Desarrollo software enfocado a la optimización en consumo de un sistema autónomo pasarela BLE - LoRa

MEMORIA TRABAJO DE FIN DE GRADO

Iker Urío Echeverría

Grado en Ingeniería en Tecnologías de Telecomunicación
Universidad Pública de Navarra

Director: Israel Arnedo Gil

Junio 2017

Resumen

Internet está creciendo a pasos agigantados. No solo se están comunicando las personas entre sí, sino también los objetos. Esto es lo que se denomina *Internet of Things, IoT*.

Dentro del gigantesco mundo que es *IoT* se quiere crear un sistema donde se tengan una serie de objetos localizados en lugares donde no se dispone de conexión a Internet. Estos objetos tendrán acoplada una baliza que emitirá información sobre el objeto y su contexto a través de la tecnología *Bluetooth Low Energy, BLE*.

El objetivo de este proyecto es dotar de cobertura global a las balizas. Para ello, se va a desarrollar un programa software que haga que un módulo, el *RM186*, consiga capturar la información de las balizas en tecnología *BLE* y retransmitirla a través de la tecnología *LoRa*. Debe ser eficiente en términos de energía, ya que el sistema debe de ser autónomo. Al ofrecer la tecnología *BLE* un alcance de pocas decenas de metros, se realiza una conversión a la tecnología *LoRa* (alcances de hasta 10 km), en las zonas sin conexión a Internet. Esto permite llegar a puntos que sí la tienen y poder ofrecer así la información de manera global. Ambas tecnologías son de muy bajo consumo y están orientadas a sistemas alimentados con baterías.

La retransmisión se hará a un *servidor LoRa*, el cual tendría conexión a Internet y podría retransmitir los paquetes al *servidor final*.

El desarrollo del proyecto se realiza en colaboración con la empresa *Engineea*.

Palabras clave: *BLE, LoRa, pasarela, RM186, IoT*.

Abstract

The growth of Internet has been huge in recent years. Not only are people communicating to each other, also things are starting to communicate too. This is what is defined as *Internet of Things, IOT*.

Within the giant world that is *IoT*, it is desired to create a system that allows to have objects located in places with no Internet connection. These objects have coupled a beacon that will emit the information regarding the object and its context through the technology *Bluetooth Low Energy (BLE)*.

The aim of this proyect is to provide global coverage to the beacons. To achieve this, a software program will be developed to make possible for a *module RM186* to capture the information of the beacons in *BLE* technology and retransmit it through the *LoRa* technology. It is required for the design to be efficient in terms of energy in order to have autonomy. Regarding connectivity, *BLE* technology offers a range of few tens of meters and the conversion to *LoRa* (maximum range 10 km) will achieve to provide global connectivity in areas without Internet connection. The consumption of both technologies is very low and they are oriented to systems powered by batteries.

The system would transmit to a *LoRa server*, which would have Internet connection and could retransmit the packets to the *final server*.

The development of the project is carried out in collaboration with the company *Engineea*.

Keywords: *BLE, LoRa, gateway, RM186, IoT*.

Laburpena

Internet asko hazten hari da. Ez dira pertsonak bakarrik komunikatzen hari, objektuak ere komunikatzen hasi dira beraien artean. Hau *Internet of Things (IoT)* deitzen da.

IoT mundu handi onetan Internet konexiorik gabeko guneetan objektuak kokatzen dituen sistema bat sortu nahi da. Objektu hauek baliza bat edukiko dute eta honek objektuaren eta bere ingurunearen informazioa bidaliko du *Bluetooth Low Energy (BLE)* teknologiaren bidez.

Projectuaren helburua balizei estaldura globala ematea da. Hau lortzeko, software programa bat garatuko da *modulu RM186*a balizak bidalitako informazioa harrapatzeko *BLE* teknologiaren bidez eta hau *LoRa* teknologiaren bidez transmitituko du informazioa berriz. Energiari dagokionez eraginkorra izan behar da sistema autonomoa izan behar delako. *BLE* teknología hamarka metro gutxitan transmititu ahal du informazioa, baina *LoRa* teknologiara pasatzean, 10 km-ra iritzi ahal da (gehienez), horrela, konexio globala emateko Internet konexiorik gabeko guneetan. Bi teknologiek energia gutxi kontsumitzen dute eta energia bateriek ematen dituzten sistemetara bideratuta daude.

Transmisioa *LoRa zerbitzari* batera egingo da eta hau Internet konexioa edukiko duenez *zerbitzari finalera* bidali egingo ahal ditu paketeak.

Projectuaren garapena *Engineea* enpresaren lankidetzarekin egingo da.

Hitz gakoak: *BLE, LoRa, pasabidea, RM186, IoT.*

Agradecimientos

Por un lado quería dar las gracias a mi tutor **Israel Arnedo Gil** y a **Fernando Ruiz Tadeo** por darme los recursos necesarios para realizar el *Trabajo de Fin de Grado* correctamente y a la **Universidad Pública de Navarra** por haberme formado durante estos últimos años.

Por otro lado, quería darle especialmente las gracias a **Mikel Induráin Huarte** por haberme ayudado durante el desarrollo del *TFG*. Ha sabido guiarme correctamente en momentos donde he tenido dificultades para continuar y no ha dudado ni un instante en explicarme cualquier cosa que me pudiese servir de ayuda.

Iker Urío

Pamplona, España. Junio 2017.

Índice

Capítulo 1

Introducción

Resumen: Este capítulo presenta una breve introducción al proyecto. Se analiza el estado actual de *Internet of Things*, *IoT*, o *Internet de las Cosas* en español, y se plantea un problema junto con su posible solución. Además se describen las tecnologías que van a ser usadas en el proyecto para tener en cuenta sus características en el desarrollo.

1.1. Estado del arte

Hoy en día todo el mundo usa Internet, ya sea para trabajar, para aprender algo o simplemente para pasar el rato. Cada vez las personas están más conectadas a Internet y no hay ningún problema en conseguir comunicarse con alguien que está a cientos o miles de kilómetros de manera muy económica. El caso es que esto no se va a quedar aquí. Actualmente no solo se están comunicando las personas sino también se están empezando a comunicar los objetos. Esto es lo que se denomina *Internet of Things*, *IoT*, o como se diría en español, *Internet de las Cosas*.

IoT hace que se pueda tener objetos independientes, autónomos, para que nos hagan la vida más fácil. Por ejemplo, se puede tener un frigorífico que cuando se le vaya acabando la comida pida él solo la compra al supermercado a través de Internet. Esto sólo es un ejemplo, pero las aplicaciones de *IoT* son infinitas: control de riego, medición de datos de consumo, rastreo de vehículos, etc. y todo esto a tiempo casi real.

Esto se está expandiendo cada vez a más lugares, pero aún falta mucho por cubrir. Concretando un poco, en el mundo comercial, en un almacén al que le llegan un montón de palés de mercancía al día, se necesita un cierto control para poder administrar todo bien. Esta administración es muy

importante ya que si se hace mal puede suponer perder grandes cantidades de dinero. Por lo tanto, el primer paso es saber bien qué se tiene y para eso, en un principio, debe haber personas encargadas en controlar la llegada y salida de los paquetes. El problema está en que puede haber fallos humanos al controlar ésto y por esta y otras razones se ha empezado a hacer uso de las nuevas tecnologías para sustituir el control humano y manual por uno autónomo y automático.

El hecho de que el control se autónomo y automático tiene aún más ventajas. La más importante es que reduce el coste a la empresa, ya que una vez obtenido el dispositivo *IoT* se ahorra el dinero que conlleva tener empleados controlando todo. Aún teniendo trabajadores se les facilitaría mucho su trabajo. Además teniendo la información actualizada casi a tiempo real, se pueden poner alarmas para poder actuar con rapidez en caso de ocurrir cualquier error o avería. También el hecho de registrar y des-registrar los paquetes de esta manera conlleva mayor rapidez y esto en una empresa se traduce a mayor beneficio.

Existen varios proyectos en marcha con el objetivo de que esto se consiga y funcione correctamente y uno de ellos lo está desarrollando la empresa *Engineea* en colaboración con la *Universidad Pública de Navarra, (UPNA)*. Lo que se está consiguiendo hacer es que mediante balizas insertadas en los palés que lleven los paquetes, se transfiera información del contenido y del entorno del palé a un teléfono móvil mediante *Bluetooth Low Energy*, *BLE*. Estos teléfonos móviles a su vez mandan ésta información a un servidor principal el cual hará la gestión de toda la información y se tendrá un control total. Estos móviles serán de empleados de la empresa y ellos no tendrán que hacer nada para que este sistema de comunicación funcione. Será suficiente con tener una aplicación concreta instalada y al pasar el paquete a una distancia menor a unas pocas decenas de metros del trabajador, el móvil recibirá la información. Será recomendable que este trabajador sea el repartidor, ya que así se asegura la recepción de la información en el móvil de éste al transportar la mercancía de un lado a otro.

El problema es que hay vacíos en este sistema. Por ejemplo, si al trabajador se le ha acabado la batería o si está en un sitio donde no hay cobertura, la información no va a poder llegar al servidor. Por esta razón, se necesita de una solución alternativa para este tipo de circunstancias.

1.2. Objetivo del proyecto

Como se ha visto en el apartado anterior, hay un problema que solventar en el sistema de comunicación que ha estado desarrollando la empresa *En-*

gineea y la *Universidad Pública de Navarra*. El desarrollo del proyecto se va realizar con la colaboración de dicha empresa.

El proyecto va a tener como objetivo final conseguir que llegue la información que están proporcionando las balizas a través de *BLE* a un servidor mediante otra tecnología llamada *LoRa*, de manera que consuma la menor energía posible. Este servidor deberá ser capaz de transmitir toda la información al servidor central a través de Internet para así quedar unido al proyecto general realizado por la empresa *Engineea* y *UPNA*. Esta última parte será el siguiente paso a realizar por lo que no incumbe a este proyecto. No obstante, se va a tener en cuenta para mandar los paquetes de datos de manera que luego sea más fácil realizar el último paso.

La recepción de paquetes a través de la tecnología *BLE* y la transmisión de paquetes a través de *LoRa* se va a realizar en una placa con un módulo industrial. Este proyecto se va a centrar en el software del módulo no en el hardware. Más adelante, en otro proyecto, se intentará crear una placa con un módulo propio para que sea más económico. Debe ser eficiente en términos de energía, ya que el sistema debe de ser autónomo y si el software funciona en un módulo, también deberá funcionar en que se creará en un futuro.

Las tecnologías a usar se han escogido para facilitar lograr el objetivo final de que el módulo consiga interconectar las balizas con el servidor con la menor energía posible. Al ofrecer la tecnología *BLE* un alcance de pocas decenas de metros, se realiza una conversión a la tecnología *LoRa* (alcances de hasta 10 km), en las zonas sin cobertura Internet. Esto permite llegar a puntos que sí la tienen y poder ofrecer así la información de manera global. Ambas tecnologías son de muy bajo consumo y orientadas a sistemas alimentados con baterías.

A continuación, en la figura 1.1, se muestra el recorrido que va a seguir la información.

Como se ha comentado, este proyecto se centra en el módulo que hace de pasarela entre la tecnología *BLE* y *LoRa*. Los colores de los recuadros de la figura 1.1 ayudan a situarse mejor en que fase se va a profundizar más y en cual menos:

- **Azul**: Los dispositivos están programados y configurados antes de empezar el proyecto. Por tanto no hay que programar o configurar nada.
- **Verde**: Es lo que realmente interesa en el proyecto. Es lo que se va a tener que programar y configurar a mayor profundidad, la recepción

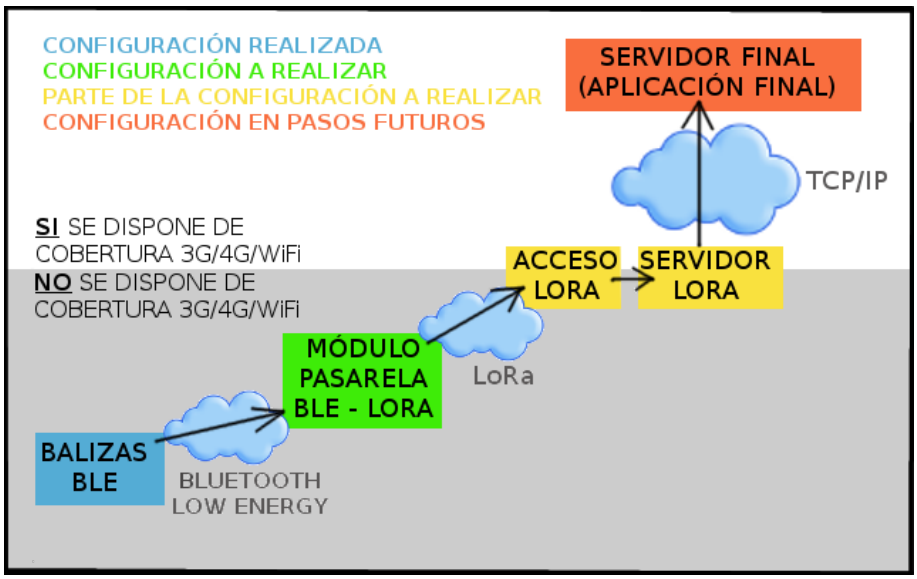


Figura 1.1: Esquema general del sistema

de paquetes vía *BLE* y la transmisión de paquetes vía *LoRa*. Se va a programar usando el lenguaje *SmartBASIC*.

- **Naranja**: Solo interesa la recepción de paquetes vía *LoRa*. Se va a programar y configurar sólo la recepción de paquetes que van vía LoRa, pero no la transmisión. Por tanto solo se va a tener que configurar lo necesario para poder recibir paquetes vía *LoRa*. Se va a programar y configurar usando *Node-RED* y *JavaScript*.
- **Rojo**: Se deja la programación y configuración para realizarla en un futuro. No incumbe a este proyecto.

Por otro lado, en la figura 1.1 también se puede ver por que tecnología se hacen las trasmisiones.

Para concluir, se distingue, por un lado la parte *gris* donde no se dispone cobertura 3G/4G/WiFi y por otro lado, se distingue la parte *blanca*, la cual sí que dispone de esa cobertura. El proyecto se ha realizado por motivo de esta la *gris*, ya que si no se usa la tecnología *LoRa* esas partes se quedarían incomunicadas con el servidor final. El objetivo del proyecto es cubrir la conectividad que falta en lugares para poder ofrecer cobertura global.

1.3. Elección de los componentes hardware

Como se ha mencionado anteriormente, el proyecto se va a centrar en el desarrollo del software del sistema autónomo y no en el hardware. Por

tanto, antes de empezar a desarrollar el programa se deben de elegir los **componentes hardware** adecuados para lograr el correcto funcionamiento del sistema.

1.3.1. Balizas BLE

Las balizas que emiten mensajes a través de *BLE* las ha desarrollado la empresa *Engineea* en colaboración con la *UPNA*. Se quiere que este proyecto se implemente al proyecto general, por lo que la elección de los emisores de mensajes *BLE* tiene que ser la misma. La baliza se puede ver en la figura 1.2.

Figura 1.2: Baliza BLE

Estas balizas emiten mensajes cada 4 segundos y han sido configuradas para enviar información sobre ellas y su entorno. Manda información como la presión atmosférica, la temperatura del chip, el estado de batería, etc.

1.3.2. Módulo pasarela BLE - LoRa

Para el siguiente paso, recibir mensajes *BLE* y mandar mensajes *LoRa*, se va a usar una tarjeta comercial (figura 1.3a). Ésta tendrá un un front-end de comunicaciones *BLE* para poder capturar los mensajes que manden los dispositivos *BLE* y un un front-end de comunicaciones *LoRa* con la que se mandará la información que resulte útil. Además dispone de un microchip *RM186* (figura 1.3b), donde se ejecutará el software que se cree en este proyecto.

Para configurar este módulo se usa el software *UwTerminalX*, el cual hace sencillas las tareas de configuración del módulo. Con éste software se podrá configurar los parámetros que necesita el módulo para conectarse al *LoRa*

(a) Módulo RM186 (b) Microchip RM183

Figura 1.3: Módulo RM186

gateway y además se podrá ejecutar el software que se cree en este proyecto. Después se podrá hacer que, solo con darle energía al módulo, se ejecute el programa creado.

El modulo puede ejecutar programas que hayan sido escritos en el lenguaje *SmartBasic* y ningún otro más. Las características de este lenguaje de programación se describirán en el *Capítulo 2*, comentado las ventajas y desventajas de su uso.

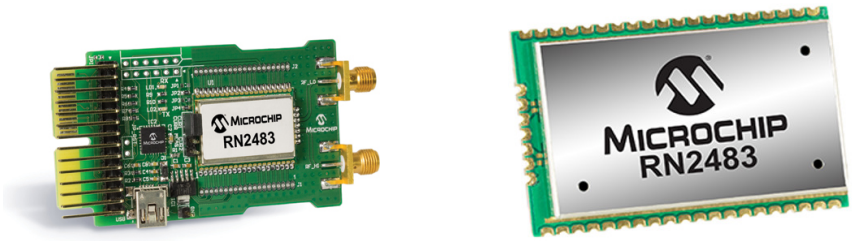
1.3.3. Gateway LoRa

El último componente hardware que se ha tenido que elegir es el *LoRa gateway*. Este es el hardware que más libertad se ha tenido al escogerlo. Se han barajado diversas opciones, pero aquí se van a explicar las tres más destacadas:

a) Módulo RN2483

Una de las opciones para hacer un *LoRa gateway* es hacer uno casero, utilizando el módulo *RN-2483-PICTAIL* (figura 1.4a) el cual tiene el microchip *RN2483* (figura 1.4b).

El fin de éste módulo no es usarse como *gateway*, sino como nodo *LoRa*. Los nodos *LoRa* son los que se tienen que conectar al *gateway* (el *módulo RM186* mencionado antes es un nodo). Sin embargo, se barajó la posibilidad de modificar este módulo para poder ejecutar en él un software para que se comporte como un *gateway*. Esto puede ser posible porque este dispone de un front-end de comunicaciones de 868 MHz, el cual se usa para transmitir



(a) Módulo RN-2483-PICTAIL (b) Microchip RN2483

Figura 1.4: Módulo RN2486

LoRa y por tanto podría servir para recibir.

Después de ver la complejidad de hacerlo de esta manera, se ha visto que es demasiado complicado y se ha descartado esta opción.

b) Raspberry Pi y módulo iC880

Otra de las opciones es crear el *gateway* de forma casera, pero de otra manera. Esta vez se va a usar una *Raspberry Pi* e instalar en ella el software necesario. Para poder recibir señal *LoRa* se hace uso de otro módulo, el *iC880*, ya que éste dispone de una front-end de comunicaciones a 868 MHz, la frecuencia a la que se transmite *LoRa* en Europa. Para no estar soldando al unir los dos módulos, se puede hacer uso de una placa que hace de interfaz entre ellas, la cual es muy útil y muy económica. En la figura 1.5 se puede ver las dos partes unidas con la placa interfaz.

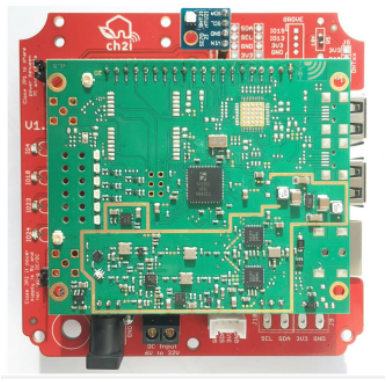


Figura 1.5: Raspberry Pi con módulo iC880

Se ha cogido una *Raspberry Pi* y se ha conseguido instalar el software

necesario, pero al final se ha decidido que esta no era la mejor opción para hacerlo. El *gateway* no solo tiene que conseguir captar señales, sino que luego tiene que mandarlas a un servidor *LoRa* para que éste desencripte y analice la información recibida. Se ve que se pueden mandar los datos a un servidor de *The Things Network*, pero hay soluciones mejores.

The Things Network es una comunidad bastante activa en la que la gente registra sus propios *LoRa gateways* y otros pueden usarlos de manera gratuita. Los *gateways* se configuran para que manden la información recibida a sus servidores. Si se tiene la suerte de que se dispone de un *gateway* alrededor que pueda ser utilizado, la elección de éste dispositivo hardware se puede saltar al tenerlo ya disponible y se ahorraría dinero. Sin embargo, no se dispone de uno, ya que actualmente en España no hay muchos y menos en Navarra. Por lo tanto crear uno de la forma descrita y registrarlo a ésta comunidad es buena opción, pero al final se descarta por otra opción más óptima.

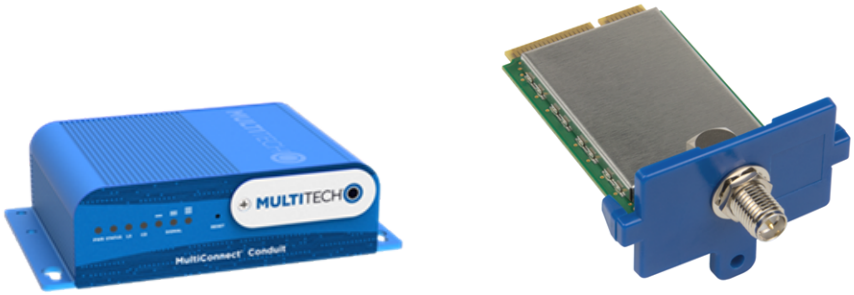
c) MultiConnect Conduit

La última opción que se ha pensado es comprar un *gateway* comercial. En un principio esta solución se descartó ya que no eran muy económicos, pero tras estudiar las ventajas que tendría tener uno de éstos, se ha decidido comprar uno. Se ha optado por comprar uno de la marca *Multitech*.

Esta marca no solo vende el *gateway*, sino que vende un servidor. Se ha escogido el *MultiConnect Conduit* (figura 1.6a) y para que éste actúe como *LoRa gateway*, también se ha comprado un módulo que se inserta en la parte trasera de éste. Lo que se tiene que insertar es el *Multitech Conduit LoRa gateway* (figura 1.6b), que es lo que realmente es el *gateway*. Lo demás es el servidor. Concretamente se ha comprado la versión *AEP* con software *Node-RED*, para poder configurarlo de manera más sencilla.

Se h a visto que es la manera más sencilla de tener un *gateway* y además tener uno propio y no uno compartido, siempre es más fiable y seguro. Además se puede trasladar a distintos lugares para hacer distintas pruebas. Por otro lado, tener un servidor *LoRa* propio, no el que proporciona *The Things Network*, da aún mayor flexibilidad a la hora de realizar pruebas.

Para concluir, decir que tener directamente un *gateway y servidor* fiable es una gran ventaja, ya que permite centrarse más en el *módulo RM186* el cual es el que realimente interesa que funcione de manera eficiente en este proyecto.



(a) MultiConnect Conduit (b) Multitech Conduit LoRa Gateway

Figura 1.6: MultiConnect Conduit

1.4. Descripción de las tecnologías usadas

Para el correcto desarrollo del proyecto se van a hacer uso de diversas tecnologías. En este apartado se van a describir brevemente, haciendo más hincapié en lo que realmente va a ser útil y necesario en el desarrollo.

1.4.1. Bluetooth Low Energy (BLE)

Bluetooth Low Energy, *BLE*, es una tecnología inalámbrica desarrollada por *Bluetooth* con objetivo de comunicar entre sí dispositivos pequeños con poca energía. Es una funcionalidad añadida al estándar de *Buetooth* a partir de la versión 4.0.

Opera en la misma banda que *Bluetooth Classic* o *WiFi*, **2.4 GHz**. Está basada en un *microchip* de bajo consumo del mismo tamaño que los dispositivos *Bluetooth*. Por otro lado, no está pensado para el *streaming* de datos y su alcance real es de unos pocos metros. El objetivo de *BLE* es conseguir objetos conectados que puedan tener una vida útil de varios meses e incluso años, alimentados con una pila de botón. Por esta razón, *Engineea* y la *UPNA* han hecho uso de esta tecnología para poder transmitir mensajes con bajo consumo.

En este proyecto va a interesar saber cómo recibir los paquetes de datos que llegan por *BLE*. La transmisiones las van a realizar las balizas, por lo que no se va a tener que tener que configurar ninguna transmisión. Una vez que se consiga recibir y procesar la información se tendrá que estudiar las diversas características de recepción que tiene *BLE* para su óptimo funcionamiento. Habrá que decidir si se quiere obtener más información a cambio de consumir más energía o viceversa, intentando llegar a un punto óptimo medio entre la eficiencia y el bajo consumo energético.

Las características a configurar, entre otras, son el porcentaje de tiempo que queremos que esté escuchando mensajes *BLE* nuestro dispositivo, el tamaño de la memoria caché de recepción o el tipo de recepción, pasiva o activa.

