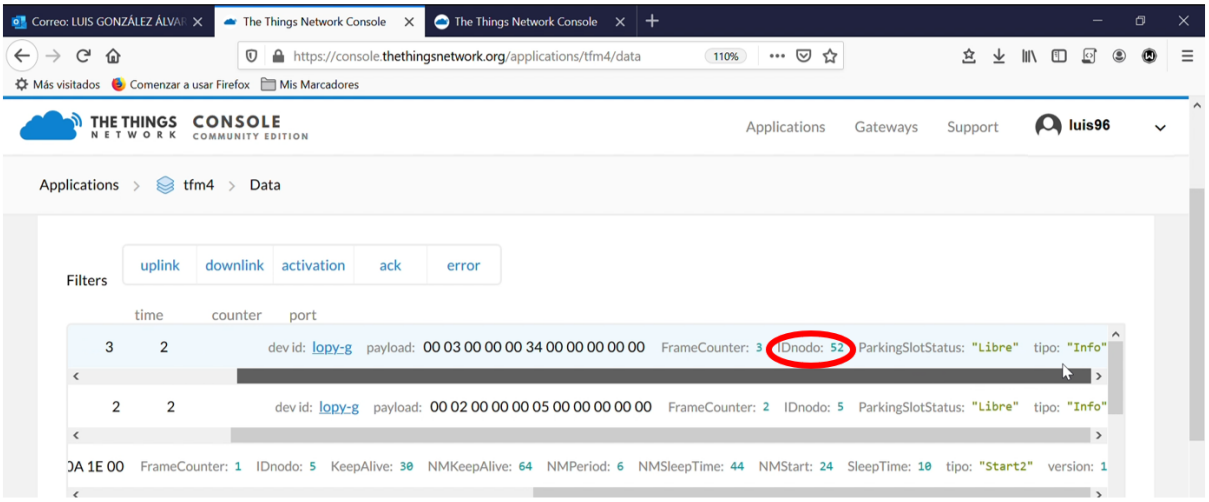


Figura 10.35: Tramas enviadas por el nodo final tras ser configurado.

Con el cambio de identificador queda reflejado el correcto funcionamiento de la trama de configuración.

Tras finalizar las dos pruebas de funcionamiento, se puede enunciar que la aplicación *Smart Parking* basada en la plataforma genérica presentada en este TFM queda validada experimentalmente.

Capítulo 11. Conclusiones y líneas futuras

11.1 Conclusiones

Tras completar el proceso de validación experimental de la solución *Smart Parking* desarrollada a partir de su implementación sobre la plataforma genérica propuesta en el presente TFM, se puede afirmar que se han cumplido los objetivos establecidos inicialmente. Así, se han estudiado las tecnologías de comunicación que se integran en la plataforma HW/SW y se han aplicado en un caso de uso concreto.

Otro objetivo cumplido ha sido el de la caracterización del sensor y los dispositivos utilizados. También se ha llevado a cabo el desarrollo del *firmware* de los diferentes nodos que forman la arquitectura de la plataforma propuesta, desde el nodo final hasta el nodo *gateway*, pasando por el nodo estación base.

Se ha integrado la solución en TTN, creando y configurando el *gateway* y la aplicación correspondiente al caso de uso en este servicio. En TTN, además, se ha desarrollado un decodificador y un codificador de tramas con el fin de acomodar la información recibida o transmitida a conveniencia.

En resumen, al haber cumplido todos los objetivos planteados, se ha logrado desarrollar una plataforma HW/SW genérica que permite comunicarse bidireccionalmente usando las

tecnologías de comunicación *BLE*, *VLC* y *LoRa/LoRaWAN*. Así mismo, esta plataforma integra la funcionalidad completa de cada módulo de comunicación, de tal forma que es aplicable a cualquier caso de uso dentro del ámbito IoT que requiera de las prestaciones que la solución desarrollada ofrece. Además, esta plataforma se ha planteado que sea de tipo *open source* con el fin de que, para cada aplicación específica que se vaya a implementar sobre la plataforma genérica, se pueda adaptarse fácilmente a los requisitos que demande cada solución IoT.

La plataforma HW/SW diseñada cuenta con varias propiedades y características que ofrecen al usuario una solución completa, inalámbrica y configurable. El desarrollo de la solución se ha realizado en todos los elementos de la arquitectura, caracterizando cada elemento que interviene en ella, desde el nodo final hasta TTN. El hecho de conocer cada elemento permite mejorar la solución sin generar dependencias entre diferentes bloques de la plataforma HW/SW.

Además, la solución desarrollada es inalámbrica, tanto en corto como en largo alcance. La comunicación entre el nodo final y el nodo estación base se realiza a través de enlaces *VLC* y *BLE,* por lo que esta conexión inalámbrica es de corto alcance, pero la comunicación entre el nodo estación base y el nodo *gateway* sí es de largo alcance. Este último tramo de comunicación permite adaptar la plataforma en diferentes situaciones como, por ejemplo, en grandes zonas habilitadas para el estacionamiento, donde un sistema similar pero cableado supondría un coste y un impacto visual importante. Que la solución sea de este tipo favorece claramente la instalación, y el mantenimiento, además de la disminución del impacto visual. Adicionalmente, no se requiere de conectividad a Internet en la zona de estacionamiento, ya que únicamente se debe contar con acceso a Internet en el punto en el que se vaya a ubicar el nodo *gateway*.

Una de las características más importantes de la solución propuesta es que es configurable. Es decir, el administrador del sistema puede configurar la solución con los parámetros que más se adecuen a las necesidades de su solución IoT.

Otra propiedad de la solución propuesta es que es escalable. La aplicación se ha prototipado para soportar comunicaciones con un nodo final y una estación base, no

obstante, siguiendo políticas de escalabilidad, se puede desarrollar una solución multinodo.

