

UNIVERSIDAD POLITÉCNICA DE MADRID

**ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN**

**MÁSTER UNIVERSITARIO EN
INGENIERÍA DE SISTEMAS ELECTRÓNICOS**

TRABAJO FIN DE MÁSTER

**Desarrollo de un protocolo de comunicación
Bluetooth para uso y gestión de una red de Internet de
las Cosas con pantallas de tinta electrónica**

Salvador Fco Criado Melero

Madrid, 2018

Desarrollo de un protocolo de comunicación Bluetooth para uso y gestión de una red de internet de las cosas(IoT) con pantallas E-INK

Salvador Fco Criado Melero

Palabras clave: internet de las cosas (IoT), *bluetooth low energy*, pantallas E-Ink

Resumen: En este trabajo se muestra el proceso de diseño de una aplicación de Internet de las cosas (*IoT*), que se encargará de gestionar una red de pantallas de ultra-bajo consumo. El objetivo principal será poder seleccionar el contenido de éstas pantallas desde un nodo central, optimizando al máximo el consumo de energía.

El principal reto al que nos enfrentamos será el envío de “grandes” cantidades de datos a través de *Bluetooth Low Energy*. En esta red se distinguirán dos roles principales, los nodos centrales, encargados de gestionar la red, y los nodos periféricos, que albergarán las pantallas. Para las pantallas trabajaremos con la tecnología *E-Ink*.

Development of Bluetooth Low Energy protocol to control an Internet of Things (IoT) newtwork with E-INK screens

Salvador Fco Criado Melero

Key words: Internet of Things (IoT), *bluetooth low energy*, low energy, E-Ink Screens.

Summary: In this Project we show the process of development of a IoT application which will be responsible of control an ultra-low energy screens network. The principal aim is to select the content that shows those screens from a central node, optimizing to the maximum the energy consumption.

The main challenge of this project is the data sending of “big” amount of data with the Bluetooth Low Energy protocol.

We can identify two principal's roles on the network, the centrals nodes, which will control the network, and the peripherals nodes, which will hold the screens. For the development of these screens we will work with W-Ink technology.

A el universo, por toda su energía,

A Diego y mis compañeros de departamento, por haber formado parte activa de este proyecto,

A mis compañeros de Master y a Julián, por acompañarme en el camino,

A todos los profesores que nos han aportado tanto este año,

Gracias.

El hombre razonable se adapta al mundo; el irrazonable intenta adaptar el mundo a sí mismo. Así pues, el progreso depende del irrazonable. –

George Bernard Shaw

ÍNDICE DE FIGURAS	12
ÍNDICE DE TABLAS	15
1. INTRODUCCIÓN.....	16
1.1. MOTIVACIÓN Y OBJETIVOS	17
1.2. PLANIFICACIÓN	19
1.3. ESTRUCTURA DEL DOCUMENTO	20
2. ESTADO DEL ARTE.....	23
2.1. REDES IoT	24
2.2. PANTALLAS E-INK.....	27
2.3. BLUETOOTH LOW ENERGY (BLE).....	29
2.3.1. <i>BLE Stack</i>	30
2.3.2. <i>Capa física</i>	31
2.3.3. <i>Roles en las comunicaciones BLE</i>	32
2.3.4. <i>Protocolo de descubrimiento</i>	33
2.3.4.1. Mensajes en un descubrimiento de dispositivos BLE.....	33
2.3.4.2. GAP	34
2.3.5. <i>Protocolos de conexión</i>	35
2.3.5.1. Roles en una conexión BLE	36
2.3.5.2. Base de datos GATT	36
2.3.5.3. Tipo de peticiones GATT.....	38
2.3.5.4. Conexiones GATT.....	39
2.3.6. <i>Servicios y características</i>	41
3. SELECCIÓN DEL PROTOCOLO.....	44
3.1. SELECCIÓN DE PARÁMETROS BLE	45
3.1.1. <i>Parámetros y roles de descubrimiento BLE</i>	45
3.1.2. <i>Parámetros y roles de conexión BLE</i>	46
3.1.3. <i>Parámetros seleccionados</i>	47
3.2. PERFIL BLE	48
3.2.1. <i>Servicios por defecto</i>	48
3.2.1.1. Generic Access.....	48
3.2.1.2. Device information	49
3.2.2. <i>SPP Service</i>	50
3.2.2.1. Características del servicio SPP	51

3.2.3. <i>Almacenamiento imagen</i>	52
3.2.4. <i>Tamaño de paquete</i>	53
4. DISEÑO DE HARDWARE.....	56
4.1. PERVERSIVE DISPLAY	57
4.2. ATIK-020	60
4.3. ARTIK 520.....	61
4.4. SNIFFER NRF51822	63
5. IMPLEMENTACIÓN DEL SISTEMA	66
5.1. NODOS PERIFÉRICOS	67
5.1.1. <i>Serial debug</i>	68
5.1.2. <i>Actualización de pantalla</i>	69
5.1.3. <i>Gestión de eventos</i>	72
5.1.4. <i>Modos Energía del sistema</i>	74
5.1.5. <i>Planificación tareas</i>	75
5.2. NODOS CENTRALES	76
5.2.1. <i>Búsqueda de dispositivos</i>	77
5.2.2. <i>Envío de imagen</i>	80
6. RESULTADOS Y CONCLUSIONES	83
6.1. RESULTADOS	83
6.1.1. <i>Análisis de consumo</i>	84
6.1.1.1. Consumo en modo Advertising	85
6.1.1.2. Consumo en modo de actualización de pantalla.....	88
6.1.1.3. Consumo en situación real	89
6.1.2. <i>Tiempo de envío de imagen</i>	91
6.2. PRESUPUESTO	94
6.3. CONCLUSIONES Y TRABAJO FUTURO	95
REFERENCIAS	98