El envío de los mensajes se hacen en modo *broadcast*, por tanto, la información emitida podrá ser captada por más de un dispositivo. Estas transmisiones se denominan eventos de ***Advertising***. De esta manera se permite enviar datos a los dispositivos que estarán en un estado *Scanning*, sin necesidad de establecer ninguna conexión.

De estos *Advertisings* se va a obtener información que resulte útil. Para ello se va a describir el *Protocolo BLE* para saber cómo obtener el contenido útil de él y después se va a analizar un protocolo desarrollado por la empresa *Engineea* y la *UPNA* que está encapsulado en el protocolo anterior.

Protocolo BLE

El *Protocolo BLE* es muy amplio, pero aquí solo se va a explicar lo que interesa para la recepción y análisis de los mensajes emitidos por esta tecnología. La **estructura** de un **paquete BLE** se puede ver en la figura 1.7

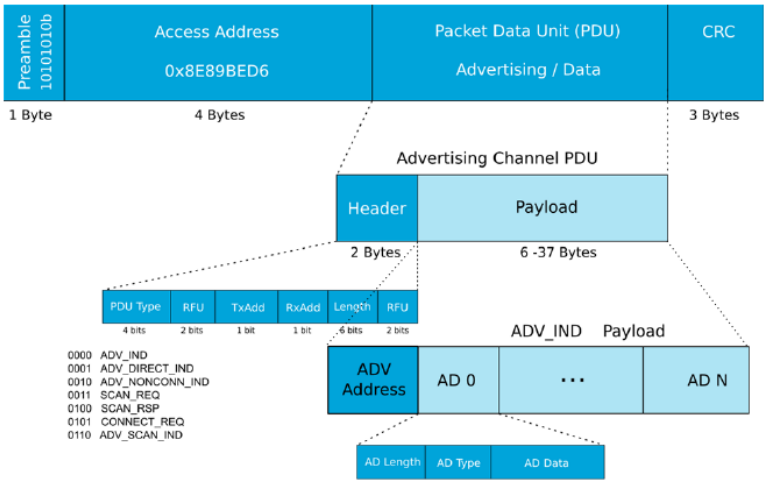


Figura 1.7: Estructura de un paquete *BLE* [3]

Como se ha mencionado anteriormente, los mensajes van a ser de tipo *Advertising* ya que se quiere que otros dispositivos escaneen y analicen los datos mandados sin tener que conectarse a ellos.

Sobre todo interesa obtener dos cosas del *Protocolo BLE*. Por un lado,

conocer la **dirección MAC** es de vital importancia, ya que así se puede reconocer de que dispositivo o baliza viene el mensaje. Por otro lado hay que saber donde se encuentra el *Protocolo Engineea* (se llama así en este proyecto, pero ha sido desarrollado por la empresa *Enginea* y la *Universidad Pública de Navarra*), ya que aquí va a estar la información útil.

Toda esta información se obtiene en el campo *Paquet Data Unit*, donde hay una cabecera y a continuación están el resto de datos. De la cabecera se puede obtener la dirección MAC de la baliza y en el resto de datos hay diversos tipos de *Advertisings* uno detrás de otro.

El protocolo define 3 campos por cada *Advertising*: la **longitud**, el **tipo** de *Advertising* y los **datos** del éste. Una vez vez definida la longitud del *Advertising*, se dice de que tipo es mediante un **flag**. Estos *flags* pueden decir que el **AD** contiene información sobre el **nombre local**, la *potencia de transmisión*, la *clase de dispositivo*, etc., pero en este proyecto solo interesan los tipos **0x01** y **0xFF**.

El primero de ellos indica que hay varios *flags* y tendrá longitud de 2 bytes. El segundo marca que es de tipo *Manufacturer Specific Data*, el cual indica que se puede meter aquí datos de un *Advertising* propio. Por esta razón la empresa *Engineea* ha metido aquí su propio protocolo. La estructura del *paquete BLE* que se acaba de mencionar se puede observar en la figura 1.8.

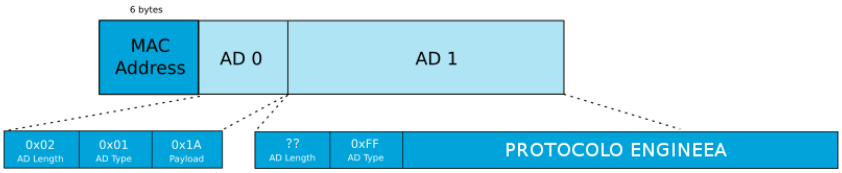


Figura 1.8: Estructura de un paquete *BLE* [3]

La longitud disponible de la trama es de **37 bytes**. Como se ha mencionado antes el primer *Advertising* usa 2 bytes de datos y el segundo en cambio 25 bytes de *payload* para meter la información que se quiera en el paquete, que en este caso será el **Protocolo Engineea** el cual se va a describir a continuación. El resto de información del paquete no va a resultar útil.

Protocolo Engineea

Este protocolo ha sido desarrollado por la empresa *Engineea* y *Universidad Pública de Navarra*. Por motivos de propiedad industrial solo se va a

mencionar algún aspecto del protocolo.

Como se ha mencionado anteriormente, este protocolo está encapsulado dentro del *Protocolo BLE*. Éste **define la comunicación entre el *beacon BT* o baliza y el *LoRa gateway***, en el que se querrá poder interpretar los datos recibidos. Además, en el módulo que va a convertir el *mensaje BLE* en uno *LoRa*, el *RM186* o el hardware que se haga en sustitución de éste, se va a introducir como complemento la posibilidad de sacar información de estos paquetes, por lo que aquí también habrá que aplicar este protocolo. Las balizas mandan los mensajes en modo ***broadcast*** y el módulo receptor va a estar en modo **pasivo** leyendo datos.

La trama dispone los siguientes datos que dan información sobre el objeto y su contexto:

- **Manufacturer**: Fabricante.
- **Model**: Modelo.
- **Model Version**: Número de versión.
- **Batch**: Número de lote.
- **Number-In-Batch**: Número asignado dentro del lote.
- **Capabilities and Status**: *Flags* que indican qué se puede hacer con el *Beacon BT*.
- **Estimated Battery Level**: Porcentaje de batería estimado.
- **Remaining Time**: Tiempo restante de batería.
- **PIRE**: Potencia Isotrópica de Radiación Equivalente.
- **Max Beacon Refresh Time**: Tiempo de refresco máximo.
- **Chip Temperature**: Temperatura que mide el termostato interno.
- **Atmosphere Pressure**: Presión atmosférica.
- **Seed**: Semilla aleatoria generada por el *Beacon BT* o baliza.
- **Hash**: Firma simple para reconocer que los paquetes han sido generados por un *Beacon BT*.

Cada uno de estos campos está metido en un grupo y según en cual esté, se obtiene la información de una manera u otra. Algún grupo tiene datos de longitud fija y algún otro variable. El proceso de obtención de esta información queda reservada a la empresa *Engineea* y a la *UPNA*.

1.4.2. LoRa/LoRaWAN

LoRa es una red de comunicaciones basada en el protocolo *LoRaWAN*, *Long Range Wide Area Network*. Como su nombre indica es para comunicaciones de **largo alcance**.

Ésta compite con otra tecnología llamada *Sigfox*, la que al igual que *LoRa*, es una red de baja tasa de transferencia, por lo que son ideales para un sistema de comunicación que se quiera que sus dispositivos consuman poca energía. La diferencia que tiene *LoRa* con *Sigfox* es que *LoRa* es ***Open Source***, esto es, cualquier empresa que quiera desplegar su propia red *LoRa* puede hacerlo libremente.

Ésta tecnología en Europa usa la **banda de los 868 MHz** creando una comunicación bidireccional. Está diseñada para garantizar las comunicaciones en cualquier situación, ya sea a mucha distancia, en interiores o en ubicaciones subterráneas.

Existen redes *LoRa* que están abierta al usuario *IoT*, pero solo están disponibles en algunos países como en Francia, Bélgica, Suiza, Países Bajos y Sudáfrica. **En España actualmente no hay una red comercial** por lo que se va a tener que montar una red propia. Es cierto que existe una comunidad llamada *The Things Network* en la que la gente comparte sus redes para que otros puedan usarlas y así conseguir hacer una red lo más grande posible, pero no está muy desplegada en España.

La configuración de esta tecnología debe hacerse en más de un sitio. Para empezar, hay que saber cómo manejar los mensajes *BLE* en el módulo *RM186* para poder luego mandarlos a través de *LoRa*. Luego se tiene que saber cómo transmitirlos y cómo recibirlos en el *gateway* elegido. Por último, al igual que se hace con *BLE* hay que ir cambiando parámetros de configuración para que el módulo consuma poca energía y por tanto sea lo más autónomo posible. La configuración de los parámetros en la parte de *LoRa* es más complicada al disponer de más cosas a configurar. Como es de esperar, el hecho de que consuma menos energía conlleva a que se está procesando o mandando menos cosas, por lo que habrá una mayor pérdida de paquetes. Habrá qué valorar que interesa más y el qué se puede despreciar para **conseguir un equilibrio entre funcionalidad y bajo consumo**. Éstos parámetros se explicarán más adelante y se elegirá la configuración óptima.

Para entender mejor esta tecnología se va a explicar el *Protocolo LoraWAN*.

Protocolo LoRaWAN

Por un lado se tiene *LoRa* que su protocolo especifica el modo de transmisión en **capa 2**. Aquí se explica cómo se mandan los mensajes físicamente. Por otro lado tenemos *LoRaWAN*, en el que además de lo anterior incluye el protocolo en la **capa de transporte**. Por tanto se va a explicar brevemente esta última para tener una idea general del funcionamiento de esta tecnología.

LoRaWAN, Long Range Wide Area Network es una especificación de *Low Power Wide Area Network, LPWAN*, la cual esta diseñada para dispositivos de bajo consumo, por lo que es útil para lograr que el módulo consuma poca energía. En la figura 1.9 se ven las **capas** que tiene esta tecnología y se ve que en **Europa** se transmite en la **frecuencia de 868 MHz** (también podría transmitir a 433 MHz, pero en este proyecto no se va a usar esta banda).

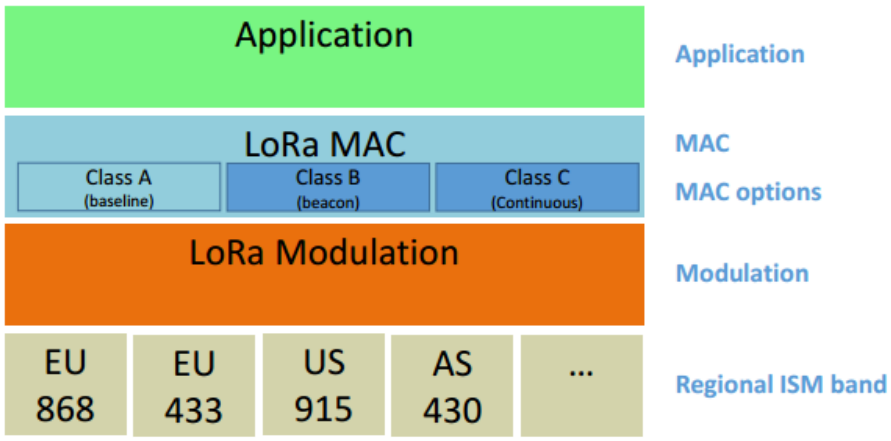


Figura 1.9: Capas LoRa/LoRaWAN [9]

También se distinguen varias clases:

- **Clase A**: Tiene las características básicas de LoRaWAN. Permite una comunicación bidireccional, con dos pequeñas ventanas de recepción. La transmisión no se puede realizar mientras alguna de las dos ventanas de recepción estén escuchando.

- **Clase B**: Es un complemento a la *Clase A*. Se añade una nueva ventana de recepción con la que poder escuchar, en un intervalo que se puede configurar. Sirve para poder recibir *mensajes LoRa* aunque no esté una de las dos ventanas de recepción de la *Clase A* escuchando.

- **Clase C**: Al igual que la *Clase B*, es un complemento a la *Clase A*. Es parecida al anterior, con la diferencia que la ventana que añade está siempre escuchando.

De las *3 clases*, la que menos pérdidas tiene es la *Clase C*, pero es la que más energía va a gastar al estar todo el rato con una ventana de recepción escuchando. La *Clase A* en cambio, al no tener ninguna ventana de recepción extra, es la que menos consume. A la hora de elegir la *clase* en la que se quiere trabajar, hay que valorar que es lo que interesa más, la baja pérdida de paquetes o bajo consumo de energía.

Por otro lado se dispone **dos modos** para interconectar los *nodos* con el *gateway* y el *servidor*:

- **Over-The-Air Activation, *OTAA***: En este modo solo hay que especificar las claves o llaves de aplicación y su ID, *Application Key (AppKey)* y *Application EUI (AppEUI)*, que hacen el papel de *SSL* y *DHCP* respectivamente. Esto es, proporciona una conexión segura y se identifica el dispositivo. Por otro lado tenemos las claves *Network Session Key (NwkSKey)* y *Application Session Key (AppSKey))*, las cuales no se tienen que especificar cuales son, ya que se crean en el proceso de unión. En este modo el nodo final se tiene que **unir** al *servidor LoRa* y así se crea una sesión.

  Al hacer la unión entre el nodo y el servidor, el nodo le manda un mensaje *join-request* al servidor en el que se incluyen la *AppEUI* y la *AppKey* más el *device EUI (DevEUI)* que identifica al dispositivo (este valor se genera aleatoriamente, aunque si se desea, se puede añadir un segundo *DevEUI* y así añadirle otro identificador con el valor que se desee).

  El servidor chequea los valores recibidos y luego recalcula el *Message Integrity Code (MIC)* con el *AppKey*. El *MIC* sirve para verificar la integridad del mensaje, esto es, para asegurarse que el mensaje no ha sido corrompido por alguien. Si el valor *MIC* recalculado es correcto, el servidor responde al nodo con un *join-acept* dentro de la ventana de recepción del nodo. A continuación el servidor crea otra valor propio llamado *AppNonce* y calcula las claves *NwkSKey* y *AppSKey* (por esta razón no hace falta crearlas). Éstas se calculan en base a los valores mandados en el mensaje *join-request*.

  En el mensaje *join-accept* se incluyen el valor *AppNonce*, *DevAddr* diciendo que retardo va ha haber entre las ventanas de recepción en ese nodo y la lista de los canales que se pueden usar.

- **Activation By Personalization, *ABP***: En este modo se tiene que especificar todo lo mencionado antes, ya que aquí **no hay proceso de unión** y por tanto hay que configurar todo manualmente. Realizar la conexión entre el nodo y el servidor de este modo es **menos seguro** que la anterior, ya las claves *NwkSKey* y *AppSKey* no son calculadas por el propio servidor y éstas podrían ser robadas (ya sea porque ha entrado alguien en el ordenador en el que se están configurando las claves o porque se han dejado apuntadas en un papel y alguien lo ha cogido).

Por tanto, por seguridad y sencillez se va a usar el primer modo, ***OTTA***. Además de ser éste el recomendado por la mayoría de los desarrolladores.

Hay que decir que aunque sea algo más seguro, si se configuran correctamente las dos claves mencionadas (*NwkSKey* y *AppSKey*) la conexión es igualmente de segura. Esto ocurre ya que una vez que la conexión esta establecida de cualquiera de las dos maneras, gracias a la combinación de estas dos claves los mensajes que se envíen estarán encriptados y firmados para que no puedan manipulados por otros nodos, como por ejemplo haciendo un ataque *man in the middle* (ponerse en medio de una conexión entre el servidor y el nodo para obtener o manipular la información que se está transmitiendo). Al estar encriptados los mensajes no se pueden leer y tampoco se tiene autorización para leerlos al estar firmados.

1.4.3. Programando en Smart Basic en el módulo RM186

El software que se va a ejecutar en el módulo *RM186* se va hacer usando el lenguaje de programación *SmartBasic*.

Se va a analizar la posibilidad de mandar el paquete igual que se recibe o se va a añadir algo más. Para ello, en primer lugar, se va ha decidir cómo va a ser el proceso que va a realizar el programa de manera autónoma para realizar de manera eficiente la recepción y transmisión (poniendo *buffers, timers, timeouts,* etc.). En segundo lugar, se debe conseguir que el programa, además de que todo lo anterior se cumpla correctamente, tenga unas cuantas variables con las que poder cambiar las características del proceso de recepción y de transmisión de manera muy simple. Así, una vez hecho el programa, se podrá usar para hacer pruebas empíricas cambiando las diversas características de las tecnologías mencionadas anteriormente (ventana de recepción/transmisión, cada cuanto se mandan mensajes al servidor, etc.). Estas pruebas se realizarán en el *Capítulo 4*, donde se deberá elegir el valor de estas variables.

Para efectuar el desarrollo de lo descrito, este lenguaje tiene sus ventajas

y sus desventajas:

De las **ventajas** más importantes que tiene *SmartBasic* es que se dispone de **funciones pre-configuradas** con las que hacen que la realización de un evento sea muy fácil de programar. Por ejemplo, una vez entendidas las tecnologías *BLE y LoRaWAN*, es fácil de recibir y analizar mensajes por *BLE* y lo mismo pasa al transmitir por *LoRaWAN*. Al principio es complicado comprender cómo está pensado este lenguaje para ser programado, pero una vez que se comprende, se ve que no es un lenguaje complejo. Todo el lenguaje se basa en **eventos**. Se recibe un evento, el cual se ha configurado anteriormente para que se quede escuchando y una vez ocurrido éste, pasa a ejecutarse una función para realizar lo que se haya configurado.

Pero como se ha mencionado antes, el uso de este lenguaje también tiene sus **desventajas**. El lenguaje es bastante sencillo, lo que una vez entendida filosofía de programación que tiene no es complicado de programar, pero el hecho de que sea sencillo conlleva a que tiene **muchas limitaciones**. Está bien cuando se quiere hacer cosas que también las han pensado los desarrolladores del lenguaje, pero a la hora de inventar cosas es más complicado. El desarrollo del programa se debe adecuar al lenguaje y esto conlleva a crear más líneas de código que las que se necesitaría en otro lenguaje más completo como lo es *Java o C*. Además, no es un lenguaje conocido, es una modificación de *VisualBasic*, pero no es lo mismo. Por ello, **no hay mucha documentación** ni en libros ni en Internet y no se disponen de foros para preguntar dudas. Eso supone perder bastante tiempo en conseguir cosas que con otros lenguajes se harían de forma mucho más sencilla. Por ello se han hecho unas pocas funciones para que la programación sea más sencilla. Estas funciones están en el *Apéndice A*.

En **resumen**, el programa se realiza con este lenguaje porque el módulo *RM186* está pensado para compilar y ejecutar código escrito con este lenguaje aunque no sea el ideal, pero es cierto que en ciertos puntos concretos da bastantes facilidades.

Capítulo 2

Desarrollo de la estructura del sistema

Resumen: Este capítulo informa de cómo es la estructura de todo el sistema. Además se describe la configuración de las fases más importantes de manera independiente. Después se ve cómo se hace la unión entre esas fases. El objetivo de este capítulo es conseguir hacer que un mensaje recorra todo el sistema, sin profundizar en cómo hacer que esto sea óptimo en términos de energía.

2.1. Estructura del sistema

La estructura del proyecto se puede ver en la figura 2.1. Como se puede apreciar hay varias fases:

- La **primera** es mandar mensajes por *BLE* y captarlos con un módulo que pueda hacer la transformación de *BLE* a *LoRa*.
- La **segunda** es mandar toda la información captada a un *gateway* que pueda recibir *LoRa*.
- La **tercera** fase es mandar los datos a un servidor *LoRa* y desencriptar la información recibida.
- Para finalizar, la **cuarta** y última fase es mandar lo que se desee al servidor central, donde se tendrá una aplicación propia y ésta habrá que configurarla para que acepte paquetes enviados desde dicho servidor.

Se aprecia que cada fase se hace usando una tecnología diferente. En la primera se manda a través de la tecnología *BLE*. En la segunda a través de

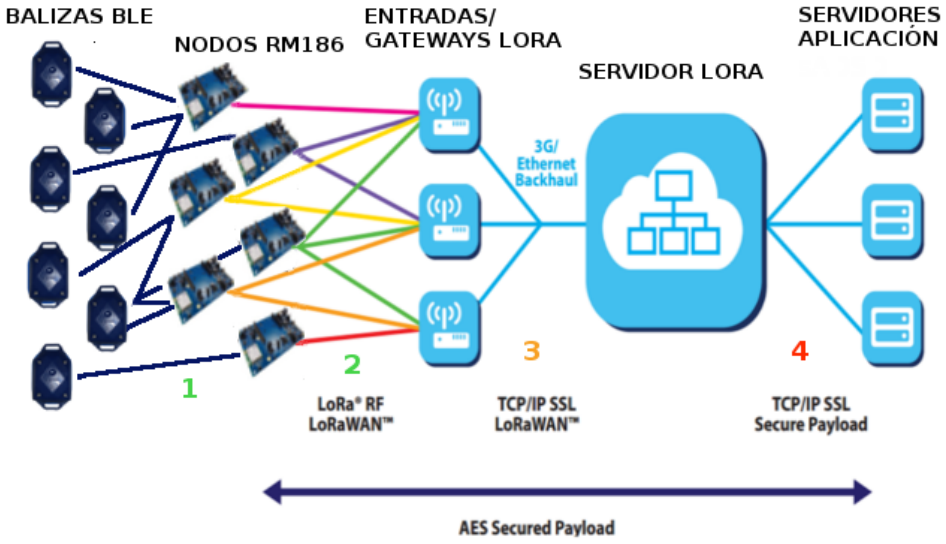


Figura 2.1: Estructura del sistema completo [10]

LoRaWAN. Por último, las últimas dos se mandan a través de *TCP/IP* de forma segura segura gracias a al encriptación por *SSL*.

Aunque se hayan descrito 4 etapas, en este proyecto se usa un *servidor LoRa* con un *LoRa gateway*, por lo que las **fases 2 y 3** se hacen en **una sola**.

El objetivo del proyecto trata de cómo hacer la *conversión BLE-LoRa* de la manera más eficiente posible, por lo que la *cuarta fase* no se realiza, ya que no incumbe al proyecto. No obstante, se ha tenido en mente ésta última, tratando de mandar los mensajes de forma que luego sea sencillo de realizar esta ultima fase y por tanto, unir este proyecto al proyecto general.

Los dispositivos que emiten mensajes **BLE** están mandando paquetes cada 4 segundos. Estos dispositivos estarán en los objetos en los que se quiera tener controlada su localización, como por ejemplo, en palés de mercancía. Los dispositivos que van a hacer la conversión *BLE-LoRa* estarán en grúas o sitios similares donde no se disponga de cobertura Internet. Por lo tanto, teniendo los dispositivos *BLE* en los palés y el módulo que pasa de una tecnología a otra en la grúa, cuando esta grúa coja un palé, captará los mensajes que se están enviando y se podrá transmitir información sobre los objetos y su entorno a un *gateway* a través de *LoRa*. Este *gateway* estará a unos cuantos kilómetros (unos 10 km en campo abierto y 2-3 km en zona

urbana) y éste si que estará conectado a Internet. Por lo tanto, éste podrá mandar la información al servidor general a través de *TCP/IP*, Internet.

Este último paso, la transmisión de datos al servidor final donde se procesarán, solo se va a pensar cómo hacerlo, no se va a realizar. El objetivo del proyecto no es conseguir llevar los paquetes hasta el servidor final, sino hacer la conversión *BLE-LoRa* lo más eficiente posible.

A continuación se van a describir por un lado la *fase 1* y por otro las *fases 2 y 3*. Se va a explicar como poder realizarlas independientemente, configurando los dispositivos correctamente. No se va a profundizar en la configuración óptima de éstas, eso se deja para el *Capítulo 3*, pero sí que se van a realizar pasarela entre las dos en un último punto.

2.2. Configuración básica del UwTerminalX

La primera cosa que se debe saber antes de empezar a programar es manejar el software **UwTerminal**. Con el se podrán cargar las diversos programas en modo prueba que se vayan haciendo, aunque por motivos de memoria también se deberán ir borrando.

Después de descargar el software de manera gratuita e instalarlo en un ordenador, se conecta el *módulo RM186* al ordenador mediante un puerto serie. El software esta disponible para distintos sistemas operativos, como **Linux**, **Windows** o **MAC**. La instalación en cada sistema operativo es similar, con alguna mínima diferencia. Además, para poder usarlo, en *Linux* se deben de dar permisos de escritura y lectura en el puerto donde se conecta el módulo.