Con respecto al *use of case* planteado, con esta solución se logran grandes beneficios, tanto sociales como económicos. Una aplicación *Smart Parking* como la realizada permite ahorrar tiempo de circulación en busca de aparcamiento a los usuarios, disminuyendo de esta manera su estrés y nerviosismo al volante. Como consecuencia de todo esto, se reduce el número de estacionamientos ilegales y la congestión en las carreteras. La disminución del tiempo de circulación de los vehículos también produce un beneficio medioambiental, pues se reduce la emisión de gases contaminantes. En definitiva, los beneficios de una solución como la desarrollada son diversos y numerosos.

A continuación, se exponen las conclusiones obtenidas al finalizar la realización del Trabajo Fin de Máster, desde la perspectiva de la experiencia adquirida en su desarrollo:

- Los dispositivos LoPy de Pycom funcionan perfectamente en aplicaciones IoT como la desarrollada. Además, un aspecto favorable de la selección de estos dispositivos ha sido que cuenta con una gran cantidad de documentación en la página oficial de la empresa de Pycom, así como en sus foros.

- El uso de un módulo receptor óptico muy directivo en la comunicación *VLC* propicia que se deba realizar una instalación precisa de la ubicación de los nodos finales y estación base, con el fin de lograr una visión directa (LoS) entre ambos, y que así, la comunicación sea adecuada.

- El estudio y caracterización de los sensores es un factor clave para el éxito de una solución de este tipo. Se debe realizar un estudio previo de la zona de instalación de los sensores con el fin de determinar el valor umbral o de comparación más adecuado que se definirá posteriormente en el *firmware*.

- El hecho de que no existieran librerías del sensor desarrollado en lenguaje *MicroPython* para su integración con los dispositivos de Pycom, ha obligado

a desarrollar dichas librerías a partir de las especificaciones de los sensores facilitadas por los fabricantes.

- El uso de un *nanogateway* en lugar de un *gateway* completo reduce el coste de la solución significativamente, pero limita sus posibilidades de comunicación, y prescinde de utilidades como por ejemplo el *ADR* de *LoRaWAN*, que aumentan la eficiencia de la red.

- La plataforma TTN facilita la creación de aplicaciones IoT y permite visualizar cómodamente el flujo de datos de cada *gateway* o aplicación registrada. Además, incorpora una documentación adicional que facilita su uso.

- Se ha demostrado que la integración de las tecnologías de comunicación *VLC*, *BLE* y *LoRa/LoRaWAN* en una única plataforma HW/SW para aplicaciones IoT es posible.

11.2 Líneas futuras

El trabajo realizado supone la combinación de conceptos relacionados con IoT, las tecnologías de comunicación *VLC*, *BLE* y *LoRa/LoRaWAN*, y *Smart Parking*. A partir de esta combinación se ha ideado una solución que se ha visto reflejada de manera práctica en un prototipo de la solución. No obstante, no deja de ser la aplicación de una serie de conceptos teóricos sobre varios dispositivos, y bajo una perspectiva de investigación básica. Por lo tanto, se puede pensar más adelante en esta aplicación como una plataforma HW/SW profesional para una solución IoT completa. Es decir, el siguiente paso, o línea de futuro puede ser el desarrollo de una solución comercial basada en la plataforma prototipada en el presente TFM.

A continuación, se describen las ideas principales que se considera que se pueden desarrollar como continuación de este Trabajo Fin de Máster:

- El nodo final y el nodo estación base desarrollado no cuentan con una cubierta de protección adecuada como para desempeñar su función sin

peligrar su integridad en condiciones de exterior. Es decir, se plantea realizar el desarrollo de un sistema de protección o *housing* para el nodo final y el nodo estación base que permita mantenerlo en perfecto estado y sin disminuir su funcionalidad, se plantea como otra línea posible de desarrollo.

- La integración de módulos de comunicación *VLC* en soluciones IoT no es trivial, y en el futuro se puede mejorar este binomio en términos de eficiencia energética o alcance.

- En momentos puntuales, ante la aparición de una ráfaga del mismo valor binario en el módulo transmisor óptico, se produce una situación de parpadeo. Se puede implementar un código Manchester que evite que se produzca esta circunstancia.

- Aprovechar la instalación de iluminación para soluciones IoT basadas en *VLC* puede ser una solución óptima para aplicaciones como la presentada en el *use of case* del presente TFM.

- En cuanto a la solución *Smart Parking* del *use of case*, se puede desarrollar una aplicación externa a TTN dónde el usuario pueda visualizar el estado de las plazas de estacionamiento de un *parking* completo.

- De mismo modo que en el caso del usuario, se puede desarrollar una aplicación externa a TTN que permita al administrador del servicio de estacionamiento configurar la plataforma de forma remota con los parámetros de configuración que mejor desempeño energético o funcional puedan dar a su sistema.

Bibliografía

[1] Statista, “Internet de las cosas: dispositivos interconectados en el mundo 2020 | Estadística,” 2018. [Online]. Available: https://es.statista.com/estadisticas/638100/internet-de-las-cosas-numero-de-dispositivos-conectados-en-todo-el-mundo--2020/. [Accessed: 26-May-2018].

[2] L. E. M. Matheus, A. B. Vieira, L. F. M. Vieira, M. A. M. Vieira, and O. Gnawali, “Visible Light Communication: Concepts, Applications and Challenges,” *IEEE Commun. Surv. Tutorials*, pp. 1–1, 2019.

[3] D. Agrawal, A. Mishra, K. Springborn, S. Banerjee, and S. Ganguly, “Dynamic interference adaptation for wireless mesh networks,” in *2006 2nd IEEE Workshop on Wireless Mesh Networks*, 2006, pp. 33–37.