ÍNDICE DE FIGURAS

Figura 1 IoT	17
Figura 2 Ejemplo ilustrativo del nodo periférico	18
Figura 3 Ejemplo ilustrativo nodo central	18
Figura 4 Requisitos principales de diseño	19
Figura 5 Planificación del proyecto	20
Figura 6 IoT	23
Figura 7 Logo Eink	24
Figura 8 Bluetooth Low Energy	24
Figura 9 Internet of Things (IoT).....	25
Figura 10 Evolución de IoT	26
Figura 11 Principales retos IoT	27
Figura 12 E Ink Display.....	28
Figura 13 Tecnología E Ink	29
Figura 14 Bluetooth Low Energy	29
Figura 15 Capas del protocolo Bluetooth 4.0 [13].....	30
Figura 16 Canales utilizados por BLE [13]	31
Figura 17 Roles protocolo BLE [15]	32
Figura 18 Proceso de descubrimiento Bluetooth [13].....	34
Figura 19 Roles en el protocolo GAP [17]	35
Figura 20 Consumo de potencia típico periférico advertisement mode	35
Figura 21 Roles en una conexión BLE [17].....	36
Figura 22 Jerarquía de base de datos GATT	38
Figura 23 Proceso Write y Write_without_reponse.....	39
Figura 24 Intercambio de mensajes GATT [13]	40
Figura 25 Consumo de potencia en estado de conexión [17]	40
Figura 26 Slave Latency ilustración [17].....	41
Figura 27 Objetos de transacción GATT [13]	42

Figura 28 Esquema de la red IoT	44
Figura 29 Resumen conexión BLE	48
Figura 30 Protocolo envío imagen	51
Figura 31 Struct de imagen	52
Figura 32 Tamaño de paquetes de imagen.....	53
Figura 33 Envío de paquetes write without reponse	54
Figura 34 Eink Pantallas	56
Figura 35 Samsung ARTIK	57
Figura 36 Pantalla E Ink Pervasive Display [23].....	58
Figura 37 Dimensiones E-Ink Pervasive Display	58
Figura 38 Resolución pantalla E Ink [23]	59
Figura 39 Driver EPD pantalla [23].....	59
Figura 40 Extensio Board frontal.....	60
Figura 41 Extension Board reverso.....	60
Figura 42 ATIK 020.....	61
Figura 43 Artik 520 Module [25].....	62
Figura 44 Funcionalidades ARTIK 520 [25]	63
Figura 45 nRF518822	64
Figura 46 Nordic nRF	64
Figura 47 Ejemplo sniffer wireshark	65
Figura 48 Simplicity Studio	67
Figura 49 Energy profiler Simplicity Studio	68
Figura 50 Ejemplo de captura serial	69
Figura 51 PDI Apps Software.....	70
Figura 52 Cableado pantalla	71
Figura 53Máquina de estados ARTIK020	73
Figura 54 Planificación de tareas periférico	75
Figura 55 ARTIK IDE	76
Figura 56 FlowChart Busqueda BLE.....	78
Figura 57 Formato salida escaneo dispositivos	79

Figura 58 Búsqueda dispositivos ARTIK 520	80
Figura 59 FlowChart Envío imagen.....	81
Figura 60 Batería de alimentación	85
Figura 61 Autonomía en función de la potencia.....	87
Figura 62 Consumo para transmisión de 0 dBm.....	87
Figura 63 Consumo para transmisión de 10,5 dBm.....	88
Figura 64 Envío y actualización pantalla.....	89
Figura 65 Consumo medio diario	90
Figura 66 Señal transmisión de imagen	92
Figura 67 Captura wireshark intervalo de conexión	93
Figura 68 Señales entre dos paquetes	93
Figura 69 Seguridad IoT	96
Figura 70 Distintos tamaños de pantalla	97
Figura 71 Packaging E-Ink	97

ÍNDICE DE TABLAS

Tabla 1 Parámetros de advertising	46
Tabla 2 Selección de parámetros conexión.....	47
Tabla 3 Generic Access Service	49
Tabla 4 Direcciones UUID Servicio	52
Tabla 5 Consumo de potencia modo advertising en funcion de TX power.....	86
Tabla 6 Presupuesto modelo comercial	94
Tabla 7 Presupuesto de diseño y desarrollo	94

1. Introducción

Este proyecto está englobado en el proyecto INPAINK (infraestructura para presentación de información visual de bajo consumo y mantenimiento basado en pantallas de tintas electrónica), con código RTC-2016-4881-7, llevado a cabo en el departamento de Ingeniería de Sistemas Telemáticos de la Universidad Politécnica de Madrid. Surge como alternativa para facilitar la muestra de información visual de manera más automatizada, energéticamente eficiente y centralizada.

El documento que tiene en sus manos mostrará el diseño e implementación de una aplicación IoT que se encargará de gestionar una red de pantallas desde un punto central.

A sí mismo, las decisiones de diseño se tomarán en base a la optimización del consumo, ya que se pretende dotar de la mayor autonomía posible a la red. Para ello, utilizaremos principalmente los conceptos internet de las cosas (*IoT*) y *bluetooth low Energy (BLE)*.

Al final del documento mostraremos los resultados y conclusiones derivados de este proceso de diseño.



Figura 1 IoT

1.1. Motivación y objetivos

El objetivo principal de este Trabajo de Fin de Master es el desarrollo del protocolo de comunicación *BLE* dentro de la red *IoT*, cuyo objetivo principal será la gestión una red de pantallas desde un nodo central, de manera que podamos seleccionar la imagen que queremos mostrar. Dentro de esta red BLE, se distinguen dos roles principales:

- Nodo periférico: será el dispositivo que contenga la pantalla. Se pretende dotar a este dispositivo de la mayor autonomía posible, por lo que se tendrá muy en cuenta el consumo a la hora de diseñarlos. En la Figura 2 se muestra un ejemplo ilustrativo de lo que podría ser un nodo periférico.