Una vez abierto el programa se debe aceptar la licencia del software y se continua a configurar los siguientes parámetros para el correcto funcionamiento de éste:

- **Port**: Se introduce el puerto serie en el que hemos conectado el módulo (por ejemplo el *ttyUSB0*).
- **Baudrate**: Debe tener un valor de *115200*.
- **Parity**: *None*
- **Stop Bits**: *1*.
- **Data Bits**: *8*
- **Handshaking**: *CTS/RTS*.

En la casilla ***Device*** se puede poner un nombre a la configuración y pinchando en el botón ***Save device configuration*** se pude guardar la configuración establecida para que no se tenga que configurar siempre que entremos al programa.

Pinchando en el botón ***OK***, se abre el puerto serie y se pasa a la ventana donde se podrá configurar parámetros del módulo mediante comandos ***AT*** y en el cual se podrá **compilar**, **cargar** y **ejecutar** el programa.

Se va a realizar un programa simple, en el que pulsando un botón del módulo va a aparecer un mensaje diciendo que se ha presionado o soltado el botón. No se va a explicar como se ha realizado el programa porque no es relevante, solo se quiere explicar cómo compilar, cargar y ejecutar programas.

Para ello, una vez creado el programa se van a seguir los siguientes pasos:

- Se pincha en la pantalla negra con el **botón derecho** y se le da a ***XCompile + Load***. Con esto se compila el programa y se carga en el módulo.
- Se selecciona el programa con extensión **.sb**.
- Se puede comprobar que se ha cargado correctamente con el siguiente comando:

```
at+dir
```

- Una vez hecho esto, se puede ejecutar el programa con el comando

```
at+run "nombre_del_programa"
```

  sin la extensión *.sb*. También se puede ejecutar introduciendo simplemente el *nombre del programa*.

- En este instante se habrá ejecutado el programa y en él, lo primero que se ha puesto ha sido un mensaje diciendo que el programa ha empezado y que se presione el botón para ver los siguientes mensajes.
- Pulsando y soltando el botón del módulo aparecerán mensajes indicando ambas acciones. Esto se puede realizar todas las veces que se quiera.

En la figura 2.2 se puede ver todo el proceso que se acaba de describir. Decir que se ha pulsado y soltado el botón 3 veces a la hora de probarlo.

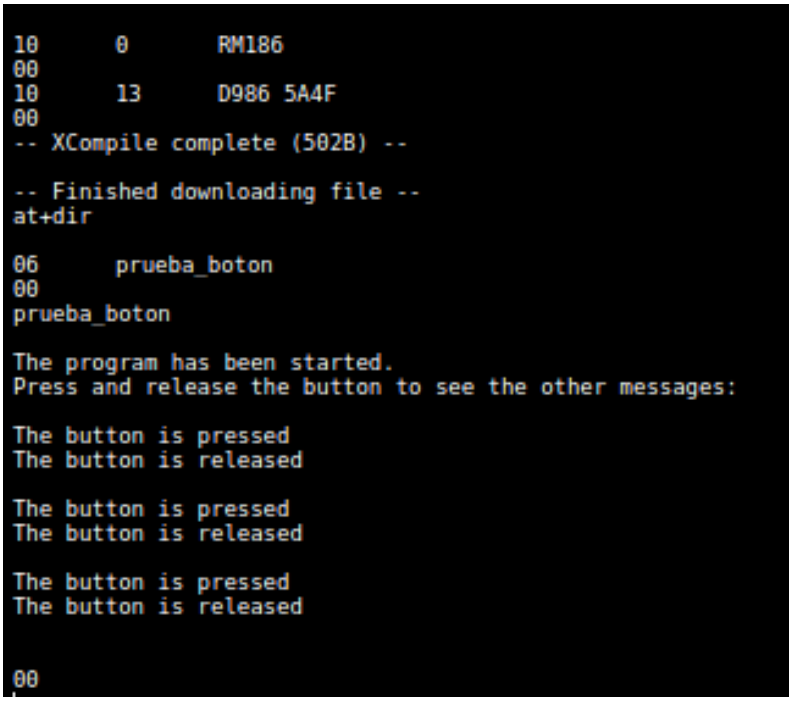


Figura 2.2: Ejecución de un programa sencillo en *UwTerminalX*

Para reiniciar el módulo se pulsa otro botón. Para verificar que se ha salido del programa correctamente se tiene que asegurar que aparece código *00* al pulsar la tecla *Enter* del ordenador, el cual indica que ya no estamos dentro del programa, sino introduciendo *comandos AT*. Este botón no se puede configurar, solo sirve para reiniciar módulo.

Estando fuera del programa se puede reiniciar el módulo pulsando ese botón o introduciendo el siguiente comando:

```
atz
```

Por otro lado, como se ha mencionado antes la memoria del módulo se llena. Para borrar esta memoria se debe introducir el siguiente comando:

```
at&f 1
```

Si además de la memoria se quiere borrar toda la configuración (del módulo, no de *UwTerminalX*), en vez del comando anterior hay que introducir el siguiente:

```
at&f 256
```

Durante los siguientes apartados se irán describiendo más *comandos AT*, según se vayan necesitando.

2.3. Escaneo y filtrado de paquetes BLE

Como se ve la figura 2.1, el **primer paso** del sistema que se va a intentar crear es la recepción de paquetes *BLE* en el dispositivo *RM186*. Las balizas que emiten *BLE* no mandan mensajes en modo *unicast*, esto es, a un solo dispositivo, sino que mandan en modo *broadcast*, a todos los dispositivos. Por esa razón solo se tiene que configurar el dispositivo receptor de forma que escanee estos paquetes. Con el lenguaje de programación *SmartBASIC* se ha podido realizar esto correctamente.

Los paquetes recibidos se almacenan en una memoria caché y de aquí se puede obtener información sobre ellos. Se dispone del **siguiente comando** con el que se obtiene información acerca de la **dirección MAC origen** (*periphAdd*), del paquete, los **datos** (*fullAD*), de este paquete, el número de **paquetes descartados** (*nDiscarded*), y la **potencia** (*nRssi*), con que han llegado los paquetes (*RSSI: Received Signal Strength Indication*):

```
looprc = BleScanGetAdvReport(periphAdd$, fullAD$,
    nDiscarded, nRssi)
```

Una vez obtenida esta información, el siguiente paso es sacar más información sobre los datos del paquete ya que en este momento solo se dispone de una secuencia de bytes. Indagando en ellos se ha conseguido **filtrar** los mensajes que provienen de dispositivos de *Engineea*.

Hay un comando que separa los datos por *Advertisings*:

```
rc = BleGetADbyIndex(0, fullAD$, nADTag, ADval$)
```

En él, se obtienen los *BLE Advertisings* de todo el mensaje. En el comando de arriba por ejemplo, se está sacando el primero al indicarle como primer parámetro un *0*. En los demás parámetros se obtiene la información que devuelve el comando:

- **fullAD**: Información sobre todos los *Advertisings*.

- **nTag**: Número de *TAG*, en hexadecimal, el cual indica el tipo de *Advertising* que es.
- **ADval**: El contenido del *Advertising*, la información útil.

Los mensajes de *Engineea* solo van a tener 2 *Advertisings*, por lo que para el **filtrado** se obtienen los dos primeros, en el caso de que los haya. El primero debe de ser de tipo 1 (*0x01* en hexadecimal) y el segundo de tipo 255 (*0xFF*) como se ha explicado en el *Capitulo 1*. Los paquetes que no cumplan estas características son **descartados**, como se ve en el siguiente trozo de código:

```
if element0Tag == 1 && element1Tag == 255 && strcmp(
    ADManufacturer$, "FFFF") == 0 then
// trozo de código que analiza los paquetes de Engineea
endif
```

Esto es, serán analizados los paquetes que tengan el primer *tag* a *0x01*, el segundo a *0xFF* y además, el primer byte del segundo *Advertising*, sea también *0xFF* (ésto es el fabricante y según el **Protocolo Engineea**, se ha decidió poner ese byte indicando que es un dispositivo de este proyecto (o del proyecto general)). Los demás paquetes serán **descartados**.

Una vez hecho esto se descompone el segundo *Advertising* en su totalidad, a nivel de bit, en el cual está el contenido que realmente interesa. Aquí también se analiza si el paquete es de *Engineea* analizando los bytes y en el caso de que no lo sea se **descarta**. Así se concluye el filtrado y **deja solo los paquetes provenientes de las balizas realizadas por *Engineea* y la *UPNA***.

Por otra parte, decir que se ha añadido una nueva característica al programa. Aunque no se tenga que analizar toda la información que nos proporciona el paquete, concretamente donde se encuentra el *Protocolo Engineea*, se ha desarrollado un algoritmo que obtiene toda la información que nos proporcionan las balizas *BLE*, aprovechando la descomposición del paquete anterior. Así, se da la seguridad al desarrollador que los paquetes recibidos están siendo tratados correctamente y no hay fallos en el código. Además si por cualquier motivo se quiere obtener información en el mismo *módulo RM186*, de esta manera se puede obtener sin tener que analizar esto en el *servidor LoRa* o *servidor final*.

A continuación se va a verificar el correcto funcionamiento de esta parte. Para ello, el programa se ha creado siguiendo lo descrito anteriormente y

solo dejando **pasar mensajes provenientes de balizas BLE que lleven el *Protocolo Engineea* encapsulado**, las demás se descartan. Para que la prueba sea más sencilla, se va a realizar en un lugar donde sólo se disponga de una sola baliza emitiendo, verificando así que pasa el filtrado y se muestra en pantalla.

Una vez creado el código hay que seguir los siguientes pasos:

- Se compila, carga y ejecuta el software en el *módulo RM186* como se ha explicado anteriormente.
- Se espera a que el módulo capte un *mensaje BLE*.
- Se verifica que la *dirección MAC* y los *datos* de la baliza se han impreso en la pantalla correctamente.

En la figura 2.3 se puede ver como se imprime por pantalla la **dirección MAC** y los **datos** del mensaje. Estos últimos se pintan de forma hexadecimal, pero esto es una forma para interpretarlos. En el posterior envío por *LoRa* se mandarán los *bytes*, sin importar como se han interpretado al escribirlos por pantalla. Esta es la forma en la que va a funcionar el programa en el ***modo normal***.

Por otro lado, en la figura 2.4 se ha ejecutado el programa que se ha realizado para verificar la correcta descomposición del paquete en el filtrado, para que el desarrollador este seguro de que lo ha hecho bien. Estos datos no son necesarios enviarlos, ya que solo es la interpretación de los *datos* pintados un poco antes, los cuales si que se van a enviar. Por lo tanto, esta parte del programa no se va a implementar en el *programa final*, pero es muy útil tenerla para realizar pruebas en ***modo debug***.

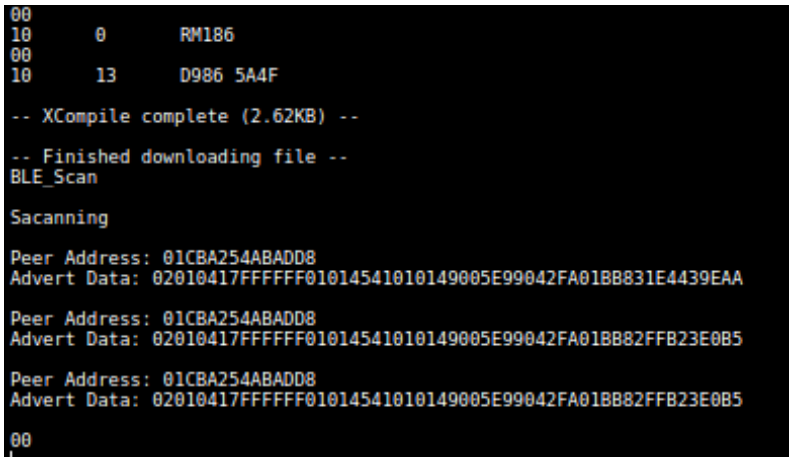


Figura 2.3: Captura de *mensajes BLE - modo normal*

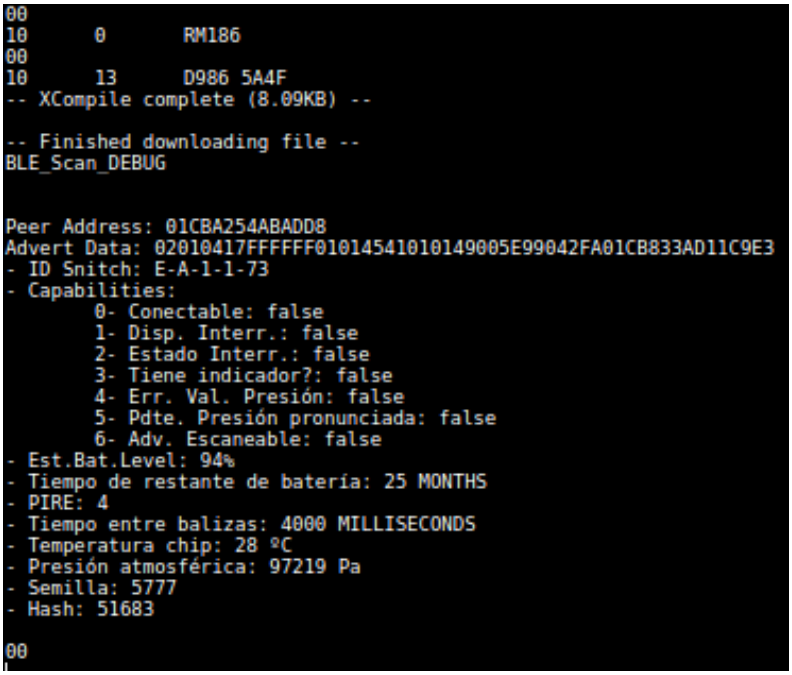


Figura 2.4: Captura de *mensajes BLE - modo debug*

En las dos pruebas se ha puesto a escanear *mensajes/paquetes BLE* de manera continua. En la primera se muestra por pantalla 3 mensajes que ha mandado la baliza (estas mandan mensajes cada 4 segundos) y en la segunda se muestra una sola captura.

2.4. Transmisión y recepción de paquetes vía Lora

El **segundo paso** y el **tercer paso** consisten en la transmisión de mensajes vía *LoRa*. Para ello en el otro extremo se tiene que tener un *LoRa gateway* que redirigirá los paquetes a un *servidor LoRa*. El *gateway* solo hace de pasarela al servidor, no realiza ninguna otra acción.

Lo primero que se debe hacer es elegir como se va realizar la conexión entre el nodo y el servidor. Como se describió en el *Capitulo 1* puede ser que el nodo se una a una sesión con el servidor, *configuración OTAA*, o que se tenga que configurar todos los parámetros manualmente para poder luego transmitir, *configuración ABP*. Como se dijo, se va a realizar la transmisión vía *OTTA* ya que así se deja en manos del servidor parte de la configuración y además de ser más sencillo, es más seguro.

Los parámetros que hay que configurar de esta forma son los siguientes,

tanto en el nodo como en el servidor:

- Se debe identificar la aplicación con algo parecida a un ID con ***Application EUI***.
- Para que la aplicación se transmita de forma segura, se debe especificar una llave, la ***Application Key***. Con esta se encriptarán y desencriptarán los mensajes en los dos extremos (en el nodo y el servidor, para que así nadie más consiga captar la señal y robar o modificar la información.

La configuración del nodo se realiza mediante *comandos AT*, siendo el primero para la *Application EUI* y el segundo para la *Application Key*:

```
at+cfgwx 1000 "0123456789ABCDEF"
at+cfgwx 1002 "0123456789ABCDEF0123456789ABCDEF"
```

En el servidor en cambio, se dispone de una interfaz gráfica. Para ello se conecta el ordenador a la misma red del servidor (mediante un *cable Ethernet*). La *dirección IP* por defecto es la *192.168.2.1* y se podrá acceder a ella mediante un explorador de Internet (siempre que se esté en la misma red, como se acaba de mencionar). También se puede conectar el servidor al router por donde se obtiene en el ordenador conexión Internet. Para ello se debe desactivar el protocolo DHCP en el servidor y asignarle una IP fija (para que el router no asigne una al alzar y luego no se pueda acceder al servidor porque no se sabe cual es la IP) dentro de la red que cree el router. Además se debe tener el *firewall* configurado correctamente para que habilite su acceso. En este paso es mejor conectar el servidor directamente al ordenador como se ha dicho al principio y una vez configurado todo correctamente, pasar a conectarlo al router.

Lo primero que aparece al entrar es un formulario para identificarse para poder entrar dentro. El **usuario** y **contraseña** por defecto es **admin** para **ambos** por defecto, pero se recomienda cambiarlos, por lo menos la contraseña, por motivos de seguridad.

Lo siguiente es configurar parámetros básicos del *servidor* y del *LoRa gateway* que se ha insertado en el anteriormente. Se dispone de una **guía rápida** y lo único que se tiene que hacer es seguirla. No se tarda más de uno o dos minutos en completarla.

Una vez hecho esto, el siguiente paso es configurar el *servidor LoRa* en mayor profundidad pinchando en la pestaña ***Setup*** y luego ***LoRa Network***

Server. Se introducen los parámetros mencionados antes como se puede ver en la figura 2.5.

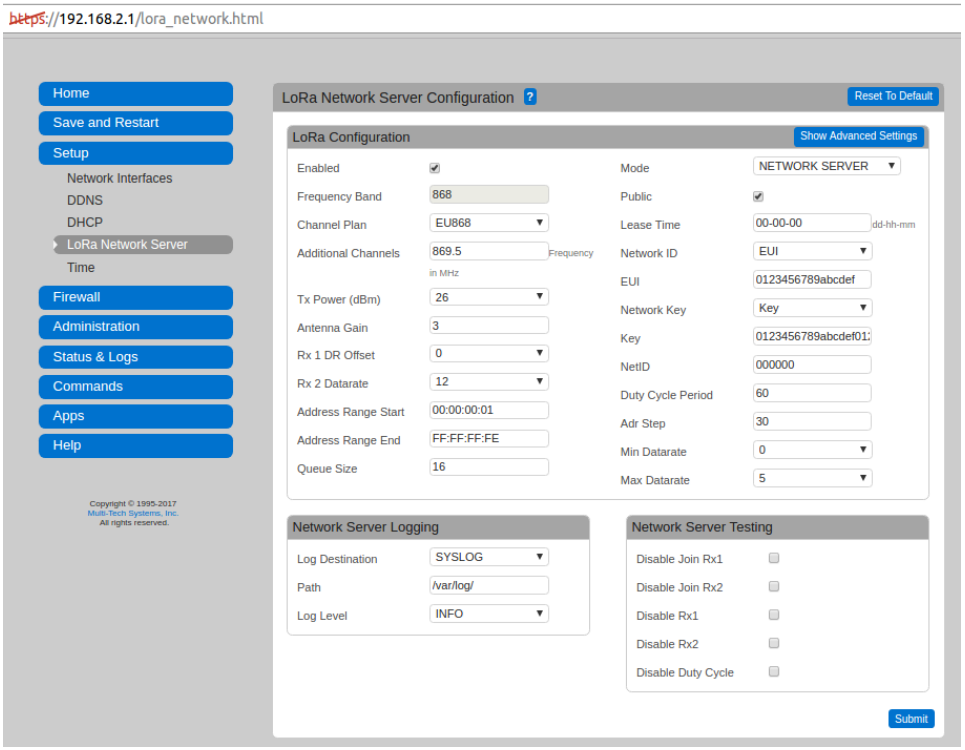


Figura 2.5: Configuración del servidor LoRa

Además de los parámetros mencionados, se puede ver en la figura 2.5 que también se hace público el servidor. Los parámetros restantes, de momento, se han dejado por defecto ya que así se conseguirá que el paquete recorra el camino deseado, cumpliendo así el objetivo de esta fase, pero es posible que haya parámetros que haya que cambiar para que el consumo de energía sea menor. En el *Capitulo 3* se analizarán alguno de estos parámetros haciendo alguna prueba y así tratar de acercarse al sistema autónomo óptimo deseado.

Como se menciono en el *Capitulo 1*, al hacer la transmisión vía *OTAA*, se deja al servidor encargado de crear las llaves de seguridad de la sesión en los niveles de red y de aplicación de la capa *OSI*, *Network Session Key* y *Application Session Key*, respectivamente.

Además de esto, se tiene que configurar una aplicación en el *servidor LoRa* para poder analizar y modificar los datos recibidos en él. Esta aplicación se llama ***Node-RED***.

Node-RED es una herramienta de programación que hace que **programar** sea más **visual, ya que utiliza un lenguaje de programación visual gráfico**. Está estructurada por bloques y solo con unirlos mediante un cable se pueden desarrollar muchas cosas. Hay varios tipos de bloques: de **entrada** de datos, de **salida** de datos, de **funciones**, etc., pero de momento solo interesan dos o tres. Un bloque para **recibir paquetes *LoRa***, otro para **analizar** lo que ha llegado por el bloque anterior y otro para poder **enviar** datos vía *LoRa* (en caso de que se quiera).

Esta aplicación está en el puerto *1880* del servidor, por lo que si se dispone de la *dirección IP* por defecto para entrar al servidor, se deberá introducir lo siguiente en el explorador: *192.168.2.1:1880*. Las credenciales para la identificación son las mismas que las que se dispone para entrar al *servidor LoRa*.

El resultado de la configuración de *Node-RED* se puede ver en la figura 2.6, en la cual se ha incluido el bloque de envío por *LoRa*, pero no se va ha hacer uso de él.

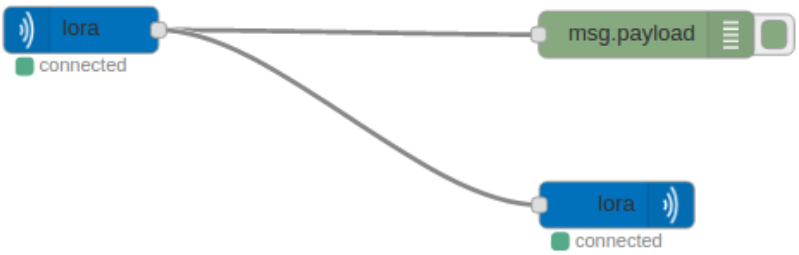


Figura 2.6: Configuración Node-RED

Una vez que todo está configurado correctamente, el siguiente paso a la hora de realizar la transmisión es unirse al *servidor LoRa* creando así una sesión. Este proceso se hace de la siguiente manera:

- Se hace la petición de unión al servidor vía *OTAA*.

```
#define LORAMAC_JOIN_BY_REQUEST  1 // Used with
    LORAMACJoin

// trozo de código del programa no relevante hasta
    llegar a la configuración de unión de LoRa
```

```
rc = LORAMACJoin(LORAMAC_JOIN_BY_REQUEST)
```

- Se espera a que el servidor le conteste al nodo.
- Si pasado un tiempo no le ha contestado el servidor, no se hace la unión y no se podrá transmitir nada.
- Si el servidor le contesta indicándole que la unión se ha realizado correctamente, se podrá transmitir datos vía *LoRa* al servidor.

Una vez creada la sesión es posible mandar mensajes desde el nodo al servidor. Para ello se va a intentar mandar un mensaje simple, *Hello World*, al servidor desde el nodo. Decir que al servidor llegan bytes y para poder analizar lo que llega de manera más o menos comprensible, se le deberá indicar en el bloque receptor *LoRa* de *Node-RED* que los datos se interpreten con el *código ASCII*.

A continuación se van a indicar los pasos a seguir para la verificación de la conectividad entre el **nodo** y **servidor** (pasando por el *gateway*).

- Escribir en el código que se quiere hacer una vez que esté unido el nodo al servidor.
  - En la variable *data* se indica lo que se quiere mandar.
  - Con el siguiente comando se realiza la transmisión, indicando por que puerto se quiere realizar la transmisión, que datos se quieren enviar y si se quiere (*1*) o no (*0*) que el servidor mande un mensaje de respuesta, indicando que los datos han llegado correctamente.

    ```
    portSend = 1
    data$ = "Hello World!"
    rc = LORAMACTxData(portSend, data$, 1)
    ```

- Se compila y carga el programa como se explicó anteriormente.
- Antes de ejecutar el programa, se va a conectar el *servidor LoRa* a la misma red que se tenga el ordenador, al igual que se ha hecho al configurar los parámetros de éste, para así poder entrar al servidor desde un explorador.
- Se entra a la aplicación *Node-RED* mediante introduciendo en el explorador lo siguiente: **192.168.2.1:1880** (en caso de tener la *dirección IP* por defecto).

- Estando dentro de *Node-RED* se va al bloque que se ha puesto antes para poder ver los mensajes recibidos vía *LoRa*, concretamente en la **pestaña *msg.payload***.