[4] D. A. Basnayaka and H. Haas, “Hybrid RF and VLC Systems: Improving User Data Rate Performance of VLC Systems,” in *2015 IEEE 81st Vehicular Technology Conference (VTC Spring)*, 2015, pp. 1–5.

[5] Zengtao Feng, Lingfei Mo, and Meng Li, “Analysis of low energy consumption wireless sensor with BLE,” in *2015 IEEE SENSORS*, 2015, pp. 1–4.

[6] J. Finnegan, S. Brown, and R. Farrell, “Modeling the Energy Consumption of LoRaWAN in ns-3 Based on Real World Measurements,” in *2018 Global Information Infrastructure and Networking Symposium (GIIS)*, 2018, pp. 1–4.

[7] B. Islam, M. T. Islam, J. Kaur, and S. Nirjon, “LoRaIn: Making a Case for LoRa in Indoor Localization,” in *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 2019, pp. 423–426.

[8] Á. Caamaño Lourido, “Interfaz VLC-Sigfox o VLC-LoRa con plataforma de gestión de sensores.” Las Palmas de Gran Canaria, 2019.

[9] “The LoPy4 is a quadruple bearer MicroPython enabled development board including LoRa, Sigfox, WiFi and Bluetooth.” [Online]. Available: https://pycom.io/product/lopy4/. [Accessed: 12-Sep-2019].

[10] L. González Álvarez, “Desarrollo de una plataforma HW/SW de Smart Parking basada en tecnología LoRa/LoRaWAN.” Universidad de Las Palmas de Gran Canaria, Las Palmas de Gran Canaria, 2018.

[11] IoT Analytics, “State of the IoT 2018: Number of IoT devices now at 7B – Market accelerating.” [Online]. Available: https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/. [Accessed: 14-Oct-2019].

[12] D. Benítez Machado, “Propuesta de arquitectura para Internet de las Cosas,” *Res. Gate*, 2016.

[13] R. Saverio, “5 Key Insights from 350+ Smart City IoT Projects,” *IoT Analytics*, 2019. [Online]. Available: https://iot-analytics.com/5-key-insights-from-350-smart-city-iot-projects/. [Accessed: 19-Oct-2019].

[14] K. Lasse Lueth, “40+ Emerging IoT Technologies you should have on your radar,” *IoT Analytics*, 2019. [Online]. Available: https://iot-analytics.com/40-emerging-iot-technologies-you-should-have-on-your-radar/. [Accessed: 19-Oct-2019].

[15] S. Noguchi, M. Niibori, E. Zhou, and M. Kamada, “Student Attendance Management System with Bluetooth Low Energy Beacon and Android Devices,” in *2015 18th International Conference on Network-Based Information Systems*, 2015, pp. 710–713.

[16] C. Swedberg, “Nicklaus Children’s Hospital Uses NFC Tags, BLE Beacons to Manage Inspections, Assets - 2016-04-26 - Page 1 - RFID Journal,” 2016. [Online]. Available: https://www.rfidjournal.com/articles/view?14394/. [Accessed: 21-Oct-2019].

[17] G. Cerruela García, I. Luque Ruiz, and M. Gómez-Nieto, “State of the Art, Trends and Future of Bluetooth Low Energy, Near Field Communication and Visible Light Communication in the Development of Smart Cities,” *Sensors*, vol. 16, no. 11, p. 1968, Nov. 2016.

[18] H. Sawant, Jindong Tan, Qingyan Yang, and Qizhi Wang, “Using Bluetooth and sensor networks for intelligent transportation systems,” in *Proceedings. The 7th International IEEE Conference on Intelligent Transportation Systems (IEEE Cat. No.04TH8749)*, pp. 767–772.

[19] J. Lin, T. Talty, and O. Tonguz, “On the potential of bluetooth low energy technology for vehicular applications,” *IEEE Commun. Mag.*, vol. 53, no. 1, pp. 267–275, Jan. 2015.

[20] P. K. Volam, A. R. Kamath, and S. S. Bagi, “A system and method for transmission of traffic sign board information to vehicles and relevance determination,” in *2014 International Conference on Advances in Electronics Computers and Communications*, 2014, pp. 1–6.

[21] S. Courreges, S. Oudji, V. Meghdadi, C. Brauers, and R. Kays, “Performance and interoperability evaluation of radiofrequency home automation protocols and Bluetooth Low Energy for smart grid and smart home applications,” in *2016 IEEE International Conference on Consumer Electronics (ICCE)*, 2016, pp. 391–392.

[22] W. Narzt, S. Mayerhofer, O. Weichselbaum, S. Haselbock, and N. Hofler, “Be-In/Be-Out with Bluetooth Low Energy: Implicit Ticketing for Public Transportation Systems,” in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, 2015, pp. 1551–1556.

[23] K. Warmerdam, A. Pandharipande, and D. Caicedo, “Connectivity in IoT indoor lighting systems with visible light communications,” in *2015 IEEE Online Conference on Green Communications (OnlineGreenComm)*, 2015, pp. 47–52.

[24] S. V. Tiwari, A. Sewaiwar, and Y.-H. Chung, “Smart home technologies using Visible Light Communication,” in *2015 IEEE International Conference on Consumer Electronics (ICCE)*, 2015, pp. 379–380.

[25] Y. H. Kim, W. A. Cahyadi, and Y. H. Chung, “Experimental Demonstration of VLC-Based Vehicle-to-Vehicle Communications Under Fog Conditions,” *IEEE Photonics J.*, vol. 7, no. 6, pp. 1–9, Dec. 2015.