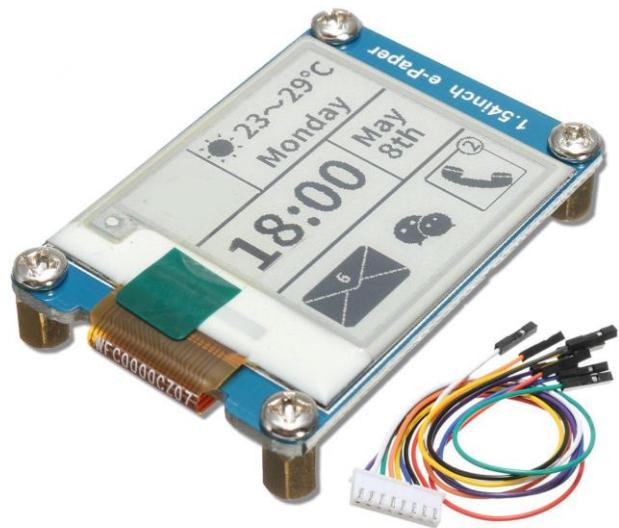


Figura 2 Ejemplo ilustrativo del nodo periférico

- Nodo central: este nodo servirá de enlace con el servidor web que se encargará de gestionar la red. Sus funciones serán sostener el servidor, descubrir y conectarse a dispositivos BLE y enviar imágenes a éstos. En la Figura 3 se muestra otro ejemplo ilustrativo de lo que podría ser un nodo central.



Figura 3 Ejemplo ilustrativo nodo central

El reto más grande al que nos enfrentamos es el envío de grandes cantidades de datos a través del protocolo *bluetooth low energy*, ya que en un principio no está diseñado para esta finalidad.

Además, será de vital importancia durante todo el diseño cuidar el aspecto del consumo, ya que se pretende dotar de la mayor autonomía posible a nuestra red.

Por lo que podemos resumir en dos nuestros requisitos de diseño principales: tiempo de envío reducido y consumo mínimo.



Figura 4 Requisitos principales de diseño

1.2. Planificación

Este proyecto se inicia el 15 de diciembre de 2017 con el objetivo de finalizarse para Julio de 2018. Para ello hemos realizado la planificación temporal que se muestra en la Figura 5.

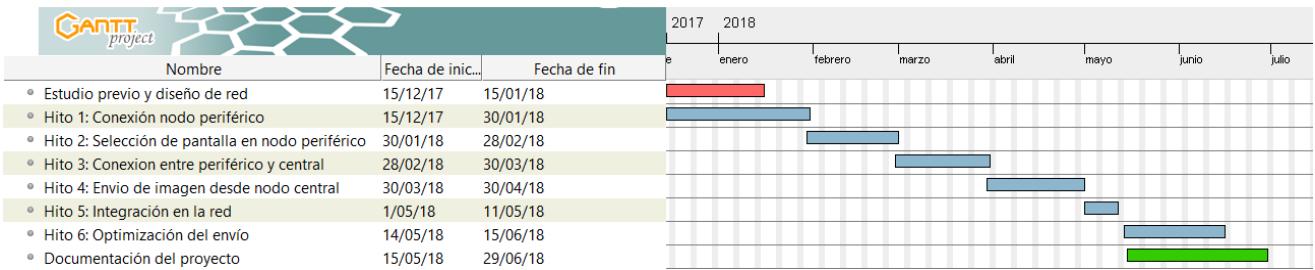


Figura 5 Planificación del proyecto

La primera fase se centrará en el estudio previo de la tecnología a utilizar y la determinación del diseño de la red. Es lo que correspondería a la fase de análisis y diseño.

Para la fase de implementación dividiremos el proyecto en seis hitos principales, de manera que nos permitan avanzar paso a paso. A continuación, se describe cada uno de ellos:

- Hito 1: implementación del protocolo BLE en el dispositivo periférico, de manera que podamos conectarnos a él desde fuera.
- Hito 2: selección de imágenes en el dispositivo periférico desde un dispositivo externo. Estas imágenes a mostrar estarán almacenadas dentro del nodo periférico.
- Hito 3: implementación del protocolo BLE en el nodo central y conexión con el nodo periférico.
- Hito 4: actualización de la imagen del dispositivo periférico que será enviada desde el nodo central.
- Hito 5: integración con la red para poder gestionar esas imágenes desde el servidor web.
- Hito 6: ajustar y optimizar el tamaño de paquetes para reducir los tiempos de envío.

1.3. Estructura del documento

El documento constará de 6 partes principales:

- Introducción (sección 1): mostraremos la organización, motivación, planificación y objetivos principales del proyecto. Nos servirá para situarnos en el proyecto y poder orientarnos en el documento.
- Estado del arte (sección 2): estudiaremos el contexto en el que se encuentran las tecnologías que vamos a utilizar para nuestros proyecto, que serán principalmente tres: redes *IoT* (sección 2.1), pantallas E-INK (sección 2.2) y *bluetooth low energy* (sección 2.3). Nos aportaran una visión general del estado actual de estas tecnologías.
- Diseño de protocolo (sección 3): en este apartado nos centraremos en el desarrollo del protocolo *bluetooth low energy* e iremos justificando las decisiones de diseño que hemos ido tomando.
- Diseño de *hardware* (sección 4):haremos un estudio y justificación de la elección del *hardware* que nos va a permitir implementar nuestra red. Se centrará en tres puntos: pantallas (sección 4.1), nodos periféricos (sección 4.2) y nodos centrales (sección 4.3).
- Implementación (sección 5): mostraremos el proceso de implementación que se ha ido llevando a cabo y las herramientas que hemos utilizado. Esta sección estará a su vez dividida en dos: implementación de nodos periféricos, que a su vez incluye la implementación de las pantallas (sección 5.1), e implementación de nodos centrales (sección 5.2)
- Resultados y conclusiones (sección 6): en esta sección analizaremos los resultados obtenidos y el grado de cumplimiento de las expectativas (sección 6.1), analizando el consumo y los tiempos de envío. También llevaremos a cabo un estudio económico (sección 6.2) y cerraremos el proyecto con las conclusiones y trabajo futuro (sección 6.3).