- Se ejecuta el programa en *UWTerminalX*.

- Se verifica en *Node-RED* que el mensaje ha sido recibido correctamente después de haberse creado la sesión entre el nodo y el servidor a través de *OTAA*.

En la figura 2.7 y figura 2.8 se ve el correcto funcionamiento de la transmisión de mensajes por *LoRa*. En la primera se ve que del *módulo RM186* se ha **transmitido** el mensaje correctamente y en la segunda que se ha **recibido** correctamente en el *servidor LoRa*. Además en la primera figura también se ve el ***ACK* recibido** en el módulo (*RX sequence completed*), indicando que se ha recibido el mensaje correctamente en el servidor.

Figura 2.7: Transmisión por LoRa del módulo RM186

2.5. Unión entre tecnologías

Una vez desarrolladas las dos partes independientemente, la captación de *mensajes BLE* y la transmisión de *mensajes LoRa*, se va a realizar la **pasarela** entre estas dos tecnologías de la manera más **simple** para verificar que la configuración se ha realizado correctamente. En el *Capítulo 3* se buscará la manera de que se emitan los mensajes recibidos al módulo de la manera más eficiente posible en términos de energía (haciendo diferentes pruebas),

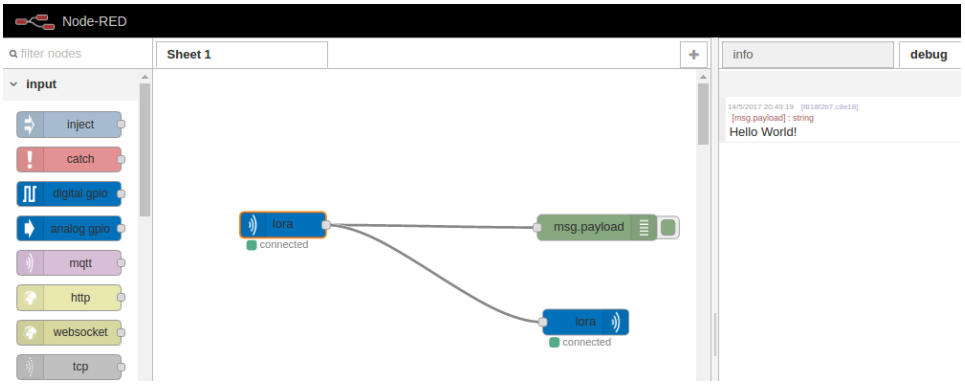


Figura 2.8: Recepción por LoRa en el servidor LoRa

intentado que se pierdan el mínimo número de paquetes posibles, pero esto ahora no es relevante.

En el ejemplo del apartado anterior solo se han captado mensajes de una sola *baliza BLE* y ahora se va a realizar la prueba en el mismo escenario, para que la verificación siga siendo sencilla. Para ver si se consigue hacer llegar un *paquete BLE* al *servidor LoRa* de manera independiente, realizar el escenario con un solo dispositivo es suficiente.

Para esto, no hace falta ningún *buffer* complejo, ya que solo se quiere mandar la *dirección MAC* de la baliza. Por tanto, al llegar un mensaje *BLE*, se va a guardar su *MAC* en una variable y luego se va a transmitir esa misma variable por *LoRa*. Por tanto, se van a mezclar los dos programas creados antes.

Una vez hecho el código, los pasos a realizar van a ser los siguientes:

- Compilar y cargar el software creado en el *módulo RM186*, como se explicó anteriormente.
- Configurar el módulo y el servidor de la misma manera que antes (esto solo es necesario hacerlo una vez, si ya esta hecho se puede saltar este paso).
- Entrar en el *servidor LoRa*, concretamente en la aplicación *Node-RED*, de la forma comentada anteriormente.
- Verificar la correcta transmisión del *paquete BLE*, a través de *LoRa*.

En el programa se ha mandado solo la *dirección MAC* de la baliza recibida, aunque como se ha visto antes, se reciben más datos a través de *BLE*.

Esto se hace para que sea más cómodo de ver si se la transmisión por *LoRa* se ha cursado correctamente, pero en el programa final se transmitirán también los datos restantes. El resultado la transmisión y recepción se puede ver el la figura 2.9 y en la figura 2.10, respectivamente.

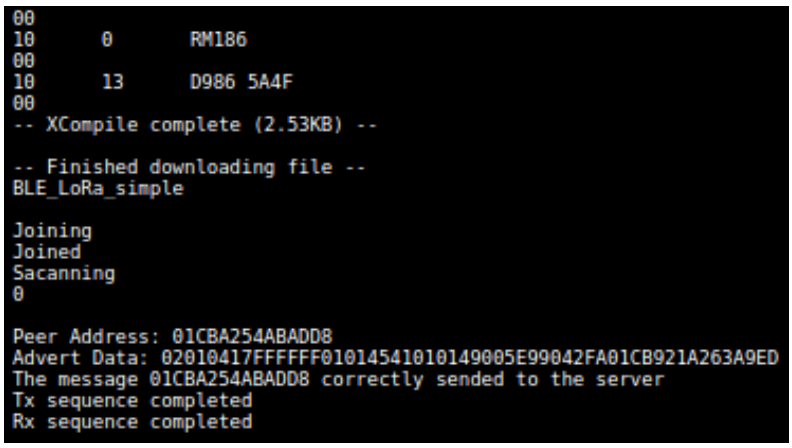


Figura 2.9: Transmisión de un mensaje BLE por LoRa del módulo RM186

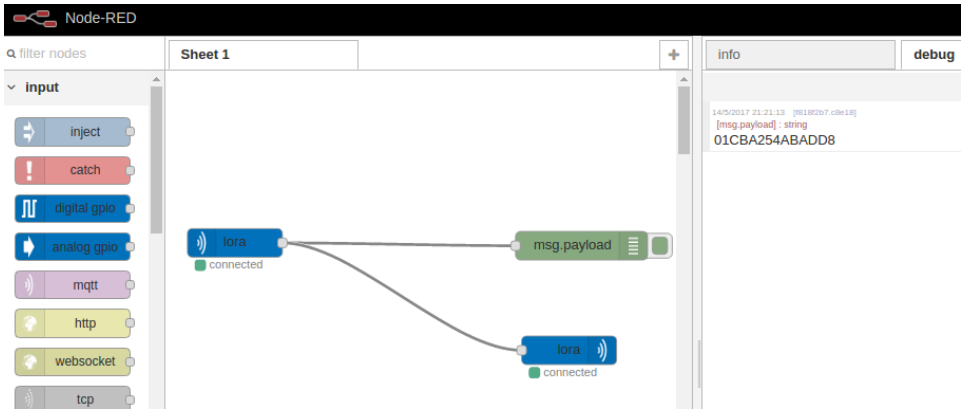


Figura 2.10: Recepción de un mensaje BLE por LoRa en el servidor LoRa

Capítulo 3

Pruebas realizadas en la pasarela BLE - LoRa

Resumen: El capítulo trata sobre las pruebas que se han realizado en las distintas fases de la estructura realizada y explicada en el capítulo anterior, para conseguir realizar la pasarela de forma más eficiente posible en términos de energía. Se piensan unas cuantas propuestas de cómo y cuándo recopilar datos que se obtienen vía BLE y de cómo y cuándo mandar datos vía LoRa. Además, se van a hacer varias pruebas de configuración de los parámetros de ambas tecnologías y luego se van a realizar pruebas en un entorno real, llamadas pruebas de campo. Por último se van a mostrar las pruebas de consumo energético realizadas.

3.1. Propuestas de desarrollo de la pasarela BLE - Lora

En este apartado se han propuesto diferentes maneras de realizar la recepción (vía *BLE*) y transmisión (vía *LoRa*) de los paquetes. Se ha pensado si se van a guardar o no los paquetes recibidos en algún buffer antes de ser enviados, cuando se van a mandar, de que manera se van a mandar, la cantidad de datos que se van a mandar, etc. Se analizan las ventajas y desventajas que se tiene en cada una de ellas para así poder decidir cual es la que más interesa para este proyecto.

No se mencionan todas las propuestas ya que algunas son muy similares a las que se proponen. Se eligen las más importantes, haciendo así un primer filtrado de propuestas. La elección de la propuesta definitiva se hace en el *Capítulo 4*.

Todas las propuestas inician el proceso estableciendo una conexión con el *servidor LoRa*. Una vez que esta hecha se procede a capturar *mensajes BLE* y filtrar estos mensajes o paquetes para que solo queden los que provienen de las balizas creadas por la empresa *Engineea* y *UPNA*.

3.1.1. Sin buffer

La primera propuesta es la **más sencilla** de todas.

En ella se mandan todos los datos que se reciben de manera instantánea. El propio dispositivo tiene una memoria caché en la que se pueden acumular unos pocos paquetes, máximo 10. Esta memoria es configurable.

Cuando se recibe el paquete pasa por un filtro que dice si contiene una serie de características, descritas en el *Capitulo 2*, y si lo pasa se manda automáticamente, sin guardarse en ningún lado.

Una vez que se haya establecido la conexión con el *servidor LoRa*, empezado a captar *mensajes BLE* y haber filtrado los paquetes, las características del proceso de recepción y transmisión de paquetes son las siguientes:

- Se recibe el primer paquete vía *BLE*.
- Se manda el paquete.
- Si no es posible mandar el paquete se descarta.

Ventajas

Esta manera de proceder tiene como ventaja que el desarrollo del software sea muy sencillo. Al no ser de gran complejidad, en caso de ocurrir cualquier error en el desarrollo es muy fácil de corregir.

Desventajas

El hecho de ser sencillo también tiene sus desventajas.

Por un lado, los paquetes no se mandan de manera eficiente. Si se mandan todos de uno en uno nada más recibirlos, hay muchas transmisiones y en estas es donde más energía se consume.

Por otro lado, aunque se llegase a querer hacer así sin importar el consumo energético, se pierden un montón de paquetes. La tecnología *LoRa* no deja mandar paquetes siempre que se quiera, sino que después de una transmisión se queda escuchando a ver si le llega algún paquete abriendo una ventana de recepción y en caso de que no reciba nada abriendo una segunda. Durante

estas ventanas de recepción no permite mandar nada por lo que los paquetes que lleguen en ese intervalo de tiempo, al no guardarse en ningún lugar se perderían. Además, también existe un parámetro llamado *dutty cycle*, en el que hace que no se pueda mandar todo el tiempo por una misma frecuencia. Por lo que al mandar un mensaje por una frecuencia hay que esperar un tiempo determinado para poder enviar otra vez datos por ahí. Es verdad que se pueden mandar los datos por distintas frecuencias, pero éstas no son muchas (6-8 si se configura el servidor adecuadamente) y al mandar muchos datos seguidos como se hace en esta propuesta, estas frecuencias se gastarán rápido. Además, cada banda de frecuencias tiene un *dutty cycle* diferente, por lo que habrá frecuencias que no se puedan ocupar en un gran porcentaje de tiempo (la peor debe estar el 99.9 por ciento del tiempo desocupada).

3.1.2. Un buffer circular

Otra propuesta para la realización de la pasarela es incluir un *buffer circular*. Al llenarse de paquetes este tipo de *buffer*, si se recibe un paquete nuevo se sobrescribirá encima del paquete más antiguo que disponga el *buffer*. Así se consigue que se pierdan los paquetes antiguos en vez de los nuevos.

Se dispone de dos punteros, uno que indica en que posición se va a leer el siguiente paquete para después transmitirlo (*puntero TX*) y otro dónde se va a poder almacenar un nuevo paquete recibido (*puntero RX*). Estos dos punteros están a suficiente distancia para que no se junten nunca, ya que el objetivo de los punteros es separar la recepción y transmisión una de la otra y si se juntan no servirían de nada.

Una vez que se haya establecido la conexión con el *servidor LoRa*, empezado a captar *mensajes BLE* y haber filtrado los paquetes, las características del proceso de recepción y transmisión de paquetes son las siguientes:

- Al llegar el **primer paquete** y se guarda en el *buffer circular* en la posición que marque el *puntero RX*. Indicará la primera posición del *buffer*.
- Siguen llegando paquetes hasta que se llegue a un número determinado de paquetes. Al llegar a este determinado número se manda un determinado número de paquetes.
- Si en un intervalo grande de tiempo no se llega a ese número determinado de paquetes no se van a mandar nunca. Por tanto también se dispone de un *timer* el cual se reinicia en cada envío. Si tras un tiempo determinado no se han enviado paquetes, **salta el *timer*** y en caso de que haya paquetes se envían.

- Al transmitirse el paquete se borra del *buffer circular*.

Ventajas

De esta manera se consigue almacenar paquetes y no perderlos. Además, configurando los punteros correctamente se puede independizar la recepción y transmisión de paquetes.

Desventajas

Lo malo de hacerlo así es que al llenarse el *buffer* se van a **sobrescribir** los paquetes nuevos encima de los antiguos. Esto hace posible que se de el caso en que algunos mensajes de un dispositivo **nunca se lleguen a transmitir** ya que para cuando llega su turno para ser transmitido ya ha sido solapado por uno nuevo de otro dispositivo. Además una mala configuración de los punteros podría hacer que se empezasen a transmitir mensajes nulos y por tanto se gastaría energía inútilmente.

Por otro lado, decir que de esta manera se asegura que se han enviado correctamente los paquetes vía *LoRa*, pero no que los haya recibido el *gateway LoRa*. Esto hace que puedan estar perdiéndose paquetes sin que nadie se entere.

3.1.3. Dos buffers normales y dos timers

Esta propuesta es más compleja que la anterior. En vez de tener un *buffer circular* como en la anterior propuesta se van a tener *dos bufferes normales*. Uno va a servir para recibir paquetes y el otro para transmitirlos. Además se va a hacer uso de dos *timers* para que el programa siga funcionando en situaciones específicas y para que el proceso sea los más óptimo posible en términos de energía.

Una vez que se haya establecido la conexión con el *servidor LoRa*, empezado a captar *mensajes BLE* y haber filtrado los paquetes, las características del proceso de recepción y transmisión de paquetes son las siguientes:

- Se recibe el primer paquete y se añade al *buffer* de recepción (buffer RX).
- Siguen llegando paquetes y se comprueba la *dirección MAC* de esos paquetes. Si coincide con la de algún paquete que ya esta en el *buffer RX*, se sobrescribe. Si no coincide se añade un nuevo paquete al *buffer RX* si no está lleno, ya que si está lleno, en ese momento de la recepción se descartará.

- Para intentar evitar la situación de que se descarte el paquete porque el *buffer RX* está lleno, al llegar un paquete con el cual se llena el *buffer*, los paquetes del *buffer RX* se pasarán al *buffer de transmisión* (*buffer TX*), para ser transmitidos. Al pasar los paquetes de un *buffer* a otro, se borran los paquetes del *buffer RX*.

- Por otro lado, no hace falta que llegue a llenarse el *buffer RX* para que se pasen al *buffer TX*, ya que puede que no se llene nunca. Para ello se ha puesto un *timer* (*timer RX*) y cuando salta se pasan los paquetes de un *buffer* a otro. Al pasar los paquetes se borran del *buffer RX*.

- Cuando el *buffer TX* tenga paquetes se empiezan a mandar poco a poco por *LoRa* al saltar otro *timer* (*timer TX*). Se ha puesto una constante donde se indica el número de paquetes que se van a pasar. No deben enviarse a la vez ni muchos paquetes, porque la tecnología *LoRa* no lo permite, ni pocos, ya que no se estaría haciendo la transmisión de manera eficiente (contra más transmisiones se haga, más energía se consume). Por tanto al saltar el *timer RX* se va a controlar que se transmitan la cantidad de *paquetes BLE* que se haya configurado.

- Aunque se estén transmitiendo los paquetes, siguen llegando paquetes nuevos vía *BLE*, ya que el *módulo RM186* dispone de dos front-en de comunicaciones, una para *BLE* y otra para *LoRa*.

- Cuando se envían paquetes del *buffer TX* al *servidor LoRa* (pasando por el *gateway LoRa*), se van borrando los paquetes del *buffer TX*.

- Al borrarse paquetes en cualquiera de los *dos buffers*, los paquetes restantes suben posiciones hasta ocupar las primeras posiciones. Esto se hace para que al llegar paquetes nuevos (en cualquiera de los *dos buffers*), se añadan debajo del último paquete y así no se tiene que estar controlando que posiciones están libres.

- Si al pasar de un *buffer* a otro, el *buffer TX* tiene menos posiciones libre que paquetes que se están pasando, los que no entren se descartarán. Se debe configurar correctamente los parámetros de transmisión para que se transmita a una velocidad vía *LoRa* que haga que el *buffer TX* intente estar siempre los más vacío posible.

Ventajas

Una de las ventajas de esta propuesta es que al haber *dos buffers* se puede independizar totalmente la recepción y la transmisión, sin tener que preocuparse de configurar correctamente los dos punteros.

Además, hay un control de paquetes recibidos. Al comprobar si hay un paquete con la misma *dirección MAC* en el *buffer RX* y si la hay sobreescribirlo, hace que se ahorren posiciones en el *buffer* y así poder recibir más paquetes y descartar menos. Las *balizas BLE* emiten mensajes cada 4 segundos, por lo que la información de un mensaje a otro será similar. Por otro lado, aunque haya cambios importantes de un mensaje a otro, el último mensaje es el que resulta útil y por lo tanto, no pasa nada por sobrescribir el nuevo en la posición del viejo.

Desventajas

La desventaja más importante de esta propuesta es que es bastante compleja de desarrollar y aunque se desarrolle correctamente, al hacer pruebas y haber algún problema, es difícil de encontrar la causa.

Además, de esta manera se asegura que se han enviado correctamente los paquetes vía *LoRa*, pero no que los haya recibido el *gateway LoRa*. Esto hace que puedan estar perdiéndose paquetes sin que nadie se entere.

3.1.4. Dos buffers normales, un timer y un timeout

Esta propuesta es similar a la anterior, por lo que también es algo compleja. Hay alguna modificación para hacer más eficiente el proceso. Van a seguir habiendo *dos buffers* y en este caso un *timer* y un *timeOut*.

Una vez que se haya establecido la conexión con el *servidor LoRa*, empezado a captar *mensajes BLE* y haber filtrado los paquetes, las características del proceso de recepción y transmisión de paquetes son las siguientes:

- Se recibe el primer paquete y se añade al *buffer* de recepción (*buffer RX*).
- Siguen llegando paquetes y se comprueba la *dirección MAC* de esos paquetes. Si coincide con la de algún paquete que ya está en el *buffer RX*, se sobrescribe. Si no coincide se añade un nuevo paquete al *buffer RX* si no está lleno, ya que si está lleno en ese momento de la recepción se descartará.
- Para intentar evitar la situación de que se descarte el paquete porque está el *buffer RX* lleno, al llegar un paquete con el cual se llena el *buffer*, los paquetes del *buffer RX* se pasarán al *buffer* de transmisión (*buffer TX*), para ser transmitidos. Al pasar los paquetes de un *buffer* a otro, se borran los paquetes del *buffer RX*. A diferencia con la anterior propuesta, va a haber un control de cuantos huecos libres tiene el *buffer RX*. Si hay más paquetes a mandar al *buffer TX* que huecos en este

buffer, solo se pasarán los paquetes que tengan hueco en el *buffer TX*, los demás se quedarán el en de recepción.

- Por otro lado, no hace falta que llegue a llenarse el *buffer RX* para que se pasen los paquetes al *buffer TX*, ya que puede que no se llene nunca. Para ello se ha puesto un *timeout* (*timerOutRX*) y cuando salta se pasan los paquetes de un *buffer* a otro. Al pasar los paquetes se borran del *buffer RX*. Al principio es pequeño, pero luego se incrementa. Esto se hace para iniciar el programa totalmente lo antes posible ya que los envíos se van a hacer cada un intervalo de tiempo una vez enviado un paquete. Esto es, se envía un paquete y se activa un *timer* (timerSend). Al saltar este último *timer* se envían los paquetes vía *LoRa* y se borran del buffer TX. Esto se hace ya que no se pueden mandar dos paquetes *LoRa* muy seguidos y de esta manera se asegura esperar un intervalo antes de enviar el siguiente.

- Por otro lado, si no hay suficientes paquetes para enviar (se ha puesto una constante para poder configurar esto), aunque salte el *timerSend* no se van a enviar. Esto se hace para intentar mandar el máximo número de paquetes en cada transmisión *LoRa*, para así reducir las transmisiones y ahorrar energía.

- Sin embargo, hay un problema. Éste surge cuando hay un intervalo muy grande de tiempo en que no llegan paquetes de dispositivos nuevos y no se añadan más paquetes al *buffer TX*. Los pocos paquetes que hay en el *buffer TX* no se mandarán nunca. Para eso se ha añadido un contador el cual se va incrementando conforme salta el *timerSend*. Llegado a un número determinado en el contador, se transmitirán los pocos paquetes que haya en el *buffer TX vía LoRa* y se borrarán de éste.

- Al borrarse paquetes en cualquiera de los *dos buffers*, los paquetes restantes suben posiciones hasta ocupar las primeras posiciones. Esto se hace para que al llegar paquetes nuevos (en cualquiera de los *dos buffers*), se añadan debajo del último paquete y así no se tiene que estar controlando que posiciones están libres.

- Cuando salte *timerSend* y se intenten mandar paquetes y no haya ninguno o haya menos de los que se necesitan para realizar la transmisión (sin tener que hacer uso del contador), se comprueba en el *buffer RX* si hay paquetes con diferente *dirección MAC* de los que ya tiene el *buffer TX* (si tiene alguno) y si los hay se pasan al *buffer TX*. Una vez hecha la transferencia si ha llegado al número mínimo de paquetes que tiene que haber en el *buffer TX* para poder realizar la transmisión, se realiza la transmisión *vía LoRa*. Si no, se incrementa el contador y sigue el proceso comentado antes.

- Además, si se llena el *buffer TX*, se intenta realizar la transmisión *vía LoRa* automáticamente, sin tener que esperar a que salte el *timerSend*.

Ventajas

Al ser está propuesta más compleja que la anterior, la pasarela se realiza más eficientemente.

Por un lado, al pasar de un *buffer* a otro no se descartan paquetes. En esta propuesta solo se van a descartar cuando el *buffer RX* esté lleno, lo que es más difícil que ocurra que en la propuesta anterior, ya que se hace la transferencia entre *buffers* en más ocasiones.

Por otro lado, tener un control de cuantos paquetes se están enviando es un punto a favor en relación a la optimización de energía. Lo que más va a consumir energía va a ser la transmisión de paquetes *vía LoRa*, por lo que con este mecanismo se ahorra bastante energía. Además, gracias al contador, se consigue que no se quede un paquete en el *buffer TX* sin mandarse indefinidamente, sino que después de saltar unas cuantas veces el *timerSend* y no haberse realizado la transmisión, se mandan los paquetes almacenados sin importar que no lleguen al mínimo número de paquetes con los que se habilita la transmisión.

Además, de esta manera se asegura que se han enviado correctamente los paquetes vía *LoRa*, pero no que los haya recibido el *gateway LoRa*. Esto hace que puedan estar perdiéndose paquetes sin que nadie se entere.

Desventajas

Como se ha visto esta propuesta complica aún más la anterior. Tiene sus ventajas, pero el desarrollo es más complejo. En caso de haber cualquier error es difícil de encontrar el motivo.

Por otro lado, intentar transmitir paquetes cada un intervalo determinado tiene sus inconvenientes. Puede que se necesite enviar más rápido porque el *buffer* se llena a gran velocidad o viceversa porque se intenta transmitir paquetes cuando la tecnología *LoRa* aún no lo permite (ya que como se comentó, después de la transmisión se habilitan dos ventanas de recepción en las que no se puede transmitir nada).

Además, de esta manera se asegura que se han enviado correctamente los paquetes vía *LoRa*, pero no que los haya recibido el *gateway LoRa*. Esto hace que puedan estar perdiéndose paquetes sin que nadie se entere.

3.1.5. Un buffer normal

En esta propuesta se vuelve a utilizar un único buffer, pero esta vez no es circular, sino normal. Se van a usar configuraciones de las anteriores propuestas y además se añade alguna más.

Se vio que los programas escritos con el lenguaje *SmartBASIC* no permiten ejecutar dos eventos a la vez. Esto es, no permite añadir al *buffer* un paquete y borrar otro al mismo tiempo. Esto hace que la recepción y transmisión de paquetes se pueda realizar de forma independiente. Por eso, el uso de un solo *buffer* es correcto y no hace falta que sea circular.