[26] T. Szili, B. Matolcsy, and G. Fekete, “Water pollution investigations by underwater visible light communications,” in *2015 17th International Conference on Transparent Optical Networks (ICTON)*, 2015, pp. 1–4.

[27] eSMARTCITY.es, “Una urbanización inteligente en Líbano que se centra en garantizar la calidad del aire ESMARTCITY,” 2017. [Online]. Available:

https://www.esmartcity.es/2017/12/05/urbanizacion-inteligente-libano-se-centra-garantizar-calidad-del-aire. [Accessed: 26-May-2018].

[28] InteliLIGHT, “Street Lighting Remote Management.” [Online]. Available: https://intelilight.eu/. [Accessed: 26-May-2018].

[29] ONU, “World Urbanization Prospects (WUP) 2018,” 2018.

[30] G. Rozas, “Efectos psicosociales , ciudad y calidad de vida P,” *Interv. Psicosoc.*, vol. 11, pp. 229–243, 2002.

[31] M. Bouskela, M. Casseb, S. Bassi, C. Luca De, and M. Facchina, “La ruta hacia las Smart Cities,” p. 148, 2016.

[32] B. Cohen, “The Top 10 Smart Cities On The Planet,” *Co.Design*, 2012. [Online]. Available: https://www.fastcodesign.com/1679127/the-top-10-smart-cities-on-the-planet. [Accessed: 26-May-2018].

[33] “IESE Cities in Motion Index 2019 | Cities in Motion.” [Online]. Available: https://blog.iese.edu/cities-challenges-and-management/2019/05/10/iese-cities-in-motion-index-2019/. [Accessed: 28-Oct-2019].

[34] ECUBE Labs, “Soluciones inteligentes de gestión de residuos | Ecube Labs.” [Online]. Available: https://www.ecubelabs.com/es/. [Accessed: 26-May-2018].

[35] SmartSantander, “SmartSantander.” [Online]. Available: http://www.smartsantander.eu/. [Accessed: 26-May-2018].

[36] E.B, “El transporte es responsable del 41% de las emisiones contaminantes en Madrid | El Boletin,” *El Boletín*, Jul-2017.

[37] AHK, “Bosch simplifica y automatiza la búsqueda de aparcamiento,” 2017. [Online]. Available: http://www.ahk.es/es/comunicacion/noticias/noticias/artikel/bosch-simplifica-y-automatiza-la-busqueda-de-aparcamiento/?no_cache=1&cHash=2b200b978297673a8a2cbdc9515fc2b7. [Accessed: 26-May-2018].

[38] M. Castro, “Smart Parking: Qué es y Cómo nos Beneficia — SITT,” *SITT*, 2017. [Online]. Available: https://www.sittycia.com/blog/2017/4/28/smart-parking-qu-

es-y-cmo-nos-beneficia. [Accessed: 26-May-2018].

[39] Smart Parking, "Car Parking Solutions." [Online]. Available: https://www.smartparking.com/. [Accessed: 26-May-2018].

[40] i+D3, "3SCPark: IoT Smart parking para la ciudad inteligente." [Online]. Available: https://imasdetres.com/3scpark-smartcity-de-aparcamiento/. [Accessed: 26-May-2018].

[41] C. Yuan, L. Fei, C. Jianxin, and J. Wei, "A smart parking system using WiFi and wireless sensor network," in *2016 IEEE International Conference on Consumer Electronics-Taiwan, ICCE-TW 2016*, 2016.

[42] "Espectro electromagnético | HANNA® instruments Guatemala." [Online]. Available: https://hannainst.com.gt/boletines/espectro-electromagnetico/. [Accessed: 04-Nov-2019].

[43] V. M. Melián Santana, "Sistemas Ópticos No Guiados." 2015.

[44] "Equivalencia de Lumen a Vatios - LLUMOR: Tienda online radiadores bajo consumo e iluminación LED." [Online]. Available: http://www.llumor.es/info-led/equivalencia-de-lumen-a-vatios. [Accessed: 26-Aug-2019].

[45] B. Llanos Flores, "Visible light communications (VLC)," Jan. 2015.

[46] "Codificación Manchester - Wikipedia, la enciclopedia libre." [Online]. Available: https://es.wikipedia.org/wiki/Codificación_Manchester. [Accessed: 28-Aug-2019].

[47] T.-H. Kwon, J.-E. Kim, and K.-D. Kim, "Time-Sharing-Based Synchronization and Performance Evaluation of Color-Independent Visual-MIMO Communication," *Sensors (Basel).*, vol. 18, no. 5, 2018.

[48] P. Das, B.-Y. Kim, Y. Park, and K.-D. Kim, "Color-independent VLC based on a color space without sending target color information," *Opt. Commun.*, vol. 286, pp. 69–73, Jan. 2013.

[49] B. Lorenzo Grandes, "Estudio del Estado del Arte de los sistemas de comunicaciones por luz visible (VLC)." Sevilla, 2016.

[50] A. Tamayo Balas, “Estudio del estándar 802.15.7 del IEEE sobre sistemas de comunicación por luz visible.” Sevilla, 2016.

[51] T. Borogovac, M. B. Rahaim, M. Tuganbayeva, and T. D. C. Little, “‘Lights-off’ visible light communications,” in *2011 IEEE GLOBECOM Workshops (GC Wkshps)*, 2011, pp. 797–801.

[52] Pycom, “Pycom - Next Generation Internet of Things Platform.” [Online]. Available: https://pycom.io/. [Accessed: 26-May-2018].

[53] Pycom, “7.1.3 LoPy · Pycom Documentation.” [Online]. Available: https://docs.pycom.io/chapter/datasheets/development/lopy.html. [Accessed: 26-May-2018].