2. Estado del arte

En esta sección faremos un análisis del estado y la proyección de las tecnologías que utilizamos para el desarrollo de nuestro proyecto, de manera que nos permitan situarnos en el contexto tecnológico.

Centraremos este análisis en tres tecnologías o conceptos: *IoT* (sección 2.1), pantallas E-Ink (sección 2.2) y *bluetooth low energy* (sección 2.3).

IoT es un concepto que se refiere a la interconexión digital de objetos cotidianos a internet [1]. Surge con la necesidad de facilitar la conexión y la gestión de los dispositivos electrónicos, motivado por el crecimiento exponencial de éstos en los últimos años. Este crecimiento, y su integración en cada vez más aplicaciones diarias, hace que sea una buena elección para nuestro proyecto.

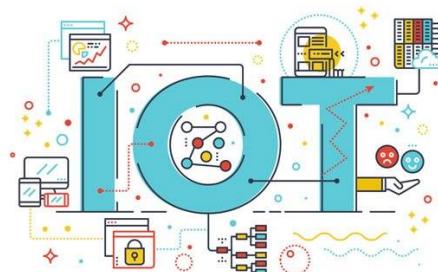


Figura 6 IoT

Las pantallas de tinta electrónica o *E-Ink* es una tecnología que permiten simular la sensación de un papel físico, ofreciendo un consumo energético muy reducido. Esto hace que sea la mejor opción para nuestra aplicación frente otras tecnologías como podrían haber sido LCD, TFT o cristal líquido.



Figura 7 Logo Eink

Bluetooth Low Energy (BLE). Es un protocolo desarrollado por *Bluetooth* [2] que ofrece comunicación entre dispositivos móviles, o computadores, y otros dispositivos más pequeños, basándose en la optimización de energía. Se posiciona como una de las tecnologías *Wireless* de corto alcance más indicadas para aplicaciones *IoT*, por lo que ha sido nuestra elección frente a otras como *Zigbee*.



Figura 8 Bluetooth Low Energy

2.1. Redes IoT

El Internet de las cosas (IoT), como hemos mencionado anteriormente, es un concepto que se utiliza principalmente para la interconexión digital de objetos cotidianos a internet. Fue lanzado por primera vez por Kevin Ashton, investigador de la MIT, en 1999, donde se realizaban investigaciones en el campo de la identificación por radiofrecuencia en red (RFID) y tecnologías de sensores. [3].

En un principio se consideró “Internet de las Cosas” como una simple extensión de la identificación por radiofrecuencia de los objetos. Pero hoy día se considera como una revolución que surgió en el siglo XIX, con la creación de las máquinas. Que se desarrolló durante el siglo XX, con la capacidad de éstas máquinas de ejecutar comandos. Y que finalmente, en el siglo XXI, están aprendiendo a pensar, de manera que pueden anticiparse y percibir lo que las personas desean. [3]

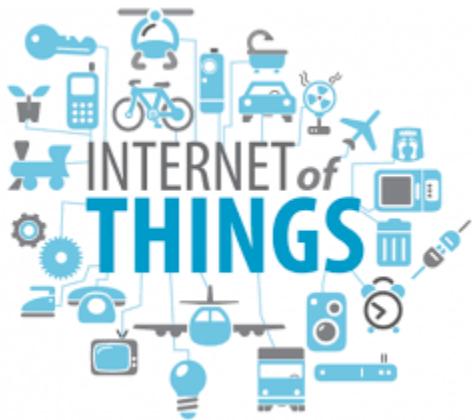


Figura 9 Internet of Things (IoT)

Todo ello llevará a cambios organizativos y sociales que impulsarán nuevas oportunidades económicas, sociales y tecnológicas. Está reconocida como una de las áreas de futuro más influyentes en el sector tecnológico y está acaparando la atención de muchos sectores industriales. [4]

El desarrollo de éste concepto permitirá modernizar la manera en que hacemos y concebimos las actividades diarias en todos los sectores, ofreciéndonos un mayor control y eficacia. Esto se conseguirá a partir del aumento de las conexiones entre las personas, los procesos, los datos y las cosas.

Según *Internet Business Solutions Group* de Cisco (IBSG) se predice que unos 50 mil millones de dispositivos estarán interconectados en el 2020.(Figura 10) [5].

Según la empresa Gartner [6], en 2020 habrá en el mundo aproximadamente 26 mil millones de dispositivos con un sistema de conexión a internet de la cosas. [7]

Aunque haya algunas diferencias en las estimaciones, en lo que no hay duda es en que están enfocadas a ocupar una posición de las más importantes en el mercado de las tecnologías, si no la

ocupan en la actualidad. Según George Osborne, ex miembro del gabinete en la sombra [8], *IoT* es la próxima etapa en la revolución de la información, haciendo referencia a la interconectividad de todo, desde el transporte urbano hasta dispositivos médicos, pasando por electrodomésticos. [7]