Una vez que se haya establecido la conexión con el *servidor LoRa*, empezado a captar *mensajes BLE* y haber filtrado los paquetes, las características del proceso de recepción y transmisión de paquetes son las siguientes:

- Se recibe el primer paquete y se añade al *buffer*.
- Se activa un *timer* (*timerSend*) y cuando salte se comprobará si hay suficientes paquetes para poder mandar.
- Si no los hay, se incrementa el contador (como en la propuesta anterior).
- Si hay suficientes paquetes o el contador llega a un número determinado, se mandan esos paquetes vía *LoRa* y se borran del *buffer*.
- Una vez enviados se reordena el *buffer* para que los paquetes que queden en el *buffer* ocupen las primeras posiciones. Así, los siguientes paquetes que se añadan no tendrán que comprobar que posiciones están libres, sino que se añadirán detrás del último paquete, en caso de que haya huecos libres.
- Mientras, se van recibiendo más paquetes. Estos comprobarán si hay algún paquete con la misma *dirección MAC* en el *buffer*. Si lo encuentran, se sobrescribirá el paquete nuevo en la posición del antiguo con la misma *MAC*. Si no, esto es, es un paquete transmitido por otro dispositivo, se añadirá un nuevo paquete al *buffer*, si hay hueco.
- Si no hay huecos libres, se intenta transmitir paquetes por *LoRa* y si se consigue se borran los paquetes enviados, se reordena el *buffer* y se añade el paquete recibido. Si no consigue transmitir, porque no está la ventana de transmisión habilitada, se descarta el paquete.

Ventajas

Una de las principales ventajas de hacerlo de esta manera es que usando un solo *buffer* **se ahorra en complejidad** y de la manera que se ha hecho

no se desperdicia mucha energía inútilmente. Además si solo se tiene un *buffer* se ahorra algo de memoria, la que se podrá usar para hacer el *buffer* más grande y poder añadir más paquetes. Por tanto, en nuestro sistema se podrán tener más dispositivos (balizas) *BLE*.

Desventajas

Esta propuesta también tiene desventajas. Intentar transmitir paquetes cada un intervalo determinado tiene sus inconvenientes. Puede que se necesite enviar más rápido porque el *buffer* se llena a gran velocidad o viceversa porque se intenta transmitir paquetes cuando la tecnología *LoRa* aún no lo permite (ya que como se comentó, después de la transmisión se habilitan dos ventanas de recepción en las que no se puede transmitir nada).

Por otro, lado de esta manera se asegura que se han enviado correctamente los paquetes vía *LoRa*, pero no que los haya recibido el g*ateway LoRa*. Esto hace que puedan estar perdiéndose paquetes sin que nadie se entere.

3.1.6. Un buffer normal y control de ACKs

Esta propuesta es similar a la anterior pero con alguna diferencia. Va a haber **un control de correcta transmisión de paquetes** al *servidor LoRa*. En otras propuestas solo se ha preocupado de que los paquetes se envíen correctamente, pero no se ha verificado si han llegado bien al *gateway* y luego al *servidor LoRa?*.

Para solucionar esto, se va a habilitar el envío una vez haya llegado el ACK del *servidor LoRa* indicando que el mensaje se ha recibido correctamente.

Una vez que se haya establecido la conexión con el *servidor LoRa*, empezado a captar *mensajes BLE* y haber filtrado los paquetes, las características del proceso de recepción y transmisión de paquetes son las siguientes:

- Se manda un paquete vacío nada más establecerse la conexión con el servidor LoRa. Esto se hace para tener un primer *ACK* y poder empezar a transmitir los paquetes.
- Empiezan a llegar paquetes y se añaden al *buffer*.
- La adición al *buffer* se hace de la misma manera que en las anteriores propuestas. Se comprueba si hay algún paquete con la misma *MAC* en el *buffer* que la del paquete recibido y si la hay se sobrescribe. Si no, se intenta mandar algún paquete vía *LoRa* para dejar hueco a los

paquetes nuevos, pero para esto, esta vez, tiene que estar la transmisión habilitada, esto es, ha tenido que llegar el *ACK* del paquete anterior.

- Al llegar el *ACK* del paquete vacío se habilita la transmisión. En este momento se podrán transmitir paquetes vía *LoRa* si hay suficientes paquetes para mandar. Si no los hay, se incrementará el contador hasta que llegue a un límite en el que se transmitirán los paquetes que haya en el *buffer* (si los hay).

- Si llega el *ACK* y no hay suficientes paquetes para transmitir, al no mandarse ningún mensaje no se va a recibir un nuevo *ACK*. Por lo tanto, se ha puesto un *timer*, el cual se reinicia con cada envío, con el que cuando se activa, se intenta transmitir los mensajes que hay en el *buffer*. Del mismo modo que antes, si hay pocos paquetes se incrementará el contador. Este *timer* se tiene que hacer lo suficientemente grande para que no salte antes de que se haya recibido un ACK. En caso de que saltase no pasaría nada, ya que la transmisión estaría inhabilitada, pero es más óptimo si no hace ni siquiera intentar enviar.

- Por otro lado, si se llena el *buffer* también se va a intentar enviar. Lo conseguirá si el *ACK* del anterior mensaje ha sido recibido y está la transmisión habilitada.

- Si no llega el *ACK*, internamente salta un *tieme out* notificando que no ha llegado el *ACK*. En este momento se procede a enviar de nuevo el mismo mensaje. Esta acción se realizará hasta que llegue el *ACK* correctamente. En caso de no llegar, se supondrá que se esta demasiado lejos del *gateway* y por tanto la conexión se ha perdido.

Ventajas

La ventaja más importante de este proceso es que asegura que el mensaje ha sido recibido en el servidor antes de enviar el siguiente. Así se tiene mayor fiabilidad y aumenta la calidad del sistema.

Por otro lado sigue habiendo mecanismos como *timers* y contadores con los que solucionar problemas en situaciones específicas.

Desventajas

Una de las desventajas de hacerlo de esta manera es que puede que el dispositivo piense que se ha perdido el mensaje cuando no es así. Esto es, el paquete llega al servidor correctamente, manda un *ACK*, pero este se pierde. Por tanto, el dispositivo emisor mandará otra vez el mismo paquete aunque se haya realizado la transmisión anterior correctamente.

Por otro lado decir que se ha añadido algo de complejidad al sistema, dando posibles complicaciones en su desarrollo y posteriores pruebas.

3.2. Pruebas de configuración de parámetros

En este apartado se van a describir las características de las tecnologías *BLE* y *LoRa* y se van a hacer pruebas cambiando valores de éstas. Todas estas pruebas se van a hacer de forma independiente unas de las otras, para así simplificar la elección de la configuración que se cree que es mejor en cada parte.

3.2.1. Pruebas de configuración de parámetros BLE

En la configuración de la **tecnología *BLE***, lo que interesa es configurar bien la recepción de de mensajes, ya que no se va a transmitir ningún mensaje vía *BLE*.

Las características que son posibles de modificar relacionadas con la recepción de paquetes vía *BLE* son las siguientes:

- **Ventana de escaneo**: Es una ventana en la que se habilita el escaneo *BLE*.
- **Intervalo de escaneo**: Es el intervalo de tiempo que está escaneando dentro de la ventana *BLE*. Si esta es igual que la *Ventana de escaneo* se estará escaneando la mayor parte del tiempo (menos unos pocos milisegundos que pasan entre una ventana y otra, ya que cambia de canal necesitando unos pocos milisegundos).
- **Tipo de escaneo**: Puede ser *Pasivo* o *Activo*. La principal diferencia entre un tipo y otro es que en el primero no hace falta mandar nada para obtener información, sino que solo se tiene que poner a escuchar.
- **Tamaño de la memoria cache del escaneo**: Es es tamaño de la memoria caché que tiene el receptor *BLE*. Aquí se irán almacenando paquetes provisionalmente hasta que lleguen a ser analizados. Si llega un paquete *BLE* cuando esta memoria está llena, se descartará.

Por un lado decir que se quiere hacer un **esceno pasivo**, ya que las balizas mandan mensajes en modo ***broadcast*** y solo hace falta que el receptor se ponga a escuchar, sin necesidad de mandar ningún mensaje ni alterar la red.

Por otro lado, al querer que no se pierda ningún paquete es recomendable que la *Ventana de escaneo* y el *Intervalo de escaneo* sean los mismos.

Además se ha puesto el mayor tamaño de memoria caché que se podía, ya que como se ha comentado, no se quiere que se pierdan muchos paquetes en esta etapa.

Por tanto se van a hacer pruebas solo para ver que duración deben tener la ventana e intervalo de escaneo. No es recomendable que estos valores sean muy pequeños, ya que entre ventana y ventana salta de canal y de esta manera saltaría muchas veces de canal, perdiendo algún paquete entre salto y salto (aunque serían muy pocos paquetes los que se perdiesen al dejar solo de escanear 0.625 milisegundos entre intervalo e intervalo).

Para la realización de estas pruebas se ha hecho un *script* en el que van a **filtrar** todos los paquetes que no lleguen de una baliza determinada (en concreto los que lleguen de la baliza con dirección MAC *CBA254ABADD8*). Se va a hacer un escaneo durante **2 minutos** y se va a contar cuantas veces se ha recibido un paquete de esa baliza. Se van a ir cambiando los valores en un intervalo razonable y en el *Capitulo 4* se analizarán estos resultados y se elegirá el más óptimo.

A continuación se muestran las pruebas realizadas variando la ventana e intervalo de escaneo.

Ventana e intervalo de escaneo de 20 milisegundos

Con la *ventana e intervalo de escaneo* a 20 milisegundos se han hecho las siguientes pruebas:

	Ventana e intervalo de escaneo de 20 ms
1	3 paquetes
2	18 paquetes
3	12 paquetes
4	10 paquetes
5	9 paquetes

Tabla 3.1: Ventana e intervalo de escaneo de 20 milisegundos

En **media** se han recibido **10.5 paquetes** correctamente en 2 minutos de la baliza con dirección MAC *CBA254ABADD8*

Ventana e intervalo de escaneo de 50 milisegundos

Con la *ventana e intervalo de escaneo* a 50 milisegundos se han hecho las siguientes pruebas:

	Ventana e intervalo de escaneo de 50 ms
1	1 paquetes
2	2 paquetes
3	12 paquetes
4	18 paquetes
5	9 paquetes

Tabla 3.2: Ventana e intervalo de escaneo de 50 milisegundos

En **media** se han recibido **8.4 paquetes** correctamente en 2 minutos de la baliza con dirección MAC *CBA254ABADD8*

Ventana e intervalo de escaneo de 150 milisegundos

Con la *ventana e intervalo de escaneo* a 150 milisegundos se han hecho las siguientes pruebas:

	Ventana e intervalo de escaneo de 150 ms
1	4 paquetes
2	4 paquetes
3	6 paquetes
4	5 paquetes
5	6 paquetes

Tabla 3.3: Ventana e intervalo de escaneo de 150 milisegundos

En **media** se han recibido **5 paquetes** correctamente en 2 minutos de la baliza con dirección MAC *CBA254ABADD8*

Ventana e intervalo de escaneo de 300 milisegundos

Con la *ventana e intervalo de escaneo* a 300 milisegundos se han hecho las siguientes pruebas:

	Ventana e intervalo de escaneo de 300 ms
1	4 paquetes
2	5 paquetes
3	7 paquetes
4	10 paquetes
5	5 paquetes

Tabla 3.4: Ventana e intervalo de escaneo de 300 milisegundos

En **media** se han recibido **6.2 paquetes** correctamente en 2 minutos de la baliza con dirección MAC *CBA254ABADD8*

Ventana e intervalo de escaneo de 500 milisegundos

Con la *ventana e intervalo de escaneo* a 500 milisegundos se han hecho las siguientes pruebas:

	Ventana e intervalo de escaneo de 500 ms
1	9 paquetes
2	1 paquetes
3	0 paquetes
4	22 paquetes
5	8 paquetes

Tabla 3.5: Ventana e intervalo de escaneo de 500 milisegundos

En **media** se han recibido **40 paquetes** correctamente en 2 minutos de la baliza con dirección MAC *CBA254ABADD8*

Ventana e intervalo de escaneo de 1000 milisegundos

Con la *ventana e intervalo de escaneo* a 1000 milisegundos se han hecho las siguientes pruebas:

	Ventana e intervalo de escaneo de 1000 ms
1	0 paquetes
2	23 paquetes
3	4 paquetes
4	1 paquetes
5	0 paquetes

Tabla 3.6: Ventana e intervalo de escaneo de 1000 milisegundos

En **media** se han recibido **5.6 paquetes** correctamente en 2 minutos de la baliza con dirección MAC *CBA254ABADD8*

Problemas encontrados en las pruebas realizadas

Como cada paquete se manda de la baliza *BLE* cada 4 segundos, en el *módulo RM186* lo **ideal** es que en 2 minutos se recibiesen **30 paquetes**. Se ha visto que no se ha hecho ni acercar a esta cantidad. Además dentro de la misma prueba ha habido mucha diferencia entre un intento y otro. Por ello, se han estudiado que problemas deben de estar ocurriendo y se ha conseguido llegar a dos motivos causantes del mal comportamiento de la recepción.

Los problemas son los siguientes:

- **Problema 1**: El problema principal de la mala recepción es que se han configurado mal los parámetros en las pruebas. Se ha dicho que para un escaneo del cien por cien del tiempo, sin contar el intervalo que hay al pasar de un canal a otro (0.625 milisegundos), la ventana de escaneo y el intervalo de escaneo deben de ser las mismas, lo que es cierto. Pero el módulo esta internamente programado para que si son iguales estas dos medidas, se escanee el intervalo que se haya configurado y después que se deje de escanear ese mismo intervalo. Por este motivo, el porcentaje de tiempo que se está escaneando se reduce a la mitad.
- **Problema 2**: El segundo problema a sido que el entorno donde se han hecho las pruebas había mucho tráfico de mensajes BLE y se intentaban escanear todos. Al ocurrir esto y procesar cada uno para filtrarlo, se dejaban de escanear muchos paquetes que son útiles.

Soluciones a los problemas

Teniendo en mente estos problemas, se han hecho las mismas pruebas otra vez cambiando lo siguiente:

- **Solución al problema 1**: Para resolver el problema de que no se esté escaneando solo la mitad del tiempo, el cual es el problema más grande, se ha bajado la ventana de escaneo en 2 milisegundos en cada prueba. Con esto hacemos que se esté escaneando casi todo el intervalo de escaneo, haciendo que esos 2 milisegundos no sean relevantes.
- **Solución al problema 2**: Para resolver el problema de que hay mucho tráfico, se han llevado unas pocas balizas a otro entorno y se han intentado escanear solo las de esas, haciendo así que el tráfico disminuya mucho.

Tras hacer las mismas pruebas que se han realizado antes cambiando estas dos cosas que se acaban de comentar, los resultados han sido satisfactorios. En todos los casos se han escaneado **30 paquetes *BLE*** de una sola baliza en dos minutos, es decir, lo **ideal** teniendo en cuenta que estas balizas emiten cada 4 segundos. Por tanto se llega a la conclusión que si se está en un entorno que no tenga tráfico excesivo, la parte de escanear paquetes provenientes de *balizas BLE* es adecuada.

Los resultados de estas últimas pruebas no se muestran ya que fueron los esperados, 30 paquetes en cada prueba e intento, y no era relevante ponerlos.

3.2.2. Pruebas de configuración de parámetros LoRa

En la configuración de la tecnología *LoRa*, lo que interesa es configurar bien la **transmisión** de de mensajes, ya que no se va a recibir ningún mensaje

vía *LoRa*.

Las características que son posibles de modificar relacionadas con la transmisión de paquetes vía *LoRa* son las siguientes:

- **DR** y **SF**: El *Data Rate* y *Spreading Factor* están relacionados. El primero dice cuanta tasa binaria se es capaz de mandar y el segundo cuanta distancia puede alcanzar la transmisión. Conforme el *DR* va subiendo y por tanto mandando más datos, el *SF* va bajando, acortando así la distancia de transmisión máxima posible.

  En la tabla 3.7 se muestran las posibles elecciones que se tienen junto con sus características.

DR	Configuración	Tasa binaria física (bit/s)
0	SF12@125kHz	250
1	SF11@125kHz	440
2	SF10@125kHz	980
3	SF9@125kHz	1760
4	SF8@125kHz	3125
5	SF7@125kHz	5470
6	SF7@250kHz	11000
7	FSK	50000

Tabla 3.7: Características DR [6]

  Se va a utilizar del *DR0* al *DR5*, ya que el *DR6* ocupa mayor ancho de banda y el *DR7* solo es posible usarlo como transmisión de punto a punto, pero esto no es lo que se desea ya que la información debe poder ser captada por más de un *LoRa gateway*.

- **ADR**: El *Adaptive Data Rate* es un algoritmo implementado en los dispositivos *LoRa* para que se regule automáticamente el *DR*. Lo que hace es analizar con que nivel relación señal a ruido, *SNR*, se reciben los datos y sube, baja o mantiene el *DR*. Para la realización de las pruebas, el parámetro ADR se va a dejar desactivado. Para la realizar pruebas del ADR, se necesita hacer pruebas en movimiento y por tanto se hará más adelante al hacer las *Pruebas de campo*.

Aquí interesa saber el comportamiento del sistema con distintas tasas de transmisión y para ello se van a ir configurando manualmente con el **parámetro DR**.

Se van a realizar 3 pruebas distintas y en cada prueba 5 intentos:

- **Prueba 1**: Esta prueba trata de retransmitir vía *LoRa* un solo paquete que llega mediante *BLE*. Este paquete es de alrededor de **37 bytes** y se va a estar mandando durante 2 minutos. Se cuentan el número de paquetes que se manden en ese intervalo de tiempo. Se van a hacer pruebas para los distintos *Data Rate*s.

- **Prueba 2**: La segunda prueba consiste en ver cual es el número máximo de *mensajes BLE* se pueden enviar en una sola *transmisión LoRa*. Este valor irá cambiando conforme se vaya cambiando el *DR*.

- **Prueba 3**: La última prueba es una combinación de las dos anteriores. Se va a coger el número máximo de *mensajes BLE* que se hayan podido mandar en una sola *retransmisión LoRa* y se van a estar mandando durante 2 minutos. Se van contar el número de *mensajes BLE* se pueden retransmitir durante dos minutos, enviando en cada retransmisión el número máximo de *mensajes BLE* que deja enviar ese *DR*.

Remarcar que estas pruebas **no se han realizado en un entorno real**. La distancia entre el nodo y servidor *LoRa* es de unos 3-4 metros, por lo que en un escenario real podría haber más retardo. Después de estas pruebas se van a hacer pruebas reales, pero para la configuración de parámetros, al estar hechas todas las pruebas en un entorno controlado, estos **resultados son más que aceptables** ya que lo que interesa es saber las **diferencias** que hay entre ellas, no los resultados, para así poder comparar y elegir la configuración más óptima. Sin embargo, decir que los resultados no variarían mucho en un entorno real si hubiese suficiente conectividad para enviar los datos correctamente.

Data Rate = 0, Spreading Factor = 12

Con el *Data Rate* a 0 y *Spreading Factor* a 12 se han hecho las siguientes pruebas:

	DR = 0, SF = 12		
	Prueba 1	Prueba 2	Prueba 3
1	6 paquetes	1 paquete	-
2	8 paquetes	1 paquete	-
3	7 paquetes	1 paquete	-
4	9 paquetes	1 paquete	-
5	8 paquetes	1 paquete	-

Tabla 3.8: Prueba DR0/SF12

A continuación se muestra un resumen de los resultados de cada prueba:

- **Prueba 1**: Se observa que en media se han mandado **7.6 paquetes** correctamente en 2 minutos.
- **Prueba 2**: El número máximo de mensajes BLE que se han podido mandar en una sola *transmisión LoRa* a sido de **1**.
- **Prueba 3**: La *Prueba 3* no tiene sentido, ya que va a dar los mismos resultados que la *Prueba 1*, ya que en esta se ha mandado un solo *mensaje BLE* en cada *transmisión LoRa*. Por tanto en media se pueden transmitir **7.6 paquetes** como máximo en 2 minutos con este *DR*.

Data Rate = 1, Spreading Factor = 11

Con el *Data Rate* a 1 y *Spreading Factor* a 11 se han hecho las siguientes pruebas:

	DR = 1, SF = 11		
	Prueba 1	Prueba 2	Prueba 3
1	15 paquetes	1 paquete	-
2	14 paquetes	1 paquete	-
3	14 paquetes	1 paquete	-
4	13 paquetes	1 paquete	-
5	13 paquetes	1 paquete	-

Tabla 3.9: Prueba DR1/SF11

A continuación se muestra un resumen de los resultados de cada prueba:

- **Prueba 1**: Se observa que en media se han mandado **13.8 paquetes** correctamente en 2 minutos.
- **Prueba 2**: El número máximo de paquetes BLE que se han podido mandar en una sola *transmisión LoRa* a sido de **1**.
- **Prueba 3**: La *Prueba 3* no tiene sentido, ya que va a dar los mismos resultados que la *Prueba 1*, ya que en esta se ha mandado un solo *paquetes BLE* en cada *transmisión LoRa*. Por tanto en media se pueden transmitir **13.8 paquetes** como máximo en 2 minutos con este *DR*.

Data Rate = 2, Spreading Factor = 10

Con el *Data Rate* a 2 y *Spreading Factor* a 10 se han hecho las siguientes pruebas:

A continuación se muestra un resumen de los resultados de cada prueba:

- **Prueba 1**: Se observa que en media se han mandado **24.6 paquetes** correctamente en 2 minutos.

	DR = 2, SF = 10		
	Prueba 1	Prueba 2	Prueba 3
1	26 paquetes	1 paquete	-
2	24 paquetes	1 paquete	-
3	24 paquetes	1 paquete	-
4	25 paquetes	1 paquete	-
5	24 paquetes	1 paquete	-

Tabla 3.10: Prueba DR2/SF10

- **Prueba 2**: El número máximo de paquetes BLE que se han podido mandar en una sola *transmisión LoRa* a sido de **1**.

- **Prueba 3**: La *Prueba 3* no tiene sentido, ya que va a dar los mismos resultados que la *Prueba 1*, ya que en esta se ha mandado un solo *paquetes BLE* en cada *transmisión LoRa*. Por tanto en media se pueden transmitir **24.6 paquetes** como máximo en 2 minutos con este *DR*.

Data Rate = 3, Spreading Factor = 9

Con el *Data Rate* a 3 y *Spreading Factor* a 9 se han hecho las siguientes pruebas:

	DR = 3, SF = 9		
	Prueba 1	Prueba 2	Prueba 3
1	43 paquetes	3 paquetes	66 paquetes
2	43 paquetes	3 paquetes	69 paquetes
3	42 paquetes	3 paquetes	60 paquetes
4	41 paquetes	3 paquetes	69 paquetes
5	43 paquetes	3 paquetes	72 paquetes

Tabla 3.11: Prueba DR3/SF9

A continuación se muestra un resumen de los resultados de cada prueba:

- **Prueba 1**: Se observa que en media se han mandado **42.4 paquetes** correctamente en 2 minutos.

- **Prueba 2**: El número máximo de *paquetes BLE* que se han podido mandar en una sola *transmisión LoRa* a sido de **3**.

- **Prueba 3**: Mandando 3 *paquetes BLE* en cada *transmisión LoRa*, en media, se han conseguido transmitir **67.2 paquetes**.

Data Rate = 4, Spreading Factor = 8

Con el *Data Rate* a 4 y *Spreading Factor* a 8 se han hecho las siguientes pruebas:

	DR = 4, SF = 8		
	Prueba 1	Prueba 2	Prueba 3
1	62 paquetes	7 paquetes	140 paquetes
2	53 paquetes	7 paquetes	140 paquetes
3	59 paquetes	7 paquetes	140 paquetes
4	56 paquetes	7 paquetes	140 paquetes
5	55 paquetes	7 paquetes	140 paquetes

Tabla 3.12: Prueba DR4/SF8

A continuación se muestra un resumen de los resultados de cada prueba:

- **Prueba 1**: Se observa que en media se han mandado **57 paquetes** correctamente en 2 minutos.
- **Prueba 2**: El número máximo de *paquetes BLE* que se han podido mandar en una sola *transmisión LoRa* a sido de **7**.
- **Prueba 3**: Mandando 7 *paquetes BLE* en cada *transmisión LoRa*, en media, se han conseguido transmitir **140 paquetes**.