[54] Sigfox, “Sigfox - The Global Communications Service Provider for the Internet of Things (IoT).” [Online]. Available: https://www.sigfox.com/en. [Accessed: 26-May-2018].

[55] RS-Online, “LoPy | LoPy IoT LoRa WiFi BLE Development Board | Pycom.” [Online]. Available: https://es.rs-online.com/web/p/kits-de-desarrollo-de-radio-frecuencia/1259532/. [Accessed: 26-May-2018].

[56] Pycom, “Expansion Boards & Shields.” [Online]. Available: https://pycom.io/solutions/expansion-boards-shields/. [Accessed: 26-May-2018].

[57] GitHub, “Atom.” .

[58] GitHub, “Getting Started : Why Atom.” [Online]. Available: https://atom.io/docs/latest/getting-started-why-atom#the-native-web. [Accessed: 26-May-2018].

[59] GitHub, “Pymakr Atom Package.” .

[60] Pycom, “10.3 REPL vs Scripts · Pycom Documentation.” [Online]. Available: https://docs.pycom.io/chapter/docnotes/replscript.html. [Accessed: 26-May-2018].

[61] Sparkfun, “Logic Level Converter - Bi-Directional - BOB-12009 - SparkFun Electronics.” [Online]. Available: https://www.sparkfun.com/products/12009. [Accessed: 10-Dec-2019].

[62] "2019 Bluetooth Market Update | Bluetooth Technology Website." [Online]. Available: https://www.bluetooth.com/bluetooth-resources/2019-bluetooth-market-update/?utm_campaign=bmu&utm_source=internal&utm_medium=web&utm_content=2019bmu-homepage-hero. [Accessed: 11-Aug-2019].

[63] F. J. Dian, A. Yousefi, and S. Lim, "A practical study on Bluetooth Low Energy (BLE) throughput," in *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 2018, pp. 768–771.

[64] D. Tauchmann, A. S.-I. J. of E. and Electrical, and undefined 2015, "Experiences and measurements with bluetooth low energy (ble) enabled and smartphone controlled embedded applications," *ijeee.net*.

[65] W. Stallings, "Data and Computer Communications - Spread Spectrum."

[66] "GAP | Introduction to Bluetooth Low Energy | Adafruit Learning System." [Online]. Available: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gap. [Accessed: 12-Aug-2019].

[67] "GATT Specifications | Bluetooth Technology Website." [Online]. Available: https://www.bluetooth.com/specifications/gatt/. [Accessed: 13-Aug-2019].

[68] "Home page | LoRa Alliance™." [Online]. Available: https://lora-alliance.org/. [Accessed: 26-May-2018].

[69] Libelium, "LoRa vs LoRaWAN tutorial." [Online]. Available: http://www.libelium.com/development/waspmote/documentation/lora-vs-lorawan/. [Accessed: 26-May-2018].

[70] "LoRaWAN y LoRa." [Online]. Available: http://lorawan.es/. [Accessed: 26-May-2018].

[71] Orange, "LoRa Device Developer Guide," p. 42, 2016.

[72] B. Cendón, "Las redes más usadas en IoT (Sigfox, LoRa, NB-IoT, LTE Cat M)," 2017. [Online]. Available: http://www.bcendon.com/las-redes-mas-usadas-en-el-iot/. [Accessed: 26-May-2018].

[73] RS Components, “LoRaWAN.” [Online]. Available: https://es.rs-online.com/web/generalDisplay.html?id=i%2Flora. [Accessed: 26-May-2018].

[74] LoRa Alliance, “About LoRaWAN.” [Online]. Available: https://lora-alliance.org/about-lorawan. [Accessed: 26-May-2018].

[75] Digikey Electronics, “Descripción general de la plataforma LoRa,” Jun-2017.

[76] L. Aliance, “LoRaWAN ™ 101, A technical introduction,” p. 39, 2017.

[77] J. Lampe and Z. Ianelli, “Introduction to Chirp Spread Spectrum (CSS) Technology,” no. November, pp. 1–28, 2003.

[78] Y. A. Hussin, “Chirp-Spread-Spectrum Modulation,” pp. 1–38, 2015.

[79] Semtech, “SX1276/77/78/79 Datasheet,” 2016.

[80] A. Augustin, J. Yi, T. Clausen, and W. Townsley, “A Study of LoRa: Long Range & Low Power Networks for the Internet of Things,” *Sensors*, vol. 16, no. 9, p. 1466, Sep. 2016.

[81] S. N, L. M, E. T, K. T, and H. O, “LoRaWAN ™ Specification,” 2015.

[82] I. O. Monfort, “Estudio de la arquitectura y el nivel de desarrollo de la red LoRaWAN y de los dispositivos LoRa.,” 2017.

[83] The Things Network, “The future of single-channel gateways - Gateways / Single Channel Gateway,” 2017. [Online]. Available: https://www.thethingsnetwork.org/forum/t/the-future-of-single-channel-gateways/6590/2. [Accessed: 26-May-2018].

[84] Pycom, “5.3.6 LoRaWAN Nano-Gateway · Pycom Documentation.” [Online]. Available: https://docs.pycom.io/chapter/tutorials/lora/lorawan-nano-gateway.html. [Accessed: 26-May-2018].

[85] GitHub, “Pycom-libraries,” 2018. [Online]. Available: https://github.com/pycom/pycom-libraries/tree/master/examples/lorawan-nano-gateway. [Accessed: 26-May-2018].

[86] TTN, “The Things Network.” [Online]. Available:

https://www.thethingsnetwork.org/. [Accessed: 26-May-2018].

[87] Batteries4pro, “Batería de 3.7V 180mAh Li - Po para auriculares Plantronics CS60 y C65.” [Online]. Available: https://www.batteries4pro.com/es/comunicacion/Telefonía/3,3992-batería-de-37v-180mah-li-po-para-auriculares-plantronics-cs60-y-c65-4894128110941.html. [Accessed: 26-May-2018].