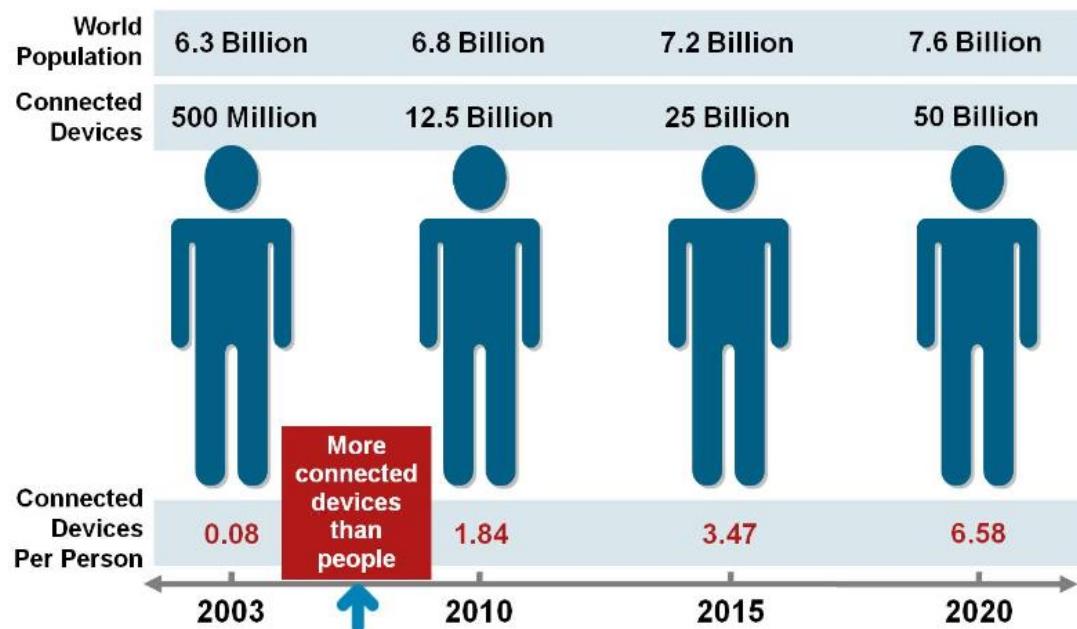


Figura 10 Evolución de IoT

Las aplicaciones en las que se utiliza esta interconexión es muy amplia, dirigiéndose a tres ámbitos principales: consumidores, empresarial e infraestructura.

Algunas de las aplicaciones de consumo más destacables podrían ser interconexión de automóviles, entretenimiento, automatización del hogar o domótica, tecnología vestible...

En el entorno empresarial surge una extensión del concepto conocido como *EIoT* (*Enterprise Internet of Things*) [9]. Se aplica a hospitales o entornos industriales en los que se produce la interconexión de dispositivos, para la localización, monitorización o actualización de información entre otras muchas aplicaciones.

También es realmente útil en el monitoreo y control de operaciones de infraestructura urbana y rural como puentes, vías férreas y parques eólicos.

Los principales retos a los que se enfrenta el *IoT* se pueden dividir en seguridad, encriptación, direccionamiento y computación (Figura 11). [9]

A causa de la gran cantidad de dispositivos que integraran este concepto y su integración e iteración con otros elementos de la red, hacen que la seguridad sea un pilar fundamental a tener en cuenta, ya que podrían comprometer la seguridad del resto de la red.

La transmisión de información sensible e interpretación de ésta, es una de los usos de este tipo de redes. Esto hace que sea especialmente importante el cifrado y encriptación de estos datos, de manera que no comprometan la privacidad del usuario.

Aludiendo nuevamente a la cantidad de dispositivos que se interconectan e iteración entre sí, plantea un nuevo reto en la integración, intercomunicación y direccionamiento entre dispositivos.

Se pretende dotar a estos dispositivos de capacidad de decisión y análisis de datos sin necesidad de terceros, por lo que la necesidad de capacidad de computo es comprensible.

Estos pilares deberán de tenerse en cuenta a la hora de desarrollar nuestra aplicación.

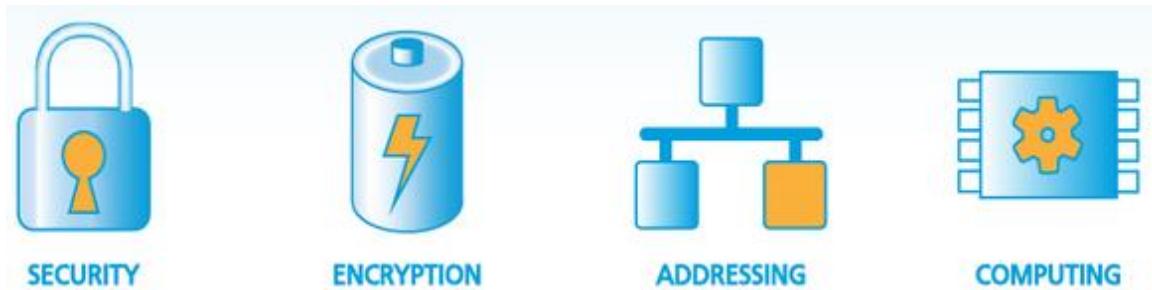


Figura 11 Principales retos IoT

2.2. Pantallas E-Ink

Las pantallas de tinta electrónica o *E-Ink* es una tecnología que permite crear pantallas planas que emulan la función de papel físico en apariencia. El consumo reducido que ofrece es una de las características que hace que sean una de las mejores opciones a la hora de desarrollar aplicaciones de visualización de datos.

Esta tecnología ha sido desarrollada por la compañía E INK [10], creada en abril de 1997 por los investigadores del Media Lab del MIT (Instituto de Tecnología de Massachusetts) [11].

Uno de los principales motivos de su creación es debido a razones ecologistas. El hecho de ser una alternativa al uso del papel, hacen que sea una opción medioambientalmente más responsable.



Figura 12 E Ink Display

El principal de esta tecnología son los libros electrónicos, aunque también se extiende a otros campos como *IoT*, *1D/2D barcode*, *Smart tags*, identificación y acceso...

Su funcionamiento está basado en biestables que harán que la imagen se mantenga en pantalla, sin necesidad de alimentación, por lo que solo se producirá un consumo de energía en las actualizaciones.

Están formadas por millones de micro capsulas como la que se muestra en la Figura 13. Cada micro capsula contiene partículas con carga positiva (blancas) y cargas negativas (negras) que se sumergen en un fluido. Cuando se le aplica un campo positivo, o negativo, las partículas cambian de posición.

En la Figura 13 podemos apreciar este proceso. En el primer caso, aplicamos un campo positivo, por lo que las partículas blancas irán a la superficie.

En el segundo caso aplicamos una diferencia de potencial, que nos permitirá regular la cantidad de partículas blancas o negras (controlando así los tonos de grises).

En el último caso vemos que las partículas negras ascenderán a la superficie al aplicar un campo negativo.