Data Rate = 5, Spreading Factor = 7

Con el *Data Rate* a 5 y *Spreading Factor* a 7 se han hecho las siguientes pruebas:

	DR = 5, SF = 7		
	Prueba 1	Prueba 2	Prueba 3
1	65 paquetes	7 paquetes	231 paquetes
2	60 paquetes	7 paquetes	231 paquetes
3	57 paquetes	7 paquetes	210 paquetes
4	64 paquetes	7 paquetes	217 paquetes
5	58 paquetes	7 paquetes	210 paquetes

Tabla 3.13: Prueba DR5/SF7

A continuación se muestra un resumen de los resultados de cada prueba:

- **Prueba 1**: Se observa que en media se han mandado **60.8 paquetes** correctamente en 2 minutos.

- **Prueba 2**: El número máximo de *paquetes BLE* que se han podido mandar en una sola *transmisión LoRa* a sido de **7**.
- **Prueba 3**: Mandando 7 *paquetes BLE* en cada *transmisión LoRa*, en media, se han conseguido transmitir **219.8 paquetes**.

Se puede subir el *Data Rate* a **6** y a **7**, pero no es recomendable. Además en esos *DR*, la distancia entre el nodo y servidor debería ser muy pequeña y por tanto no es útil usarlos en este proyecto.

Problemas encontrados en las pruebas realizadas

No son muchos los problemas que se han encontrado en las pruebas. Solo decir que se esperaba que con los *Data Rate* más bajos se consiguiese mandar más información, ya que estos van a ser los que se usen en distancias grandes, dando más amplitud al proyecto.

Soluciones a los problemas

Para mejorar la cantidad de datos que se pueden mandar en cada *Data Rate* no se puede hacer mucho. Esto depende de la tecnología LoRa. Pero si que se puede estudiar una forma más óptima de mandar los datos en caso de que el proyecto se quiera hacer más grande. Esto se deja como ampliación futura del proyecto, ya que este proyecto se centra más en otros aspectos.

Si en un futuro alguna empresa está interesada en ampliar más la distancia, se puede hacer que se envíe solo la información que más relevancia tenga para esa empresa, pero este proyecto se está realizando de manera global.

3.3. Pruebas de campo

Además de las pruebas anteriores, se han realizado **pruebas en un entorno real**. Se ha dejado el *gateway y servidor LoRa* en un punto fijo conectado a **Internet** y se ha ido con balizas emitiendo *paquetes BLE* y el *módulo RM186* que hace de **pasarela** entre las dos tecnologías a hacer pruebas de distancia con un coche.

Se podría haber puesto que el módulo *RM186* ejecutara automáticamente el programa solo con alimentarlo y así no tener que llevar el ordenador portátil encima, pero de esta manera no se pude analizar las transmisiones con tanto detalle. Se podría ver los paquetes recibidos con una serie de características (como la frecuencia por la que ha llegado el paquete, la *SNR*, los datos representados de manera legible por los humanos, etc.), pero no se podría asegurar que no hay más paquetes que se han enviado y que se

han perdido antes de llegar al destino. Además, el programa software del módulo tiene configurado que muestre la configuración (como en que DR se transmite, que frecuencias hay disponibles, cuando es posible enviar el siguiente paquete, etc.). Por estas razones se ha decidido hacer pruebas con el ordenador portátil encima.

En concreto se ha ido con **8 balizas *BLE*** y **1 módulo *RM186*** conectado al ordenador para poder ir analizando cosas en tiempo real. Además, al estar el servidor *LoRa* conectado a una red, se ha hecho un **túnel provisional** desde esta red a una red pública para poder conectarse al servidor desde el móvil y así ver si los paquetes se están recibiendo correctamente o no, también en tiempo real.

Se han realizado **dos tipos de pruebas**. Una para **medir la distancia** y otra para **medir el número máximo de datos a poder transmitir en una sola transmisión *LoRa***. Se han creado un par de pequeños programas, de configuración muy sencilla, para poder probar esto antes de tener que elegir la propuesta definitiva de las nombradas en el *apartado 3.1.*, ya que estas pruebas también influyen en su decisión.

Se han recorrido los mismos caminos en las dos pruebas. En la figura 3.1 y en la figura 3.2 se ven estos caminos, indicando donde está la salida y dónde la llegada.

En la figura 3.1 se puede apreciar dos caminos relativamente pequeños. En la figura 3.2 se hizo un camino más largo.

3.3.1. Prueba de distancia

En el *subapartado 3.2.2.* se ha comentado algo sobre el *Adaptive Data Rate*. Se ha dicho que es un mecanismo que regula solo el *Data Rate* según la relación *SNR* con la que lleguen las señales. Por lo tanto, para hacer pruebas de distancia se va a activar esta opción, ya que así se deja a un lado la incógnita de con que **DR** enviar datos. También se podía haber dejado desde un principio el *DR* a 0, ya que es el *DR* que a más distancia puede transmitir y por tanto en esta prueba es lo que interesa, pero se ha activado para que se vea experimentalmente su funcionamiento.

Como se ha mencionado antes, se han recorrido **3 caminos distintos**. A continuación se van a describir los resultados más importantes de estos caminos.

Figura 3.1: Pruebas campo de distancia corta

Figura 3.2: Prueba campo de distancia larga

Camino 1

En este camino, el de color **amarillo** de la figura 3.1, la prueba de distancia no ha sido muy satisfactoria. Al principio del trayecto iba bien la

transmisión, pero cuando se ha topado con un pequeño montículo que hacia perder la visión directa con el servidor situado en el punto de salida, se ha cortado, dejando así de transmitir datos.

Sin embargo se ha seguido un poco más ya que la carretera cogía algo de altura y finalmente se ha vuelto a recuperar la transmisión, aunque no con muy buena calidad. Cuando se ha llegado al *destino 1* de la figura 3.1, se ha vuelto a perder la conexión.

Camino 2

El segundo camino, el de color **verde** de la figura 3.1, ha sido más largo que el primero, pero los resultados han sido mejores que los del primero. Por un lado había mayor visibilidad durante todo el camino y por otro lado, aunque es verdad que había puntos en donde se ha perdido la conexión, el *destino 2* estaba a mayor altura que el 1 y se han podido transmitir paquetes perfectamente.

Camino 3

El tercer camino, el de color **naranja** de la figura 3.2, es el camino más largo. Durante el trayecto se han podido transmitir paquetes en momentos puntuales, pero conforme se iba llegando al *destino 3*, se iba ganando altura y se ha podido transmitir paquetes adecuadamente.

Por tanto, decir que el destino del camino 3 ha sido en donde se ha conseguido mandar a mayor distancia, pero como se ha visto, esto ha sido posible porque al estar más alto se tiene mayor visibilidad.

No se puede especificar una distancia máxima en la que se pueden transmitir datos de manera global, ya que como se ha visto en las dos primeras, estando a menor distancia, se ha conseguido transmitir con mayor dificultad que en la tercera. Esto es debido a que hay factores que influyen como los obstáculos entre emisor y receptor, el tiempo, etc.

Sin embargo, lo que si que se puede afirmar es que teniendo buena visibilidad como la que se dispone estando en el *destino 3* y con el tiempo en una situación no muy favorable (se ha probado en un día de lluvia y funcionaba) y teniendo la antena transmisora dentro de un coche, no fuera, lo que atenúa la señal, se ha conseguido transmitir datos en una distancia de **3.74 Km**. Esta distancia es más que suficiente en lugares en los que no son muy grandes como en puertos, zonas de parques eólicos, etc. sin conexión a Internet.

En cuanto al *ADR*, en los tres recorridos es verdad que conforme el coche se iba alejando, los datos llevaban con menor *SNR* y el *DR* iba bajando. Sin embargo, en el camino de vuelta, al llegar los paquetes con mayor *SNR*, no se incrementaba éste.

3.3.2. Pruebas de DR

Como se acaba de comentar, el *ADR* no funciona correctamente a la hora de bajar el *DR*, por lo que se han hecho pruebas para configurar el *DR* en el propio programa.

El módulo *RM186* dispone de algún botón para configurar. Uno se ha configurado para subir el *DR* y otro para bajarlo.

Se han hecho los mismos recorridos que en la *Prueba de distancia*, pero ahora se ha ido configurando manualmente el *DR*.

Se ha ido a lugares donde se ha sabido por la prueba anterior que se podía mandar datos con *DR = 0*, esto es, se podía mandar un solo paquete *BLE* (37 bytes). Además de estos 37 bytes también se manda información del paquete *LoRa*, como la cabecera, etc., pero en este proyecto interesa saber cuantos paquetes *BLE* se pueden mandar en una sola transmisión, y al ser el resto de datos un valor fijo, al hacer la comparativa se pueden despreciar.

A continuación se van a describir los resultados más importantes de estos caminos.

Camino 1

En el primer camino se ha ido incrementando el *DR* desde algo antes del *destino 1*. Aquí, al seguir teniendo algo de mala visibilidad por el montículo descrito en la *Prueba de distancia*, no se ha podido mandar más de un paquete, aunque se ha conseguido mandar paquetes con *DR* a *1* y a *2* (pero con esto también solo se puede mandar un solo paquete *BLE*).

Camino 2

En el segundo camino, ahora en el *destino 2*, se han conseguido mandar 3 paquetes a la vez al servidor *LoRa*. Esto es se ha conseguido incrementar el *DR* a *3*, como se vio en las *Pruebas de* ***LoRa***.

Camino 3

En el último camino, en el *destino 3*, también se han conseguido mandar 3 paquetes *BLE* a la vez un par de veces, pero luego ha fallado la transmisión. Eso indica que en este lugar, a visibilidad bastante buena, pero a bastante distancia, esta el límite del *DR2* al *DR3*.

Sin embargo, este punto se va a considerar como que se puede mandar paquetes con *DR* a *2*, esto es, un solo paquete *BLE* por cada transmisión *LoRa*, ya que no se puede ofrecer seguridad de que se vayan a poder transmitir 3 paquetes *BLE* siempre.

Con esto se terminan las *Pruebas de campo*, esto es, pruebas en un entorno real y no en un laboratorio. Se ha visto que el comportamiento que tienen los dispositivos en el laboratorio varía bastante al que tienen en un entorno real. Esto es debido a que hay factores que en un laboratorio no se tienen en cuenta en un principio, como el tiempo, los obstáculos, el tipo de obstáculo, la distancia del destino, la visibilidad del destino, la altura, etc.

Pero se ha llegado a la conclusión, que este sistema funciona correctamente en lugares con mucha visibilidad y donde no se necesitan mandar datos en tiempo real.

3.4. Pruebas consumo energético

Por último se van a mostrar las pruebas de consumo energético realizadas. Decir que el consumo energético de la fase de recepción de *BLE* es siempre el mismo, porque está escaneando paquetes casi todo el tiempo y al no dejar casi de escanear no varía. Por tanto se van a centrar las pruebas en la parte *LoRa*. Concretamente se va a ver que diferencia hay entre mandar paquetes del *módulo RM186* al *servidor LoRa* en un *DR* u otro.

Estas pruebas se dividen en dos grupos y son parecidas a las realizadas en las *Pruebas DR*. En primeras pruebas se valora cuanta energía consume con cada *DR* en cada transmisión, mandando por *LoRa* un solo paquete *BLE*. Después, las siguientes pruebas consisten en hacer lo mismo pero enviando el mayor número de paquetes *BLE* posible en cada transmisión *LoRa* según en el *DR* que se esté (estos valores se vieron en las *Pruebas DR*).

Decir que la escala en las figuras es la misma en todas, cada linea horizontal representa 0.4 voltios y cada línea vertical 1 segundo.

Además hay que comentar que en todas las figuras se puede apreciar dos

momentos en el que sube el voltaje. De normal se tiene una tensión de 0.38 voltios debido en su gran mayoría al escaneo de paquetes *BLE*, pero en el momento que sube el voltaje indica que se están mandando los paquetes vía *LoRa* y el segundo momento que sube más levemente indica que se ha abierto la primera ventana de recepción, la cual estará esperando la llegada del *ACK*.

En las pruebas van a interesar saber cuanto valen estas dos subidas ya que así se podrá saber en que caso se está consumiendo más energía.

Se puede saber la intensidad que circula gracias a una sencilla fórmula [12]:

$$I(mA) = \frac{V(mV)}{25{,}5}$$

Por tanto, se van a medir estas dos ventanas que se abren, la de transmisión y la de recepción. Se va a medir tanto el voltaje/intensidad como el tiempo que están abiertas. Decir que al medir el voltaje/intensidad se va a sumar al que ya está debido al escaneo *BLE*. Como nos interesa el total y a la hora de comparar la diferencia entre las distintas pruebas, es preferible mostrar los resultados de esta manera (mostrar la suma de todo, en vez de solo lo consumido por las ventanas).

3.4.1. Pruebas con un solo paquete BLE

Data Rate = 0, Spreading Factor = 12

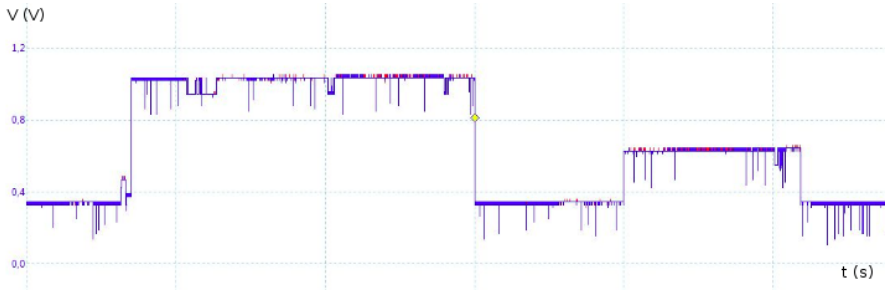


Figura 3.3: Prueba transmisión por LoRa de 1 paquete BLE con DR0

Los resultados obtenidos de la figura 3.3 son los siguientes:

- *Voltaje ventana de transmisión*: 1035 mV
- *Intensidad ventana de transmisión*: 40.59 mA

- *Voltaje ventana de recepción*: 641 mV
- *Intensidad ventana de recepción*: 25.14 mA
- **Tiempo de ventana de transmisión**: 2300 ms
- **Tiempo de ventana de recepción**: 1190 ms

Data Rate = 1, Spreading Factor = 11

Figura 3.4: Prueba transmisión por LoRa de 1 paquete BLE con DR1

Los resultados obtenidos de la figura 3.4 son los siguientes:

- *Voltaje ventana de transmisión*: 1035 mV
- *Intensidad ventana de transmisión*: 40.59 mA
- *Voltaje ventana de recepción*: 641 mV
- *Intensidad ventana de recepción*: 25.14 mA
- **Tiempo de ventana de transmisión**: 1233 ms
- **Tiempo de ventana de recepción**: 592 ms

Data Rate = 2, Spreading Factor = 10

Los resultados obtenidos de la figura 3.5 son los siguientes:

- *Voltaje ventana de transmisión*: 1035 mV
- *Intensidad ventana de transmisión*: 40.59 mA
- *Voltaje ventana de recepción*: 641 mV
- *Intensidad ventana de recepción*: 25.14 mA

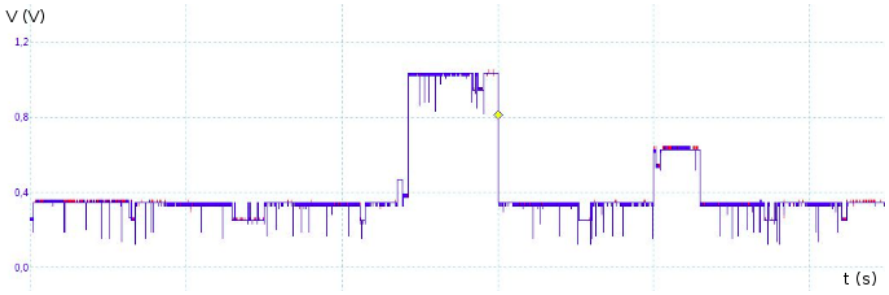


Figura 3.5: Prueba transmisión por LoRa de 1 paquete BLE con DR2

- **Tiempo de ventana de transmisión**: 575 ms
- **Tiempo de ventana de recepción**: 298 ms

Data Rate = 3, Spreading Factor = 9

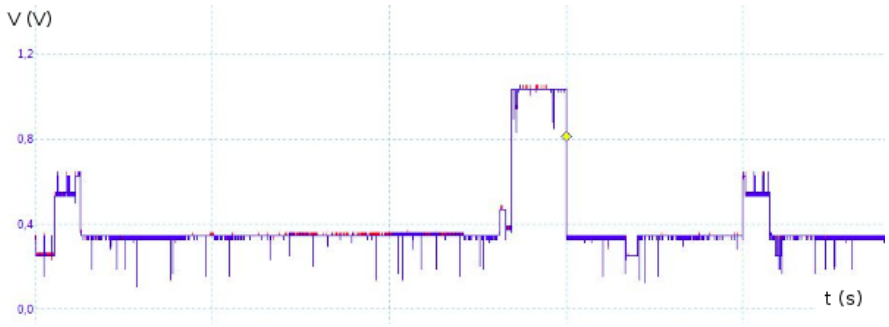


Figura 3.6: Prueba transmisión por LoRa de 1 paquete BLE con DR3

Los resultados obtenidos de la figura 3.6 son los siguientes:

- *Voltaje ventana de transmisión*: 1035 mV
- *Intensidad ventana de transmisión*: 40.59 mA
- *Voltaje ventana de recepción*: 641 mV
- *Intensidad ventana de recepción*: 25.14 mA
- **Tiempo de ventana de transmisión**: 308 ms
- **Tiempo de ventana de recepción**: 151 ms

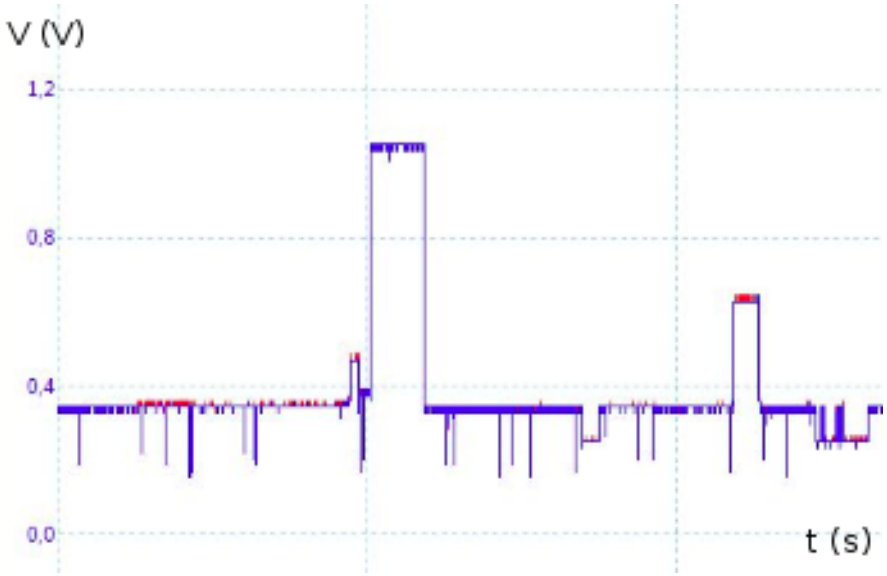


Figura 3.7: Prueba transmisión por LoRa de 1 paquete BLE con DR4

Data Rate = 4, Spreading Factor = 8

Los resultados obtenidos de la figura 3.7 son los siguientes:

- *Voltaje ventana de transmisión*: 1035 mV
- *Intensidad ventana de transmisión*: 40.59 mA
- *Voltaje ventana de recepción*: 641 mV
- *Intensidad ventana de recepción*: 25.14 mA
- **Tiempo de ventana de transmisión**: 172 ms
- **Tiempo de ventana de recepción**: 86 ms

Data Rate = 5, Spreading Factor = 7

Los resultados obtenidos de la figura 3.8 son los siguientes:

- *Voltaje ventana de transmisión*: 1035 mV
- *Intensidad ventana de transmisión*: 40.59 mA
- *Voltaje ventana de recepción*: 641 mV
- *Intensidad ventana de recepción*: 25.14 mA

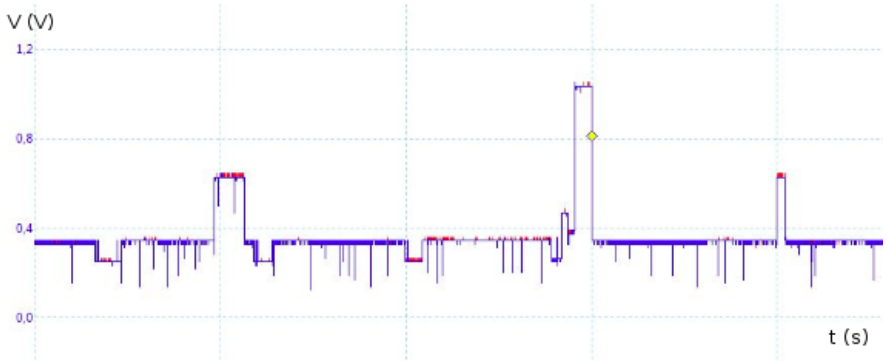


Figura 3.8: Prueba transmisión por LoRa de 1 paquete BLE con DR5

- **Tiempo de ventana de transmisión**: 90 ms
- **Tiempo de ventana de recepción**: 42 ms

Como se acaba de ver, el **voltaje/intensidad** de las dos ventanas **es siempre el mismo** en todas las pruebas. Lo que interesa en este caso es saber **cuanto tiempo dura cada ventana**, tanto la de transmisión (la grande) como la de recepción (la pequeña).

Se puede observar que contra más se sube el *DR* menores son estas ventanas, por lo que indica que se consume menos energía. Al mandarse los mismos datos en todos los casos (un solo paquete *BLE*) se ve que contra mayor es el *DR* más rápido se envía. Además la ventana de recepción que se abre también es menor.

Notar también que entre ventana de transmisión y de recepción hay *1 segundo*, lo cual concuerda con las especificaciones del protocolo *LoRaWAN*. Además tampoco se ve la segunda ventana de recepción, ya que como indica este protocolo, si se recibe el *ACK* en la primera no es necesario abrir la segunda. Decir que en caso de que se abriese, se abriría del tamaño de la ventana de recepción con *DR0*, ya que con ese *Data Rate* se envían los *ACK*s por segunda vez, para asegurarse de que llegan y no se tiene ningún problema debido a la distancia, dentro de lo que permite esta tecnología (si no se recibe con DR0 el ACK significa que o se ha perdido el paquete o la distancia entre el nodo y el servidor es demasiado grande).

Por otro lado, se puede verificar que el módulo usa la tecnología LoRa *Clase A*, ya que como se vio en el *Capítulo 1*, no se ve ninguna ventana extra durante un intervalo de tiempo (*Clase B*) o durante todo el tiempo (*Clase C*), lo que hace que se consuma menos energía.

3.4.2. Pruebas con el máximo número de paquetes BLE

Estas pruebas van a ser parecidas a las anteriores, pero enviando más datos en cada transmisión.

Como se vio en las pruebas *DR*, el número máximo de paquetes a transmitir con *DR0*, *DR1* y *DR2* es *1*, por lo que no hace falta volver a hacer estas pruebas (ya que se acaban de hacer).

Por tanto se van a realizar las mismas pruebas pero solo para transmisiones con *DR3*, *DR4* y *DR5*.

Data Rate = 3, Spreading Factor = 9

Como se vio en las *Pruebas DR*, con *DR3* se pueden enviar como máximo **3 paquetes *BLE*** en cada transmisión, por lo que esta prueba se va a realizar transmitiendo ese número de paquetes.

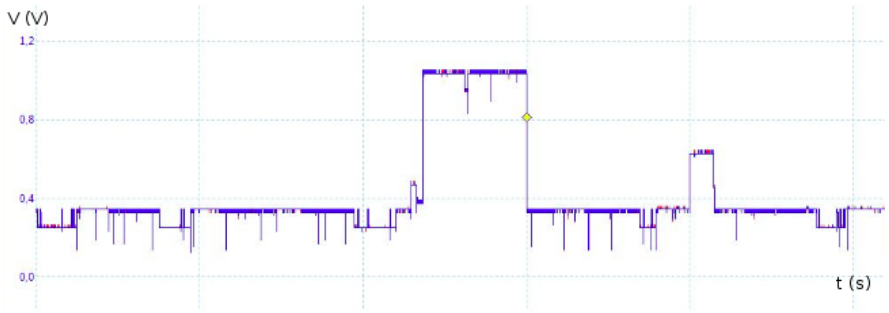


Figura 3.9: Prueba transmisión por LoRa de 3 paquetes BLE con DR3

Los resultados obtenidos de la figura 3.9 son los siguientes:

- *Voltaje ventana de transmisión*: 1035 mV
- *Intensidad ventana de transmisión*: 40.59 mA
- *Voltaje ventana de recepción*: 641 mV
- *Intensidad ventana de recepción*: 25.14 mA
- **Tiempo de ventana de transmisión**: 635 ms
- **Tiempo de ventana de recepción**: 151 ms

Data Rate = 4, Spreading Factor = 8

Como se vio en las *Pruebas DR*, con *DR4* se pueden enviar como máximo **7 paquetes *BLE*** en cada transmisión, por lo que esta prueba se va a realizar transmitiendo ese número de paquetes.