Pliego de condiciones

El Pliego de condiciones expone las condiciones bajo las que se ha desarrollado el presente Trabajo Fin de Máster. A continuación, se muestra el conjunto de herramientas *hardware*, *software* y *firmware* empleadas durante su realización.

PC.1 Condiciones Hardware

En la Tabla PC.P-1 se recogen los equipos y dispositivos *hardware* utilizados con sus principales características.

Equipo/Dispositivo	Modelo	Fabricante/Comerciante
Ordenador	-CPU *Intel* Core i3 a 2.2 GHz -4 GB RAM -320 GB HDD -2 puertos USB	Asus
LoPy	-LoPy v1.0 -LoPy v4.0	Pycom
Sensor ultrasónico	-HC-sr04	Electric Freaks
Batería LiPo	-BAT573 -3.7V -180mAh -0.7Wh	Electronic NIMO
Expansion Board	-V2.0	Pycom
Transistor MOSFET	-IRF840	LOPACAN Electrónica S.L.
Transistor BJT	-BC547C	LOPACAN Electrónica S.L.
Diodos LED	-Blanco 5 mm -Rojo 5 mm -Infrarrojo 5 mm	LOPACAN Electrónica S.L.

Lámparas	-1 lámpara de 5V -1 lámpara de 12 V	LOPACAN Electrónica S.L.
Resistencias	-Fijas -Potenciómetro:	LOPACAN Electrónica S.L.
Fototransistor	-BPW40	LOPACAN Electrónica S.L.
Fotodiodo	-PIN S5107	HAMAMATSU
Amplificador operacional	-TL082	LOPACAN Electrónica S.L.
Convertidor de nivel lógico	-Bidirectional Logic Level Converter	Sparkfun
Protoboard	-EIC-104	ANESCO S.L.
Generador de funciones	-HAMEG HM8131-2	Rohde & Schwarz
Fuente de alimentación	-HAMEG HM8142	Rohde & Schwarz
Osciloscopio	-HAMEG HM 407-2	Rohde & Schwarz

Tabla PC. 1: Condiciones Hardware.

PC.2 Condiciones Software

En la Tabla PC.P-2 se recogen las aplicaciones *software* utilizadas, con su versión correspondiente.

Aplicación	**Versión**	**Desarrollador/Comerciante**
Sistema operativo	Microsoft Windows 10 Home	Microsoft
Microsoft Office	2016	Microsoft
Microsoft Visio	2016	Microsoft
Microsoft Project	2016	Microsoft

Atom	v1.41.0	GitHub Inc.
Plataforma TTN	-	The Things Network
Pymakr	v1.5.6	Pycom
Mozilla Firefox	v71.0	Moz://a
Adobe Reader	v19.021.20058.31925	Adobe Systems Software Ireland Ltd.

Tabla PC. 2: Condiciones Software.

PC.3 Condiciones Firmware

En la Tabla PC.P-3 se muestra el *firmware* utilizado y su versión.

Firmware	**Versión**
LoPy	v1.17.3.b1

Tabla PC. 3: Condiciones Firmware.

Presupuesto

Este capítulo presenta el presupuesto que recoge los gastos generados en la realización del presente Trabajo Fin de Máster. Dicho presupuesto se divide en las siguientes partes:

- Trabajo tarifado por tiempo empleado.
- Amortización del inmovilizado material, dividida a su vez en:
    - Amortización del material *hardware*.
    - Amortización del material *software*.
- Redacción de la documentación.
- Derechos de visado del COIT (Colegio Oficial de Ingenieros de Telecomunicación).
- Gastos de tramitación y envío.
- Material fungible.

Finalmente, cuando se tengan analizados todos los puntos que forman el presupuesto, se aplicarán los impuestos vigentes y se procederá a la obtención del coste total del TFM.

P.1 Trabajo tarifado por tiempo empleado

Este concepto contabiliza los gastos que corresponden a la mano de obra, según el salario correspondiente a la hora de trabajo de un Ingeniero de Telecomunicación. Para el cálculo de este coste se utiliza la fórmula de la Ecuación P.1:

$$H = C_t \cdot 74{,}88 \cdot H_n + C_t \cdot 96{,}72 \cdot H_e \quad \text{(P.1)}$$

Donde:

- H representa los honorarios totales por el tiempo dedicado.
- H_n detalla el número de horas normales trabajadas dentro de la jornada laboral.
- C_t es un factor de corrección en función del número de horas trabajadas.
- H_e detalla el número de horas especiales trabajadas.

Para la realización del presente TFM, se han invertido un total de 300 horas. Todas ellas se han realizado dentro del horario normal, por lo que el número de horas especiales es cero. Además, de acuerdo con lo establecido por el COIT, el factor de corrección C_t que se aplica para 300 horas trabajadas es de 0,60, tal y como se puede comprobar en la Tabla P.1.

Horas	Factor de corrección
Hasta 36	1,00
Exceso de 36 hasta 72	0,90
Exceso de 72 hasta 108	0,80
Exceso de 108 hasta 144	0,70
Exceso de 144 hasta 180	0,65
Exceso de 180 hasta 360	0,60

Tabla P. 1: Coeficientes reductores para trabajo tarifado según el COIT.

Por lo tanto, en vista de la situación del TFM desarrollado, el coste total de honorarios se calcula como se muestra en la Ecuación P.2:

$$H = 0{,}6 \cdot 74{,}88 \cdot\ 300 +\ 0{,}6 \cdot 96{,}72 \cdot\ 0 = 13.478{,}40\ €\qquad \text{(P.2)}$$

El trabajo tarifado por tiempo empleado asciende a la cantidad de *trece mil cuatrocientos setenta y ocho euros con cuarenta céntimos*.