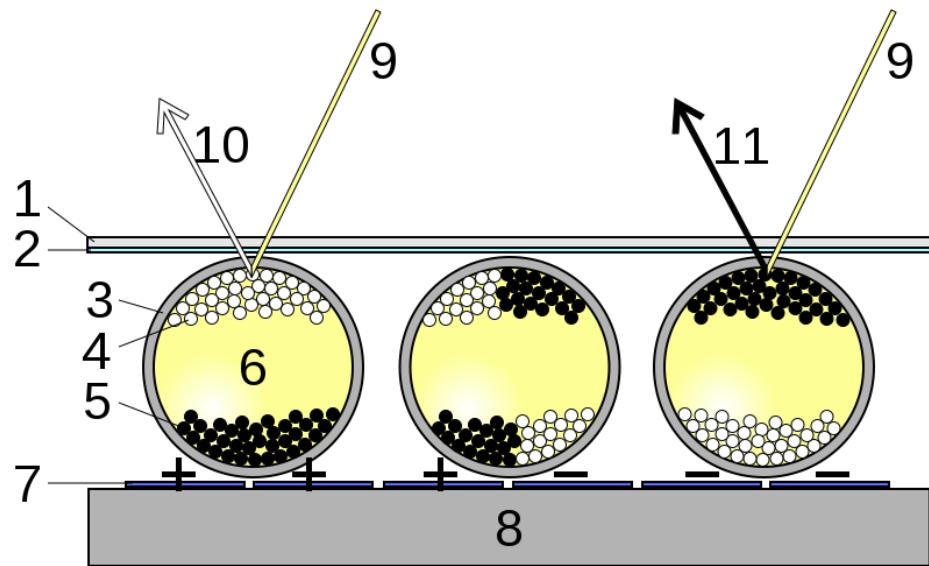


Figura 13 Tecnología E Ink

2.3. Bluetooth Low Energy (BLE)

Bluetooth Low Energy (BLE) es una tecnología *wireless* de bajo consumo desarrollada para el control y la monitorización en aplicaciones de corto alcance. Es una tecnología emergente desarrollada por Bluetooth Special Interes Group (SIG). Ha sido diseñada para cumplir las funciones del *bluetooth* tradicional con el consumo más bajo posible [12].

Se posiciona como una de las mejores alternativas a la gran cantidad de estándares *Wireless* existentes en el mercado tecnológico (IEEE 802.11b, ZigBee, ANT+, Clasic Bluetooth) [13]



Figura 14 Bluetooth Low Energy

El buen rendimiento que ofrece y la compatibilidad con la mayoría de dispositivos (PCs, tablets y smartphone) hacen que *BLE* sea un excelente candidato para una gran variedad de aplicaciones medicinales, domóticas, monitorización, seguridad, *IoT*... [13]

Como se ha mencionado, una de las características más destacables de esta tecnología es el ultra-bajo consumo en las transmisiones. Esto permite su implementación en dispositivos embebidos que requieran aplicaciones con tiempos de carga reducidos, o tiempos de duración de baterías elevados. [12].

Tiene soporte en la mayoría de plataformas recientes (IOS5+, Android 4.3+, Aplee OS x 10.6+, Windows 8+ y GNU/Linux Vanilla BlueZ 4.93+).

En los siguientes apartados profundizaremos en el funcionamiento de este protocolo y en los aspectos más importantes que se han de tener en cuenta a la hora de tomar decisiones de diseño.

2.3.1. BLE Stack

El protocolo BLE se puede dividir en tres bloques principales. El controlador, el host y la aplicación. En la Figura 15 se muestra este esquema. [14]

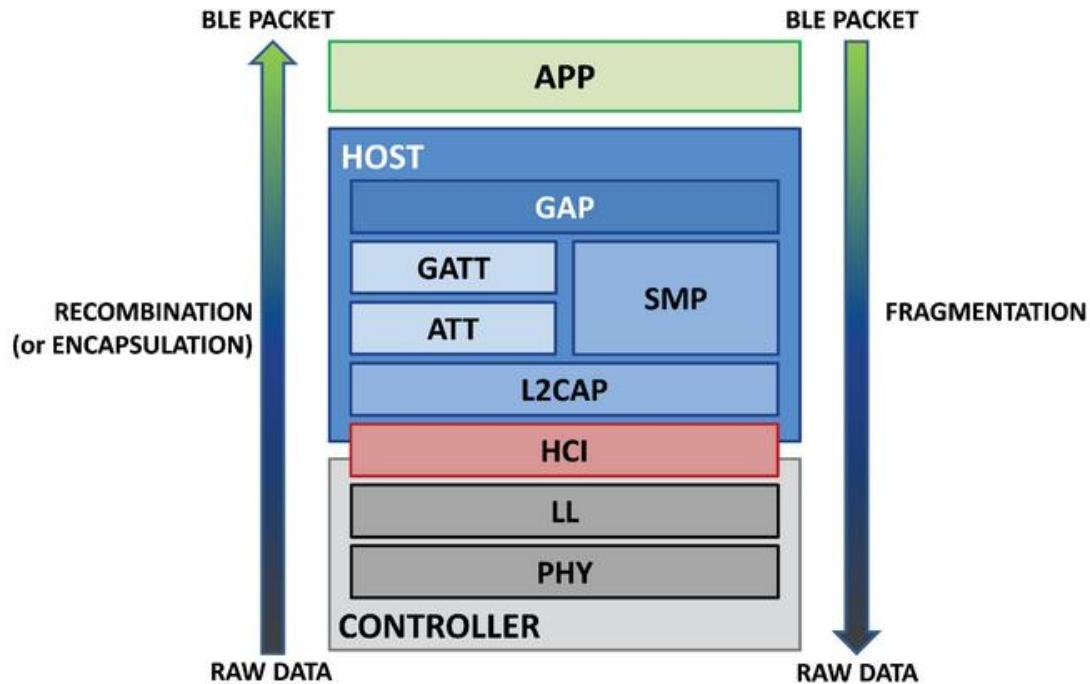


Figura 15 Capas del protocolo Bluetooth 4.0 [13]

La aplicación es el bloque más alto dentro de la *stack* y representa la interfaz con la que interactuará el usuario [13]. Esta interfaz se basa en el uso de perfiles BLE que permitirán la interoperación entre aplicaciones.