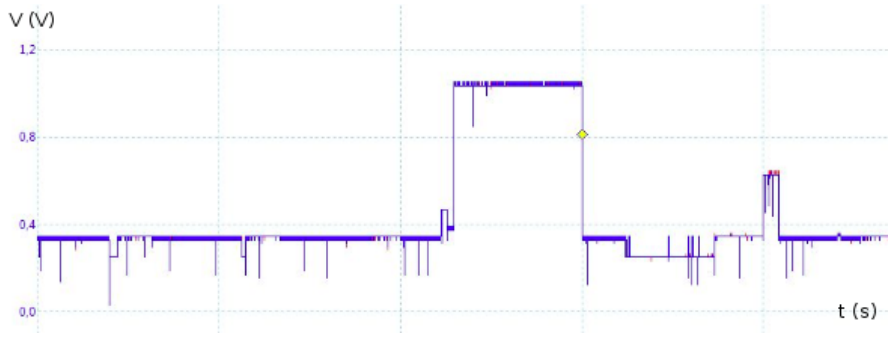


Figura 3.10: Prueba transmisión por LoRa de 7 paquetes BLE con DR4

Los resultados obtenidos de la figura 3.10 son los siguientes:

- *Voltaje ventana de transmisión*: 1035 mV
- *Intensidad ventana de transmisión*: 40.59 mA
- *Voltaje ventana de recepción*: 641 mV
- *Intensidad ventana de recepción*: 25.14 mA
- **Tiempo de ventana de transmisión**: 705 ms
- **Tiempo de ventana de recepción**: 86 ms

Data Rate = 5, Spreading Factor = 7

Como se vio en las *Pruebas DR*, con *DR5* se pueden enviar como máximo **7 paquetes *BLE*** en cada transmisión, por lo que esta prueba se va a realizar transmitiendo ese número de paquetes.

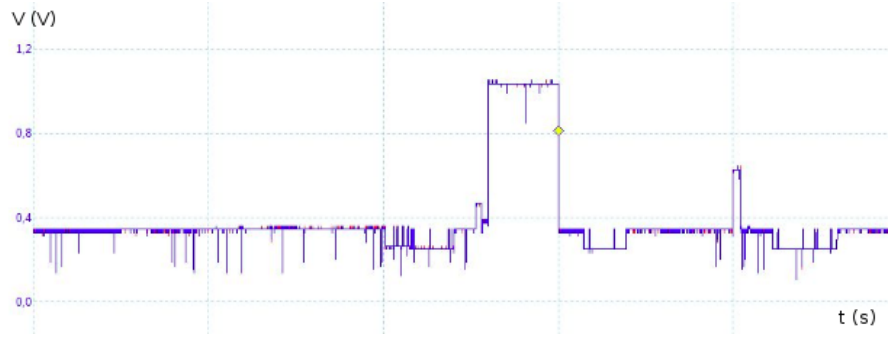


Figura 3.11: Prueba transmisión por LoRa de 7 paquetes BLE con DR5

Los resultados obtenidos de la figura 3.11 son los siguientes:

- *Voltaje ventana de transmisión*: 1035 mV
- *Intensidad ventana de transmisión*: 40.59 mA
- *Voltaje ventana de recepción*: 641 mV
- *Intensidad ventana de recepción*: 25.14 mA
- **Tiempo de ventana de transmisión**: 400 ms
- **Tiempo de ventana de recepción**: 42 ms

Como se esperaba, la ventana de transmisión sube cuando se mandan más datos, pero no sube de manera proporcional al número de paquetes enviados. Esto es, si con *DR5*, enviando *1* solo paquete tarda *90 ms*, si fuese proporcional, al enviar *7* tardaría *630 ms*, pero no es así, tarda *400 ms*. Por tanto, se afirma que **si se manda el mismo número de datos en una sola transmisión es mejor que mandarlo en muchas** (en términos de energía).

Por otro lado, la ventana de transmisión no cambia, lo que es de esperar, ya que esa ventana se abre para recibir el *ACK* y el tamaño de éste va a ser el mismo independientemente los paquetes que se hayan enviado.

Por tanto, se confirma que lo más óptimo en términos de energía es hacer las transmisiones con *DR5* y intentando mandar muchos datos a la vez para que sea más eficiente y aprovechar cada transmisión, pero como se vio en las *Pruebas de distancia*, contra mayor *DR* se tenga, a menos distancia se puede transmitir.

Capítulo 4

Configuración elegida

Resumen: Hechas las pruebas en el Capítulo 3, en este capítulo se analizan los resultados obtenidos en las pruebas y se elige la que se cree que es la mejor configuración. En primer lugar se elige la que se cree que es la mejor propuesta de la estructura del sistema, luego se elige la configuración de las tecnologías BLE y LoRa y por último se describen los parámetros de configuración propios que se han creado en el programa software eligiendo sus valores.

4.1. Elección de la propuesta de funcionamiento de la pasarela BLE - LoRa

En el *Capítulo 3* se han analizado diversos posibles procesos o comportamientos ante eventos del programa software que se va a crear. Se han descrito las características más importantes del funcionamiento de estas propuestas y además se han analizado sus ventajas y desventajas. Como se ha visto todas las propuestas tienen aspectos positivos y negativos ya que es imposible llegar a una solución totalmente óptima. Pero alguna se acerca más que otra al comportamiento que se espera en el sistema desarrollado en este proyecto.

Para empezar, la **primera propuesta** (*hacer transmisiones sin buffer*) se va a descartar ya que es demasiado simple. Es tan simple que hace que no consiga poder mandar la mayoría de paquetes y las pérdidas son enormes.

La **segunda propuesta** *(tener un buffer circular*) es aceptable, pero después se ven mejores soluciones. Está bien, pero se necesita optimizar más el sistema para que haya menos pérdidas de paquetes. Además, como se ve más adelante el hecho de que el buffer sea circular no es necesario.

Las **propuestas 3 y 4** (*tener dos buffers normales*) son las más complejas. En realidad la 4 es una mejora de la 3, por lo que ésta puede ser descartada. Es una solución bastante óptima, pero demasiado liosa. Se barajó la posibilidad de implementar ésta, pero luego surgieron nuevas ideas mejores.

Las **propuestas 5 y 6** (*un solo buffer normal*) también son complejas, pero algo menos que las dos propuestas anteriores. Como ha sucedido antes, la propuesta 6 es una mejora de la 5, por lo que aquí también ésta puede ser descartada. La propuesta 6 recoge muchas características de la propuesta 4, pero la hace más sencilla. En vez de tener que estar pasando de un *buffer* de recepción a otro de transmisión los paquetes para independizar los flujos entrantes y salientes, se juntan en uno solo. Se añade un control del *buffer* para saber cuantos paquetes se disponen en cada instante para poder reaccionar de una manera o de otra, por lo que tener todo en el mismo *buffer* no es un problema.

Por tanto, **se ha elegido la propuesta 6**, ya que es la más óptima en cuanto a consumo de energía, ya que al disponer un control del *buffer*, se han implementado mecanismos para que actúe de una forma u otra según la necesidad que se tenga. Por poner un ejemplo, si el *buffer* está casi vacío esperará a que lleguen más paquetes al *buffer* para poder enviarlos todos a la vez y así ahorrar energía en transmisiones. Si no llegan más paquetes en un tiempo determinado, se mandarán esos pocos para que el servidor tenga constancia de ellos.

La propuesta 4 también tenía implementados este tipo de control, pero tenia que tenerlo en los dos *buffers*, pasándose de uno a otro paquetes según este control. Por tanto, había una complejidad innecesaria y esto hace que en caso de que haya algún fallo, localizarlo sea más complicado.

4.2. Elección de los parámetros de configuración BLE y LoRa

En el *Capítulo 3* se han realizado distintas pruebas en el laboratorio y en un entorno real, donde se ha probado diversas configuraciones de las tecnologías *BLE* y *LoRa*. A continuación se elije las configuraciones que se creen que son más óptimas.

4.2.1. Elección de la configuración BLE

Como se vio, al escanear paquetes *BLE* que mandan las balizas cada 4 segundos, no hay muchos parámetros que se puedan configurar. Como se dijo

el modo de escaneo tiene que ser pasivo, es decir solo ponerse a escuchar sin tener que mandar ninguna petición, para que no se gaste energía inútilmente. Además, al interesar que en esta fase del sistema se pierdan los mínimos paquetes posibles, la memoria caché se va a poner al máximo.

Estos dos parámetros mencionados no hay ninguna duda de que deben de ser así, pero luego están los parámetros de ventana e intervalo de escaneo, en los que se hicieron pruebas para buscar la mejor configuración. Se quiere que en un intervalo de escaneo se esté escaneando todo el tiempo paquetes, esto es el intervalo y ventana de escaneo deben de ser iguales. Pero aunque sean iguales, se pueden configurar de muchas manera, haciendo estos intervalos y ventanas más grandes o pequeñas.

Se observo que internamente el módulo se comportaba de forma anómala, ya que los resultados de las pruebas no seguían ningún patrón lógico. Como se menciono, esto fue debido a que para que éste se comporte de la forma que se desea en este proyecto, la diferencia entre intervalo y escaneo debe de ser de al menos de 2 milisegundos, siendo además el intervalo mayor que la ventana. Con esto se consigue que esté escaneando la mayor parte del tiempo. Además de esos 2 ms que no se escanea nada, hay 0.625 ms que tarda ésta tecnología en pasar de un intervalo a otro, ya que cambia de canal.

El otro problema era que había mucho tráfico y no conseguía procesar todos los paquetes de manera adecuada. Había muchos paquetes que no son relevantes en este proyecto, y aunque se implementó un filtrado para ir descartando estos paquetes, éstos hacían que no se pudiesen recibir otros que si que importan (no conseguían llegar ni a la parte donde se filtran los paquetes, no los procesaba).

Por tanto, viendo los resultados de las pruebas, se han repetido añadiendo 2 ms entre intervalo y ventana y llevando el sistema a un lugar con poco tráfico. Aquí los resultados son satisfactorios. Además, debido a lo que se ha mencionado antes de que entre intervalos de escaneo hay 0.625 ms que no se escanea nada debido al cambio de canal, se ha elegido un intervalo y ventana de escaneo bastante grande para que este tiempo al cambiar de canales ocurra las menos veces posibles. El máximo valor que se les puede dar es de 10.24 segundos, pero se ha elegido de 10 segundos porque si se pone al límite puede dar problemas.

Por lo tanto, la configuración elegida en esta fase ha sido la que se ve en las siguientes líneas de código y además se ve como se configura:

```
#define BLE_SCAN_INTERVAL               10002 // BLE scan
```

```
   interval (miliseconds)
#define BLE_SCAN_WINDOW                10000 // BLE scan
   window (miliseconds)
#define BLE_SCAN_TYPE                      0 // BLE scan
   type --> "0": Passive, "1": Active
#define BLE_SCAN_REPORT_CACHE_SIZE        10 // BLE scan
   report cache size

// ... líneas de código no relevantes ...

rc = BleScanConfig(0,BLE_SCAN_INTERVAL)
rc = BleScanConfig(1,BLE_SCAN_WINDOW)
rc = BleScanConfig(2,BLE_SCAN_TYPE)
rc = BleScanConfig(3,BLE_SCAN_REPORT_CACHE_SIZE)
```

Esto es:

- **Intervalo de escaneo**: 10.002 segundos (los 2 ms de diferencia son para que no haya el problema comentado antes)
- **Ventana de escaneo**: 10 segundos
- **Tipo de escaneo**: Pasivo
- **Tamaño de memoria caché de escaneo**: 10 (el máximo que hay)

4.2.2. Elección de la configuración LoRa

En cuanto a la configuración de *LoRa* también se han hecho varias pruebas y se pueden sacar las siguientes conclusiones:

- Contra más sube el *Data Rate* y más baja el *Spreading Factor* más paquetes se pueden mandar en cada transmisión *LoRa*, pero la distancia a la que podemos transmitir es menor.
- El número de paquetes *BLE* que se pueden transmitir como máximo en una sola transmisión *LoRa* según su *DR* se puede observar en la figura 4.1.

- Aunque al transmitir paquetes *BLE* por *LoRa* con un *DR* distinto se puedan transmitir el mismo número de paquetes, esto no indica que se puedan mandar la misma cantidad de datos por minuto. En la *Prueba 3* se han intentado transmitir el número máximo de paquetes *BLE* que se pueden transmitir por cada *DR*, pero esta vez se van a mostrar los paquetes transmitidos en 1 minuto, no en 2, mostrando su media, en la figura 4.2.

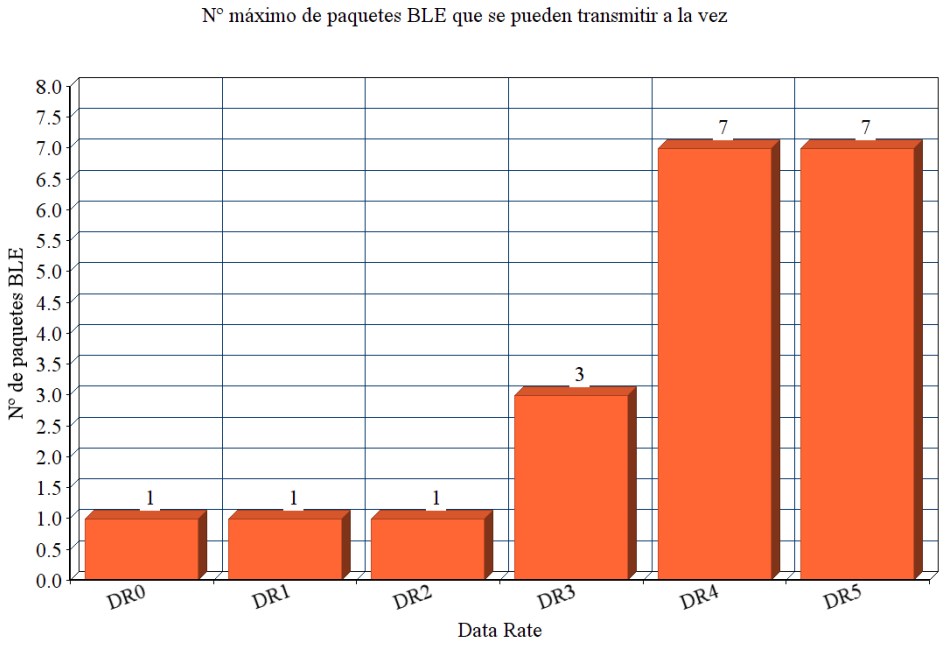


Figura 4.1: Nº máximo de paquetes BLE que se pueden transmitir a la vez

- Como se ve la diferencia entre *DR*s es grande, aunque la *Prueba 2* indique la misma cantidad de paquetes a enviar como máximo en cada transmisión. Esto es debido a que aunque se manden la misma cantidad de datos, contra mayor es el *DR* con mayor velocidad se van a mandar y por tanto se podrá mandar el siguiente paquete *LoRa* antes que en las transmisiones con *DR* más bajo.
- Por tanto, lo ideal sería hacer transmisiones con *DR5*, pero esto en grandes distancias no es posible, como se ha visto en las pruebas de campo.
- Este hecho también se puede justificar al mirar las *Pruebas de consumo energético*. Aquí se ve que contra mayor sea el *DR*, más rápido se mandan los datos (gastando por tanto menos energía), por lo que en un intervalo de tiempo determinado el número de paquetes a poder enviar es mayor.

Además de estas pruebas, también se han hecho pruebas en un entorno real que afectan a la configuración de *LoRa*. Se acaba de mencionar que con un DR alto no se pueden mandar paquetes a mucha distancia. Por tanto se ha pensado en activar la opción que tiene el protocolo *LoRa* llamada *Adaptive Data Rate*. Esta opción hace que se auto-regule el *DR* automáticamente

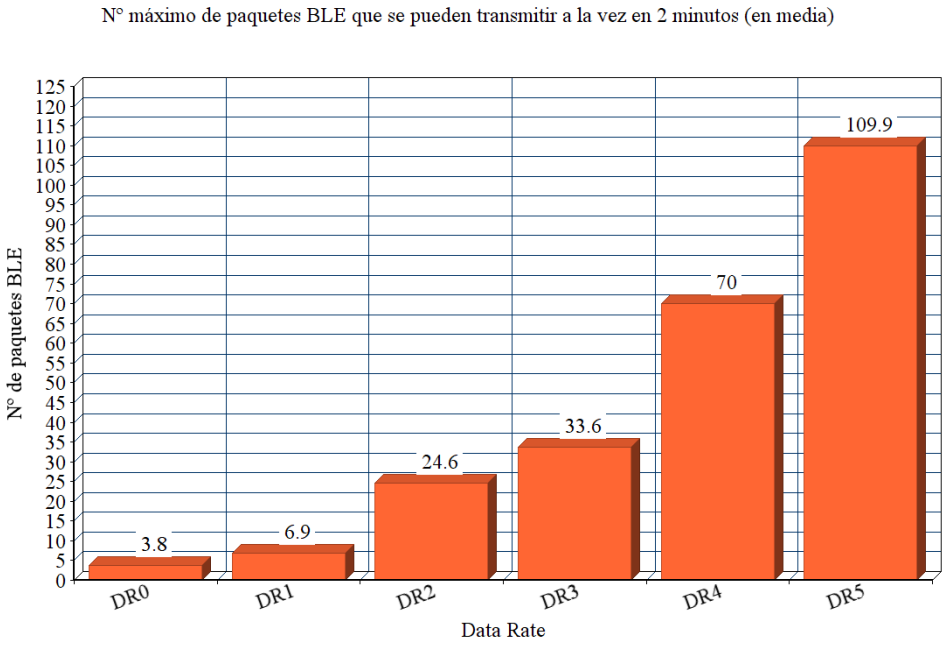


Figura 4.2: Nº máximo de paquetes BLE que se pueden transmitir en 2 minutos

según el estado de la red. Si llegan paquetes con una *SNR* baja se intentará mandar paquetes con un *DR* menor y así hasta que se pueda transmitir correctamente el paquete. Sino se consigue mandar con *DR0*, se supondrá que se ha perdido la conexión entre el nodo y el servidor debido a la distancia o a los obstáculos que se encuentren entre los dos dificultando su visibilidad.

Como se ha visto en estas pruebas en el *Capítulo 3*, el *Adaptive Data Rate* no funciona como se esperaba. Conforme se aleja el nodo del servidor si que baja el *DR*, pero luego no sube. Por tanto esta opción va a estar desactivada.

Según las *Pruebas de consumo energético* realizadas en el *Capítulo 3*, se ha visto que la forma más eficiente de mandar los datos en con *DR5* y enviando el máximo número de paquetes posible. Pero como se acaba de decir, en las pruebas de distancia se ha visto contra mayor es el *DR* a menor distancia se puede transmitir. Por tanto, no se puede especificar un valor fijo del *DR*, va a variar según a la distancia que este el módulo *RM186* del servidor *LoRa*.

Por tanto se ha decidido implementar en el programa software creado,

la acción de subir y bajar el *DR* de forma automática, pero no como lo hace el *ADR*. En el caso de que un paquete no llegue correctamente, se va a bajar el *DR* y también el número de paquetes *BLE* a enviar en cada *DR*, según lo visto arriba de este apartado. De vez en cuando se subirá de forma automática el *DR* para ver si puede enviar los paquetes de esa forma y si no puede se volverá a bajar.

Por tanto la configuración en esta parte se va a hacer de forma medio automática, ya que no hay más parámetros a modificar desde el propio programa en esta fase. Si que los hay introduciendo comandos *MAC*, como se vio en el *Capítulo 2*, al desarrollar la estructura del sistema. En este capítulo se fueron viendo estas configuraciones paso por paso y fase a fase, pero el objetivo de este capitulo no es configurar esas características, sino las características de *LoRa* que se pueden configurar en el propio programa software escrito en el lenguaje *SmartBASIC*.

4.3. Configuración de los parámetros propios del programa software creado

Además de la configuración de las dos tecnologías usadas, se han **creado más parámetros** en las distintas propuestas, de manera que se pueden modificar de manera sencilla y así poder hacer pruebas.

Como ya se ha elegido la propuesta, se van a explicar los parámetros de la propuesta elegida y lo que supone configurarlos de una forma o de otra. Estos parámetros son **fáciles de modificar** ya que, como se ha dicho, el programa ha sido desarrollado con el propósito de que estos parámetros se puedan configurar para ver cuales son las mejores maneras de conseguir un sistema lo más óptimo posible.

Como se ha mencionado al principio de este capítulo, se ha elegido la propuesta 6 y en ella las características que son posibles de modificar relacionadas con el proceso *pasarela BLE-LoRa* son las siguientes:

- **Tamaño buffer**: En la propuesta 6 hay un único *buffer* normal. Aquí se indica la longitud de éste.
- **Paquetes BLE a enviar**: Aquí se indican en número de paquetes que se van a enviar por cada transmisión como máximo. Como se ha comentado anteriormente, este número va a cambiar según el *DR*. Además de ser el número máximo, también intenta ser el mínimo. Es decir, si no hay ese número de paquetes en el *buffer*, no se envía nada aunque le toque enviar (que será cuando llegue el *ACK* del paquete anterior,

indicando que se puede realizar la siguiente transmisión).

- **Duración timer**: Como se acaba de mencionar, hay veces que no se envían paquetes porque no hay suficientes paquetes para enviar (o no hay ninguno). Este *timer* se a puesto para que salte cuando lleva mucho tiempo sin enviarse paquetes (ya que éste se reinicia con cada transmisión) y así comprueba si el *buffer* está más lleno que antes y llega al número mínimo para poder transmitir. Sino es así, se reinicia el *timer* y se vuelve a comprobar cuando vuelva a saltar. Aquí se define la duración de éste. No debe de ser ni muy pequeño, porque saltaría antes de que llegase el *ACK* del paquete anterior en los casos de que se haya mandado antes un paquete, y por tanto saltaría inútilmente, y tampoco puede ser muy grande, ya que si se deja mucho rato puede llenarse el *buffer* y no poder recibir o guardar más paquetes, por lo que habría pérdidas. Hay que buscar un punto medio.

- **Contador enviar**: Como se ha mencionado, si no hay suficientes paquetes para enviar no se envía ninguno. Pero si pasa el tiempo y no llegan más paquetes esos pocos paquetes se tienen que mandar de alguna manera, ya que sino el servidor no tendrá noticia de ellos y habrá sido como si hubiese una pérdida (aunque se manden mucho tiempo después, la información que proporcionan está des-actualizada y por tanto no sirven para nada esos paquetes). Por eso se ha creado un *contador* el cual se incrementa de uno en uno cada vez que salta el *timer*. Cuando llega al número que se configura aquí, se mandan todos los paquetes que haya en el *buffer* (en caso de que los haya, si no los hay se reinicia el *contador* otra vez). No hay problema en transmitir todos los paquetes *BLE* que haya en el *buffer* porque si ha saltado el contador es porque hay menor número de paquetes para enviar de lo que en realidad se puede enviar con el *DR* que haya en ese momento.

Por tanto, teniendo estos parámetros a configurar una vez desarrollado el programa con la propuesta 6 y los parámetros de configuración de las tecnologías *BLE* y *LoRa* explicados anteriormente, se han realizado varias pruebas para ver que configuraciones son las más óptimas. No se va a explicar el proceso de éstas ya que no es relevante. Solo se han ido cambiando de valores de manera independiente.

Lo que se va a explicar son los resultados que se han tenido y la configuración final que se va a tener en el programa software.

Primero se han realizado pruebas respecto al **tamaño del buffer**. Se quiere que este sea lo más grande posible, ya que así podrá almacenar más paquetes en sitios donde haya muchos dispositivos. Como cada dispositivo

ocupa una posición del *buffer*, no más, aunque se manden muchos paquetes del mismo dispositivo o baliza, ya que se sobrescriben los mensajes. Esto se hace porque un mensaje nuevo va a tener información más actualizada que uno viejo. Por tanto se ha intentado hacer grande éste, pero por motivos de memoria del módulo, no se ha podido hacer más grande que el de 100 posiciones, esto es, se pueden almacenar mensajes de **100** dispositivos de manera simultánea. Al intentar incrementar este valor, daba error al no tener suficiente memoria para procesar todo. Recordar que cada mensaje de cada dispositivo ocupa 37 bytes, por lo que el *buffer* ocupará **3700 bytes**.