P.2 Amortización del inmovilizado material

Se entiende como inmovilizado material aquellos recursos *hardware* y *software* empleados para la realización de este TFM.

Para el cálculo del coste de amortización en un periodo de 3 años se utiliza un sistema de amortización lineal, en el que se supone que el inmovilizado material se desprecia de forma constante a lo largo de su vida útil. La cuota de amortización anual se calcula como se muestra en la Ecuación P.3.

$$Cuota\ anual = \frac{Valor\ de\ adquisición - Valor\ residual}{Número\ de\ años\ de\ vida\ útil} \qquad (P.3)$$

El sustraendo *Valor residual* se corresponde con el valor teórico que se supone que tendrá el elemento en cuestión después de su vida útil.

P.2.1 Amortización del material *hardware*

La duración de este Trabajo Fin de Máster es de 4 meses, es decir, un periodo muy inferior al de 3 años que se estipula para el coste de amortización. Por esta razón se calculan los costes sobre la base de los derivados de los primeros 4 meses.

En la Tabla P.2 se recogen los elementos *hardware* amortizables necesarios para la realización del trabajo, indicando su valor de adquisición y su amortización, teniendo en cuenta un tiempo de uso de 4 meses.

Equipo/Dispositivo	Valor de adquisición	Amortización
Ordenador Asus X54H	439,00 €	48,79 €
LoPy (x3)	114,93 €	114,93 €
Sensor ultrasónico	0,54 €	0,54 €
Componentes electrónicos y lámparas	54,84 €	54,84 €
Batería LiPo	8,10 €	8,10 €
Expansion Board	16,00 €	16,00 €

Bidirectional Logic Level Converter	2,67 €	2,67 €
Total	636,08 €	**245,87 €**

Tabla P. 2: Amortización del material hardware.

Debido al bajo precio de todos los elementos, exceptuando el ordenador portátil, la amortización coincide con su valor de adquisición en todos ellos. La amortización del ordenador portátil se ha calculado usando el sistema de amortización lineal explicado anteriormente.

El coste total del material *hardware* asciende a *doscientos cuarenta y cinco euros con ochenta y siete céntimos*.

P.2.2 Amortización del material *software*

Para realizar el cálculo de los costes de amortización del material *software* se consideran, al igual que con el material *hardware*, los costes derivados de los primeros 4 meses.

La Tabla P.3 muestra los elementos *software* necesarios para la realización del trabajo, así como su valor de adquisición y su amortización.

Aplicación	Valor de adquisición	Amortización
Sistema operativo Windows 10 Home	0,00 € (*)	0,00 € (*)
Microsoft Office	0,00 € (*)	0,00 € (*)
Microsoft Visio	0,00 € (*)	0,00 € (*)
Microsoft Project	0,00 € (*)	0,00 € (*)
Atom	0,00 € (**)	0,00 € (**)
Plataforma TTN	0,00 € (**)	0,00 € (**)
Pymakr	0,00 € (**)	0,00 € (**)
Mozilla Firefox	0,00 € (**)	0,00 € (**)

Adobe Reader	0,00 € (**)	0,00 € (**)
Total	0,00 €	**0,00 €**

Tabla P. 3: Amortización del software.

(*) Licencia de uso proporcionada por la ULPGC.

(**) Software de acceso libre.

El coste total del material *software* es de *cero* euros.

Si se suman los costes del inmovilizado material *hardware* y *software* se obtiene el coste total de inmovilizado material. Este cálculo se muestra en la Tabla P.4.

Concepto	Coste
Material *Hardware*	245,87 €
Material *Software*	0,00 €
Total	**245,87 €**

Tabla P. 4: Amortización del inmovilizado material.

Por lo que el coste total del inmovilizado material, tanto *hardware* como *software*, es de *doscientos cuarenta y cinco euros con ochenta y siete céntimos*.

P.3 Redacción del trabajo

Utilizando la Ecuación P.4 se determina el coste asociado a la redacción de la memoria del presente Trabajo Fin de Máster.

$$R = 0{,}07 \cdot P \cdot C_n \qquad \text{(P.4)}$$

Donde:

- R son los honorarios por la redacción del trabajo.
- P es el presupuesto.

- C_n es el coeficiente de ponderación en función del presupuesto.

Para obtener el valor del presupuesto se suman los costes del trabajo tarifado por tiempo empleado y la amortización del inmovilizado material. En la Tabla P.5 se muestra este cálculo de presupuesto.

Concepto	Coste
Trabajo tarifado por tiempo empleado	13.478,40 €
Amortización del inmovilizado material	245,87 €
Total	**13.724,27 €**

Tabla P. 5: Presupuesto, incluyendo trabajo tarifado y amortización del inmovilizado material.

Por otra parte, el coeficiente de ponderación C_n para presupuestos inferiores a 30.050,00€ tiene un valor de 1,00, según el COIT. Por lo que el coste derivado de la redacción del Trabajo Fin de Máster se calcula en la Ecuación P.5:

$$R = 0{,}07 \cdot 13.724{,}27 \cdot \ 1 = 960{,}70 \ € \qquad \text{(P.5)}$$

El coste de la redacción del trabajo asciende a *novecientos sesenta euros con setenta céntimos*.

P.4 Derechos de visado del COIT

El COIT establece que, para proyectos técnicos de carácter general, los derechos de visado para el año 2019 se calculan sobre la base de la Ecuación P.6.

$$V = 0{,}006 \cdot \ P_1 \cdot \ C_1 + 0{,}003 \cdot P_2 \cdot \ C_2 \qquad \text{(P.6)}$$

Donde:

- V es el coste de visado del trabajo.
- P_1 es el presupuesto del proyecto.