El controlador incluye la capa física y la capa de enlace. Normalmente se implementa con un *system-on-chip* (SOC), el cual incluye una antena de radiación.

El *host* se ejecuta en un procesador e implica las funcionalidades de capa superior

En los siguientes apartados vamos a hacer una descripción más detallada de las capas más importantes de éste protocolo.

2.3.2. Capa física

BLE opera en la frecuencia ISM (*Industrial Scientific Medical*) a 2,4 GHz y se definen 40 canales de radio en un ancho de banda de 2 MHz, 37 se utilizan para la transmisión de datos, y 3 para comunicaciones (Figura 16) [13].

Los canales de comunicación se utilizan para la búsqueda de dispositivos, establecer la conexión, o para la transmisión *broadcast*. Todos los canales utilizan la modulación GFSK (modulación por desplazamiento de frecuencia gaussiana).

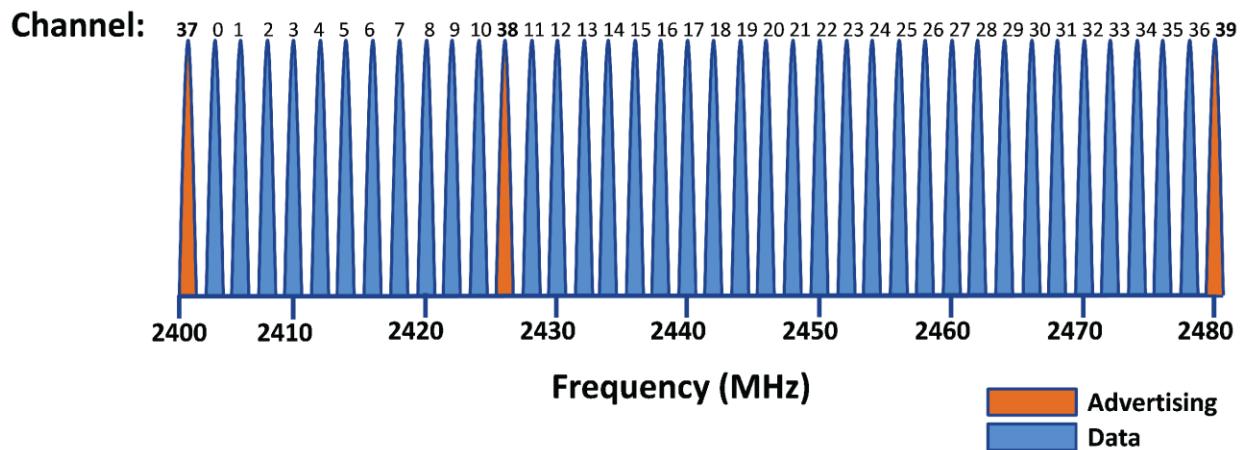


Figura 16 Canales utilizados por BLE [13]

2.3.3. Roles en las comunicaciones BLE

Como ya hemos adelantado en la Introducción, en la comunicación se identificarán dos roles principales:

- Equipos periféricos: normalmente pequeños, con bajo consumo de potencia y que suelen estar conectados a una central más poderosa (en nuestro caso serán los nodos de las pantallas).
- Equipos centrales: equipos con mayor capacidad de procesamiento, potencia y memoria. En nuestro caso será el dispositivo que utilizaremos para gestionar las pantallas.

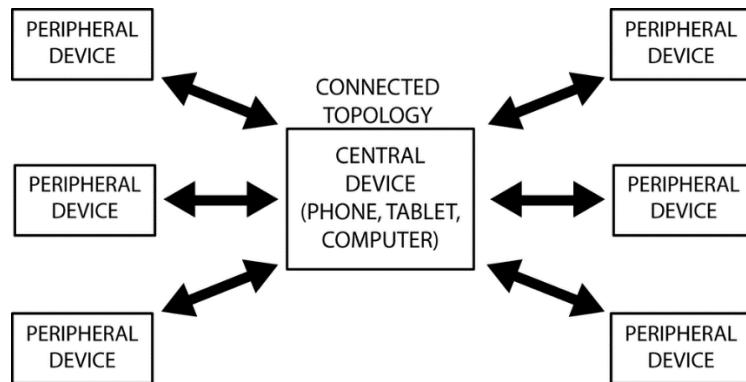


Figura 17 Roles protocolo BLE [15]

Los estados en los que estos dispositivos se puede encontrar son:

- *Standby*: modo de reposo.
- *Advertising*: visible para otros dispositivos.
- *Scanning*: modo de búsqueda de dispositivos.
- *Initiation*: iniciación de conexión.
- *Connected*: conectado con otro dispositivo.

Diferenciaremos dos etapas durante la comunicación, que se detallan en las siguientes secciones:

- Descubrimiento de dispositivos (sección 2.3.4).

- Conexión de dispositivos (sección 2.3.5).

2.3.4. Protocolo de descubrimiento

En este apartado vamos a describir cómo se lleva a cabo la conexión entre dispositivos, y las comunicaciones que se establecen entre éstos durante el proceso de descubrimiento. También definiremos los distintos roles que adquieren los dispositivos durante este proceso.

2.3.4.1. Mensajes en un descubrimiento de dispositivos BLE

La creación de esta comunicación es un proceso asimétrico en el que uno de ellos envía mensajes de que es un dispositivo disponible, y el otro, que estaría en escucha, recibe estos mensajes, enviando una petición de conexión. Hecho esto se crearía una conexión punto a punto entre los dos equipos.