Después se ha querido configurar los **paquetes a enviar**. En un principio se iba a poner un número fijo, pero al hacer las *Pruebas de DR*, se vio que el número de paquetes *BLE* que se pueden enviar en cada transmisión como máximo varía según el *DR*. Por lo tanto este parámetro va a ir **variando automáticamente** y no se debe de configurar como un parámetro fijo. Se puede decir que inicialmente valga 7, ya que el *DR* va a ser 5 al principio, pero puede que la distancia hasta el servidor sea grande y este vaya bajando.

En cuanto a la duración del **timer** se han probado diversas opciones. Como se ha dicho este valor no debe de ser ni muy grande ni muy pequeño. Al final se ha puesto **1 minuto** de duración. Así da tiempo de sobra a que llegue el *ACK* anterior si es que se ha mandado algún paquetes y en caso de que no, no espera mucho tiempo para comprobar cuantos paquetes hay en el *buffer* y si se pueden enviar alguno en ese instante. Además, para la configuración del contador va a ser más sencillo si se pone ese valor.

Como se acaba de decir, tener el *timer* de duración de 1 minuto es más sencillo a la hora de elegir el **contador**. Por cada vez que se incremente en uno el contador, significará que ha pasado 1 minuto. Por tanto, al querer que si hay pocos paquetes para mandar se manden cada un intervalo considerable de tiempo, se ha puesto un **valor de 60** en el *contador*. Esto es, si después que pase 1 hora no han llegado más paquetes con los que se llegue al mínimo de paquetes requerido para enviar, se mandarán todos los pocos paquetes que haya en el *buffer* (en caso de que haya alguno). Una vez que se hayan enviado los paquetes o haber visto que no hay ninguno en el *buffer* después de llegar el contador al valor determinado, se reinicia éste, por lo que deberá esperar otra hora para poder mandar los paquetes en caso de que llegasen muy pocos paquetes en esa hora.

Por tanto, esta es la configuración que se va a implementar en el software. Decir, que como el programa está pensado para que estos valores sean muy fáciles de modificar, en caso de querer configurarlos de manera más específica en una situación en concreto, no hay ningún problema. Por ejemplo, si

interesa tener la información actualizada lo más rápido posible, se puede disminuir el contador a 1 y así poder mandar todos los paquetes que haya en el *buffer* instantáneamente, independientemente si se llega al número mínimo de paquetes a enviar. Eso si, eso consume más energía.

Capítulo 5

Conclusiones

Resumen: Este capítulo hace una pequeña conclusión al proyecto, analizando lo que se ha conseguido y lo que falta por hacer.

5.1. Conclusiones

Como se ha visto durante este documento, se ha conseguido que los paquetes que emiten las balizas a través de *BLE*, se escaneen y se retransmitan hasta un *servidor LoRa* vía *LoRa*.

Para ello ha sido necesaria la configuración de varios parámetros, tanto en el módulo *RM186*, como en el *gateway y servidor LoRa*. Las balizas estaban ya configuradas para que emitiesen paquetes cada 4 segundos y no se ha tenido que hacer nada.

Además de la configuración, se ha desarrollado un programa software con el lenguaje *SmartBASIC*, el cual se ejecuta en el *módulo RM186*. Este software consigue hacer la **pasarela entre paquetes *BLE* y *LoRa*** que tenía como uno de los objetivos este proyecto.

Por otro lado, para que la conectividad entre las balizas y el servidor sea correcta, se ha desarrollado el software pensando también en **optimizar el consumo energético**. Por ello, se ha pensado una forma de mandar los datos recibidos de las balizas BLE al *servidor LoRa*, vía *LoRa* (cada cuanto enviar, cuantos datos enviar, qué comportamiento tener ante diferentes circunstancias, etc.) de forma que consuma la menor energía posible. También se han configurado las tecnologías *BLE* y *LoRa* de la manera más óptima posible para así poder cumplir el segundo objetivo que tenía el proyecto.

Recordar que se quiere que el módulo **consuma la menor energía posible** consiguiendo cumplir su función de **pasarela**, ya que se quiere que estos módulos sean **autónomos** además de independientes. Esto se puede implementar en **sitios donde no hay conexión a Internet**, para así conseguir llevar la información hasta el *servidor LoRa*, el cual estará a unos pocos kilómetros, y así éste al tener conexión a Internet poder enviar los datos al *servidor general*. Por tanto, si el consumo del módulo es bajo, se puede poner en lugares donde no se pueda conectar a una red eléctrica. Solo necesita una pila de botón y una placa solar que la vaya recargando. Así este se puede convertir en un sistema autónomo y esto ahorra costes en mantenimiento (no quita totalmente estos costes, ya que se puede estropear o dañar por distintas circunstancias, pero ahorra estar yendo a recargar los módulos uno por uno).

Como se ha visto en las *Pruebas de consumo energético* del *Capítulo 3*, contra mayor sea el *DR* más óptimo va a ser. Esto es, se va a enviar cada paquete consumiendo menos energía que si se transmitiese con otro *DR* más bajo. Pero por lo que se ve en las *Pruebas de distancia* también del *Capítulo 3*, cuanto mayor es el *DR*, menor es la distancia a la que se puede transmitir, por lo que se ha implementado un mecanismo que hace que se vaya regulando según la distancia a la que esté.

En el *Apéndice B* se han hecho unos cálculos de la energía que se consume. Para ello se ha supuesto que se transmiten 7 paquetes *BLE* durante 2 minutos (se transmite un solo paquete *BLE* en cada transmisión *LoRa*). Se ha supuesto la transmisión de la misma cantidad de datos en todos los casos para poder compararlos entre ellos. Se han transmitido 7 paquetes *BLE* porque es el máximo que se puede transmitir con *DR0* en 2 minutos. Se ha hecho esto con diferentes DR y en la figura 5.1 se observan los resultados obtenidos.

Se observa que contra más crece el DR, menos energía se consume al mandar los paquetes, pero la diferencia no es muy grande. Sin embargo, si se mandasen más paquetes la diferencia sería mayor, pero no se ha podido probar ya que la tecnología LoRa no lo permite. Con DR0 por ejemplo, no ha dejado transmitir más de 7 paquetes (en media) en 2 minutos. Por tanto no se han podido hacer más pruebas en las que se puedan comparar los resultados, pero se puede asegurar, que **lo más óptimo en términos de energía es transmitir los paquetes con DR5**.

Comentar también que, como se ha visto en los trozos de código insertados en esta memoria, se ha programado poniendo los nombres de las variables y las anotaciones en **inglés**. Esto se ha hecho así ya que el inglés está exten-

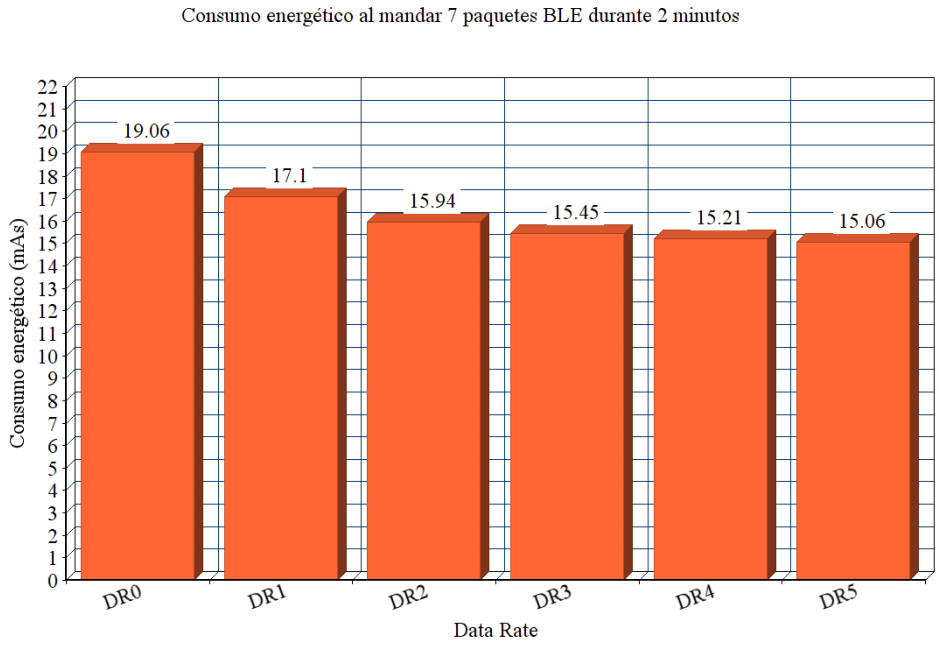


Figura 5.1: Consumo energético al mandar 7 paquetes BLE en 2 minutos

dido por la mayor parte del mundo y de esta manera cualquier persona de cualquier país podrá entender mejor el código sabiendo que significan esas variables y pudiendo leer anotaciones.

Por tanto el proceso del software desarrollado es el siguiente, de manera muy resumida:

- Captura paquetes *BLE*.
- Filtra paquetes y se queda con los que interesan.
- Los almacena.
- Cuando la tecnología *LoRa* le deja o cuando lo crea oportuno se transmiten paquetes encriptados vía *LoRa* (aquí está el proceso que se ha desarrollado para que consuma lo menos posible, descrito en el *Capítulo 3*, en la *Propuesta 6*).
- El *gateway LoRa* los recibe y se los pasa al *servidor LoRa*.
- Éste los desencripta y los procesa.
- De momento se ha configurado para que se pueda ver los bytes que han llegado en los paquetes interpretados con el *código ASCII* para

que sean legibles. También por que *frecuencia* ha llegado, cual es el *tamaño del paquete*, el *Data Rate*, etc.

Como se ha visto en las *Pruebas de distancia*, en un entorno urbano, aunque con buena visibilidad pero con las condiciones del tiempo no muy favorables, se pueden transmitir paquetes a una distancia de **3.74 km**. Por tanto, este sistema en un lugar que no sea muy grande como en un parque eólico pequeño o un puerto es suficiente.

Como se ha visto en las *Pruebas de consumo de energía* transmitir a esa distancia hace que se consuma más que a una distancia más corta (ya que se deberá usar DR0), pero da más cobertura. Sin embargo hay que decir que, en estos casos, aunque se consuma más, está alrededor del consumo que se tenía estimado, por lo que es aceptable.

5.2. Líneas futuras

Como se comentó en el *Capítulo 1* el objetivo del proyecto es conseguir desarrollar una pasarela entre la tecnología BLE y LoRa, la cual sea lo más eficiente posible en términos de energía, para ofrecer una **cobertura global** a un proyecto más general desarrollado por la empresa *Engineea* y la *Universidad Pública de Navarra* (proyecto *LOCALIZA*).

Éste tiene como objetivo conseguir escanear *paquetes BLE*, como en este proyecto, pero esta vez con un **teléfono móvil** en vez de con un módulo. Los teléfonos, al tener **conexión a Internet**, son capaces de mandar la información que han recibido de los *paquetes BLE* directamente al servidor general.

Por tanto, en lugares donde **no se disponga de conectividad a Internet, no funcionaría el sistema**. Por eso el proyecto que se ha realizado con la tecnología *LoRa* ha sido para **cubrir estos sitios donde no hay Internet**, ya que ésta tecnología consigue transmitir datos hasta un máximo de 10 km en campo abiertoy en sitios urbanos unos 2-3 km.

Entonces se ve que el proyecto realizado se tiene que expandir más, no solo tienen que llegar los datos hasta el *servidor LoRa*. El siguiente paso (el cual se sale del propósito de este proyecto) es que una vez recibidos los datos en el *servidor LoRa*, **mandarlos al servidor general vía *TCP/IP***. Se deberán **mandar de la misma manera que mandan los teléfonos móviles** en el otro proyecto, para que **el servidor general los pueda interpretar**. Además, seguramente habrá que configurar algo del servidor general para que admita conexiones entrantes por parte del *servidor LoRa*.

Hay que comentar que este proyecto se ha realizado pensando en esta última fase. Se ha intentado transmitir los datos de manera que sea fácil sacar y administrar la información en el *servidor LoRa*, para luego poder mandar los datos de manera adecuada al servidor global.

Si se consigue hacer este último paso, el proyecto realizado se podrá implementar al otro desarrollado por la empresa *Engineea* y *UPNA*, consiguiendo así **dotar de cobertura global al proyecto**.

Por otro lado, en un futuro se quiere **sustituir** el *módulo RM186* por un **hardware desarrollado por la propia empresa *Engineea* y *UPNA***. Como se ha conseguido que el módulo sea eficiente en términos de energía, solamente con una pila de botón y una placa solar se podrá alimentar, haciendo al módulo autónomo además de independiente. De esta manera se asegura un **mayor control** de los dispositivos hardware del proyecto y es **más económico**. El software creado tendrá que poder ejecutarse en el módulo creado sin problemas.

Apéndice A

Funciones creadas para facilitar la programación en SmartBASIC

Resumen: En este apéndice se muestran los trozos de código que se han creado para facilitar la programación en el lenguaje SmartBASIC, ya que no dispone de muchas funciones que otros lenguajes más completos si que tienen.

Esta función sirve para **pasar una variable tipo *Integer* a una de tipo *String***:

```
function Int_to_String(int) as string
        dim string$
        sprint #string$, int
endfunc string$
```

Ésta en cambio es para **convertir un byte** (uno solo, no más) que está guardado en una variable *String* **en forma *Hexadecimal*** a un byte representado en **forma *Binaria*** en una variable también *String*:

```
function Hex_to_Sting(hex$) as string
        dim dec, bin$
        dec = StrHex2Bin(hex$,0) // Decimal Int
        sprint #bin$, integer.b'dec //'
        bin$ = right$(bin$,8)
endfunc bin$
```

En alguna ocasión se ha necesitado que un ***0*** guardado en una variable *String* se **convierta en un *false*** o que un ***1*** **se convierta en un *true***. Para que esto sea más sencillo se ha creado la siguiente función:

```
function boolean(bit$) as string
        if strcmp(bit$, "0") == 0 then
                bit$ = "false"
        endif
        if strcmp(bit$, "1") == 0  then
                bit$ = "true"
        endif
endfunc bit$
```

Por último, a veces se ha querido **pasar una serie de bits** (con valores de *0* o *1*) guardados en una variable *String* se conviertan en un **número decima**l y se guarden en una variable *Integer*. Para eso se ha creado la siguiente función (tampoco se puede hacer **cálculos exponenciales** por lo que dentro de esta función también se ha tenido que buscar una manera de hacer estas operaciones matemáticas):

```
function Binary_to_Decimal(bits$)
        dim dec, len, b1, b2, a, b, c0$, rc3
        len = StrLen(bits$)
        dec = 0
        for a = 1 to len
                c0$ = mid$(bits$,(a-1),1)
                rc3 = ExtractIntToken(c0$,b1)
                b2 = 1
                if (len-a)!= 0 then
                        for b=1 to (len-a) //Equivalent
                           to "2^(len-a)"
                                b2 = b2*2
                        next
                endif
                dec = dec + b1*b2
        next
endfunc dec
```

Parece que se ha complicado la programación en excesivo inútilmente, pero la realidad es que el lenguaje **SmartBASIC no ofrece una manera más sencilla de realizar estas funciones**. Se han creado estas funciones generales para **facilitar** el futuro desarrollo del programa software.

Apéndice B

Cálculos de consumo energético

Resumen: En este apéndice se muestran los cálculos realizados para obtener el consumo de energía estimado según el Data Rate y Spreading Factor usados.

Como se ha visto en las pruebas realizadas en el *Capítulo 3*, se ha visto que contra más alto sea el *Data Rate* usado más paquetes se pueden mandar y además consumiendo menos energía por cada paquete enviado. Sin embargo, como es lógico, al mandarse mas paquetes se consume más energía total.

Se va a calcular la energía consumida al mandar un único *paquete BLE* en cada transmisión, para así poder comparar la energía que se consume al **mandar la misma cantidad de datos** con distinto DR.

Por un lado, se sabe que en **2 minutos** pueden mandar este número de paquetes por cada DR:

- **DR** = **0**: 7.6 paquetes/transmisiones
- **DR** = **1**: 13.8 paquetes/transmisiones
- **DR** = **2**: 24.6 paquetes/transmisiones
- **DR** = **3**: 42.4 paquetes/transmisiones
- **DR** = **4**: 57 paquetes/transmisiones
- **DR** = **5**: 60.8 paquetes/transmisiones

Sin embargo, para poder comparar los resultados mejor, se van a hacer pruebas haciendo solo **7 retransmisiones** por cada DR (las máximas retransmisiones que se es capaz de hacer con *DR0*).

Por otro lado, se saben los siguientes consumos en los distintos periodos de tiempo:

- **Ventana de transmisión**: 40.59 mA
- **Ventana de recepción**: 25.14 mA
- **Resto de tiempo**: 14.9 mA

Recordar que al mencionar el consumo de las dos ventanas se añade también el consumido al escanear *paquetes BLE*. El tiempo restante donde las dos ventanas están cerradas solo se consume energía por ésto último, por estar escaneando *paquetes BLE*.

Por tanto, en primer lugar se calcula el tiempo que está en cada uno de los tres periodos de tiempo mencionados por cada *DR*, después se multiplica ese intervalo por lo que se consume ahí, luego se suma todo y para finalizar se divide por 120 para así saber los **miliamperios que se atraviesan en 1 segundo**:

- **DR = 0**
  - Ventana de transmisión:

$$intervalo = 2300(ms) \times 7(transmisiones) = 16100ms$$

  - Ventana de recepción:

$$intervalo = 1190(ms) \times 7(transmisiones) = 8330ms$$

  - Resto de tiempo:

$$intervalo = 120000(ms) - 16100(ms) - 8330(ms) = 95570ms$$

  - **Energía consumida**:

$$Consumo = \frac{16{,}1(s) \times 40{,}59(mA) + 8{,}33(s) \times 25{,}14(mA) + 95{,}57(s) \times 14{,}9(mA)}{120} = 19{,}06mAs$$

- **DR = 1**
  - Ventana de transmisión:

$$intervalo = 1233(ms) \times 7(transmisiones) = 8631ms$$

  - Ventana de recepción:

$$intervalo = 592(ms) \times 7(transmisiones) = 4144ms$$

- Resto de tiempo:

$$intervalo = 120000(ms) - 8631(ms) - 4144(ms) = 107225ms$$

- **Energía consumida**:

$$Consumo = \frac{8{,}631(s) \times 40{,}59(mA) + 4{,}144(s) \times 25{,}14(mA) + 107{,}225(s) \times 14{,}9(mA)}{120} = 17{,}1mAs$$

- **DR = 2**

  - Ventana de transmisión:

$$intervalo = 575(ms) \times 7(transmisiones) = 4025ms$$

  - Ventana de recepción:

$$intervalo = 298(ms) \times 7(transmisiones) = 2086ms$$

  - Resto de tiempo:

$$intervalo = 120000(ms) - 4025(ms) - 2086(ms) = 113889m$$

  - **Energía consumida**:

$$Consumo = \frac{4{,}025(s) \times 40{,}59(mA) + 2{,}086(s) \times 25{,}14(mA) + 113{,}889(s) \times 14{,}9(mA)}{120} = 15{,}94mAs$$

- **DR = 3**

  - Ventana de transmisión:

$$intervalo = 308(ms) \times 7(transmisiones) = 2156ms$$

  - Ventana de recepción:

$$intervalo = 151(ms) \times 7(transmisiones) = 1057ms$$

  - Resto de tiempo:

$$intervalo = 120000(ms) - 2156(ms) - 1057(ms) = 116787ms$$

  - **Energía consumida**:

$$Consumo = \frac{2{,}156(s) \times 40{,}59(mA) + 1{,}057(s) \times 25{,}14(mA) + 116{,}787(s) \times 14{,}9(mA)}{120} = 15{,}45mAs$$

- **DR = 4**

  - Ventana de transmisión:

$$intervalo = 172(ms) \times 7(transmisiones) = 1204ms$$

- Ventana de recepción:

$$intervalo = 86(ms) \times 7(transmisiones) = 602ms$$

- Resto de tiempo:

$$intervalo = 120000(ms) - 1204(ms) - 602(ms) = 118194$$

- **Energía consumida**:

$$Consumo = \frac{1{,}204(s) \times 40{,}59(mA) + 0{,}602(s) \times 25{,}14(mA) + 118{,}194(s) \times 14{,}9(mA)}{120} = 15{,}21mAs$$

- **DR = 5**
  - Ventana de transmisión:

$$intervalo = 90(ms) \times 7(transmisiones) = 630ms$$

  - Ventana de recepción:

$$intervalo = 42(ms) \times 7(transmisiones) = 294ms$$

  - Resto de tiempo:

$$intervalo = 120000(ms) - 8740(ms) - 452{,}2(ms) = 119076ms$$

  - **Energía consumida**:

$$Consumo = \frac{0{,}630(s) \times 40{,}59(mA) + 0{,}294(s) \times 25{,}14(mA) + 119{,}076(s) \times 14{,}9(mA)}{120} = 15{,}06mAs$$

Por tanto, si se mandan 7 paquetes *BLE* en 2 minutos (1 en cada transmisión *LoRa*) los resultados son los siguientes:

- **DR = 0**: 19.06 mAs
- **DR = 1**: 17.1 mAs
- **DR = 2**: 15.94 mAs
- **DR = 3**: 15.45 mAs
- **DR = 4**: 15.21 mAs
- **DR = 5**: 15.06 mAs

Bibliografía

[1] Robin Heydon. *Bluetooth Low Energy*. 2012.

[2] Engineea. *Protocolo entre el Beacon BT y el gateway*.

[3] Argenox. *A BLE Advertising Primer*. http://www.argenox.com/a-ble-advertising-primer/

[4] Engineea. *Protocolo entre el Beacon BT y el gateway*.

[5] Semtech. *LoRa*. http://www.semtech.com/wireless-rf/internet-of-things/what-is-lora

[6] LoRa Alliance. *LoRaWAN Specification*. https://www.lora-alliance.org/portals/0/specs/LoRaWAN%20Specification%201R0.pdf

[7] The Things Network. *LoRaWAN*. https://www.thethingsnetwork.org/wiki/LoRaWAN/Home

[8] LoRa Alliance. *LoRa Alliance Technology*. https://www.lora-alliance.org/What-Is-LoRa/Technology

[9] Andrew Back. *A Closer Look at LoRaWAN and The Things Network*. https://www.rs-online.com/designspark/a-closer-look-at-lorawan-and-the-things-network

[10] Atim. *What is LoRaWAN?*. http://www.atim.com/en/technologies-2/lorawan

[11] MWR Labs. *LoRa Security*. https://labs.mwrinfosecurity.com/assets/BlogFiles/mwri-LoRa-security-guide-1.2-2016-03-22.pdf

[12] Laird. *RM1xx Series, Development Kit*. https://assets.lairdtech.com/home/brandworld/files/User%20Guide%20-%20RM1xx%20Development%20Kit.pdf

[13] Laird. *RM1xx Series, Loading and Running Applications with UwTerminalX*. https://assets.lairdtech.com/home/brandworld/files/Loading%20and%20Running%20Applications%20with%20UwTerminalX%20-%20RM1xx%20Series.pdf

[14] Laird. *SmartBASIC, Core Functionality.* https://assets.lairdtech.com/home/brandworld/files/User%20Guide%20-%20smartBASIC%20Core%20Functionality.pdf

[15] Laird. *SmartBASIC Extensions, BLE Peripheral Functions.* https://assets.lairdtech.com/home/brandworld/files/User%20Guide%20-%20RM1xx%20smartBASIC%20Extensions%20(BLE%20Peripheral%20Functions).pdf

[16] Laird. *SmartBASIC Extensions, Lora Functions.* https://assets.lairdtech.com/home/brandworld/files/User%20Guide%20-%20RM1xx%20smartBASIC%20Extensions%20(LoRa%20Functions).pdf

[17] Laird. *RM1xx Series, Using BLE and LoRa.* https://assets.lairdtech.com/home/brandworld/files/Using%20BLE%20and%20LoRa%20-%20RM1xx%20Series.pdf

[18] Laird. *RM186, Interfacing with LoRaWAN.* https://assets.lairdtech.com/home/brandworld/files/Interfacing%20with%20LoRaWAN%20-%20RM186.pdf

[19] Laird. *RM1xx Series, Connecting to a MultiTech® Conduit Gateway.* https://assets.lairdtech.com/home/brandworld/files/Connecting%20to%20a%20Multitech%20Conduit%20Gateway%20-%20RM1xx%20Series.pdf

[20] Laird. *RM1xx Series, LoRaWAN Keys and IDs Overview.* https://assets.lairdtech.com/home/brandworld/files/LoRaWAN%20Keys%20and%20IDs%20Overview.pdf