- C_1 es el coeficiente reductor en función del presupuesto.
- P_2 es el presupuesto de ejecución material correspondiente a la obra civil.
- C_2 es el coeficiente reductor en función del presupuesto de ejecución material correspondiente a la obra civil.

En la Tabla P.6 se muestra el presupuesto del proyecto, que se obtiene a partir de la suma de las secciones correspondientes al trabajo tarifado por tiempo empleado, a la amortización del inmovilizado material, y a la redacción del trabajo. Por otra parte, el coeficiente C_1 para proyectos de presupuesto inferior a 30.050,00€ es de 1,00, y el valor de P_2 es de 0,00€, ya que no se realiza ninguna obra civil. Por esta misma razón no se aplica el coeficiente C_2.

Concepto	Coste
Trabajo tarifado por tiempo empleado	13.478,40 €
Amortización del inmovilizado material	245,87 €
Redacción del trabajo	960,70 €
Total	**14.684,99 €**

Tabla P. 6: Presupuesto, incluyendo trabajo tarifado, amortización y redacción del trabajo.

De esta forma, el cálculo del coste por derechos de visado del presupuesto se realiza en la Ecuación P.7.

$$V = 0{,}006 \cdot\ 14.684{,}99 \cdot\ 1 + 0{,}003 \cdot 0 \cdot\ C_2 = 88{,}11\ € \qquad \text{(P.7)}$$

Por lo tanto, el coste por derechos de visado del presupuesto asciende a *ochenta y ocho euros con once céntimos*.

P.5 Gastos de tramitación y envío

Los gastos de tramitación y envío ascienden a *seis euros* (6,00€) por cada documento visado de forma telemática.

P.6 Material fungible

Durante el desarrollo de este Trabajo Fin de Máster se han empleado otros materiales a parte de los recursos *hardware* y *software* ya comentados. El material adicional se documenta como material fungible. En la Tabla P.7 se muestran los costes derivados de estos recursos.

Concepto	Coste
Folios	10,00 €
Tóner de la impresora	30,00 €
Encuadernado	5,00 €
Total	**45,00 €**

Tabla P. 7: Coste de material fungible.

El coste del material fungible asciende a *cuarenta y cinco euros*.

P.7 Aplicación de impuestos y coste total

La realización del presente TFM está gravada por el Impuesto General Indirecto Canario (IGIC) en un siete por ciento (7%). Teniendo en cuenta la aplicación de los impuestos se realiza el cálculo del presupuesto total del Trabajo Fin de Máster. Este cálculo se muestra en la Tabla P.8.

Concepto	Coste
Trabajo tarifado por tiempo empleado	13.478,40 €
Amortización del inmovilizado material	245,87 €
Redacción del trabajo	960,70 €
Derechos de visado del COIT	88,11 €
Gastos de tramitación y envío	6,00 €
Costes de material fungible	45,00 €
Total (Sin IGIC)	**14.824,08 €**
IGIC (7%)	1.037,69 €
Total	**15.861,77 €**

Tabla P. 8: Presupuesto total del Trabajo Fin de Máster.

El presupuesto total del Trabajo Fin de Máster "*Desarrollo de una plataforma HW/SW basada en las tecnologías VLC, BLE y LoRa/LoRaWAN para aplicaciones IoT*" asciende a *quince mil ochocientos sesenta y un euros con setenta y siete céntimos*.

Fdo.: D. Luis González Álvarez

En Las Palmas de Gran Canaria a 5 de marzo de 2020

Anexo

Anexo A. Contenido de la documentación del TFM.

En este anexo se describe la estructura de la documentación completa del TFM a partir de la lista que se expone a continuación.

- En el directorio `Memoria del TFM` se encuentra la memoria del TFM "*Desarrollo de una plataforma HW/SW basada en las tecnologías VLC, BLE y LoRa/LoRaWAN para aplicaciones IoT*" en lengua española y en formato PDF.

- En el directorio `Código` se encuentran cuatro subdirectorios:

  - En el subdirectorio `Nodo Final` se incluyen los archivos que contienen la codificación del elemento nodo final de la solución desarrollada. Estos son los archivos `ultrasonic.py` y `main.py`.

  - En el subdirectorio `Nodo Estación Base` se incluye el archivo que contiene la codificación del elemento nodo estación base de la solución desarrollada. Este es el archivo `main.py`.

  - En el subdirectorio `Nanogateway` se incluyen los archivos que contienen la codificación del elemento *nanogateway* de la solución desarrollada. Estos son los archivos `config.py`, `nanogateway.py` y `main.py`.

  - En el subdirectorio `TTN` se incluyen los archivos que contienen la codificación de los elementos *The Things Network* de la solución desarrollada. Estos son los archivos `Decoder.js`, `Encoder.js` y `JSONconfigureDownlink.txt`.

- En el directorio `Vídeo` se encuentran dos vídeos donde se demuestra el funcionamiento de la solución desarrollada.

- En el vídeo titulado `Uplink.mp4` se muestra la prueba de validación del caso de uso de *Smart Parking* para el sentido de comunicación ascendente desde el nodo final hasta TTN.

- En el vídeo titulado `Downlink.mp4` se muestra la prueba de validación del caso de uso de *Smart Parking* para el sentido de comunicación descendente hacia el nodo final.

- En el directorio `Abstract` se encuentran dos documentos:

  - En el documento `Abstract.pdf` se incluye el resumen del TFM en lengua inglesa y en formato PDF.

  - En el documento `Resumen.pdf` se incluye el resumen del TFM en lengua española y en formato PDF.

- En el directorio `Poster` se encuentra el póster ilustrativo del TFM, en formato PDF.