En la Figura 18 ilustramos el proceso de descubrimiento y de envío de tramas entre dispositivos, en la que se distinguen los siguiente procesos:

- *Advertising Data*: el equipo periférico se encarga de enviar datos para notificar de que está disponible. El periodo con el que se envían estos datos es configurable en un rango de 20 ms hasta 10,28 s. [16].
- *Scan Reponse Request*: el equipo central puede descubrir que un periférico está disponible para la conexión cuando recibe los paquetes de *advertising*. Para iniciar la conexión, le enviará un *scan reponse request* con los parámetros de conexión. Estos parámetros podrán ser actualizados en cualquier momento durante la conexión. Se amplía la información en la sección 2.3.5.4.
- *Scan reponse data*: el equipo periférico le contestará indicando que los parámetros son válidos o sugiriendo otros.

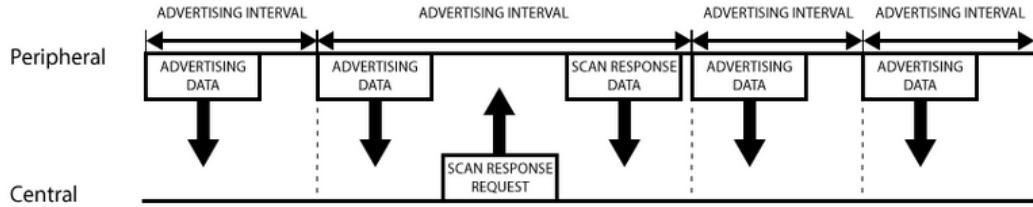


Figura 18 Proceso de descubrimiento Bluetooth [13]

2.3.4.2. GAP

Estas comunicaciones se engloban dentro de la capa de GAP. Esta capa determina el comportamiento de los equipos en las siguientes acciones [17]:

- Descubrimiento de dispositivos.
- Establecimiento de conexión.
- Finalización de conexión.
- Configuración del dispositivo.
- Comprobación de los requisitos de seguridad.

Los estados en los que un dispositivo puede encontrarse son:

- *Standby*: estado *reset* o cuando no está activado BLE.
- *Advertiser*: el equipo estará enviando paquetes que indicarán que se puede iniciar la conexión.
- *Scanner*: cuando el equipo envía una respuesta al dispositivo *advertiser*.
- *Initiator*: será el encargado de indicar los parámetros de la conexión.
- *Master*: el equipo que determina los parámetros de conexión se denominará *master*. Este será el *initiator*.
- *Slave*: el equipo que recibe los parámetros de conexión será el esclavo.

Para entender mejor estos estados, mostramos gráficamente el proceso en la Figura 19.

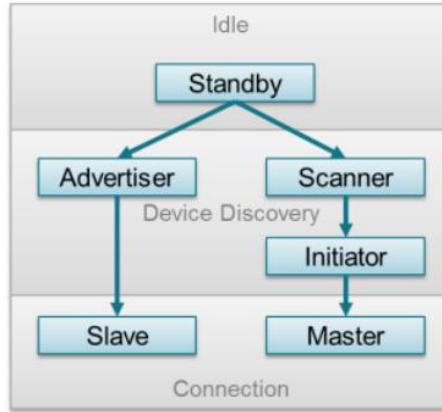


Figura 19 Roles en el protocolo GAP [17]

Entre envío de paquetes de *advertising*, los equipos periféricos, entrarán en modo *sleep* para reducir el consumo de energía. En la Figura 20 se muestra un consumo de potencia típico en modo de descubrimiento. La frecuencia de estos picos de consumo dependerá del tiempo entre paquetes que configuremos (20ms hasta 10,28s).

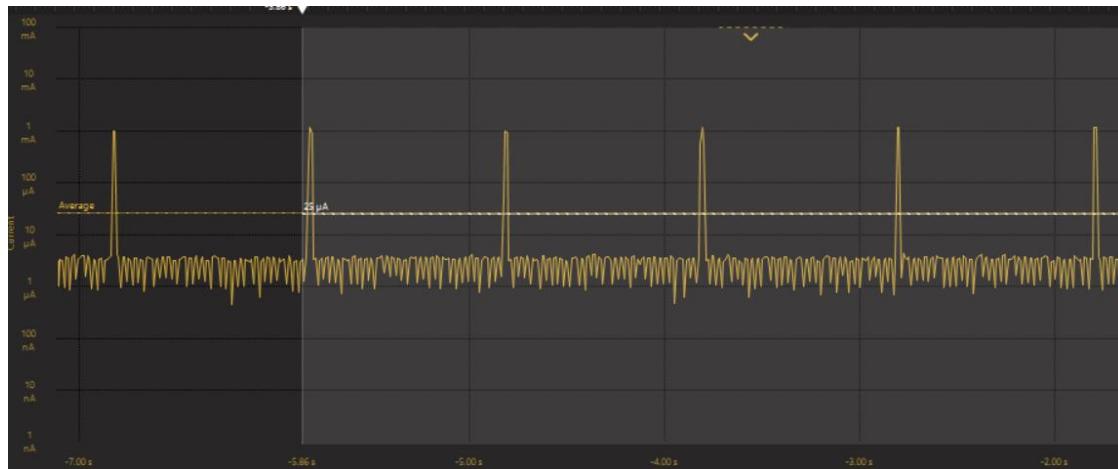


Figura 20 Consumo de potencia típico periférico advertisement mode

2.3.5. Protocolos de conexión

Una vez se establece la conexión, se producirá un envío de mensajes entre periférico y central con una frecuencia que se habrá determinado en el establecimiento de la conexión. En los siguientes apartados vamos a profundizar en este estado de la comunicación.

2.3.5.1. Roles en una conexión BLE

En una conexión de BLE, se establecen roles que determinarán la función principal de cada equipo dentro de la comunicación. Existen dos principales en función de quien almacene los datos:

- Servidor GATT: será el encargado de contener la tabla de servicios y características, que se describirán más adelante en la sección 2.3.6.
- Cliente GATT: se encargará de gestionar la tabla de servicios y características de forma remota, mediante peticiones al servidor.

En la Figura 21 se muestra gráficamente.

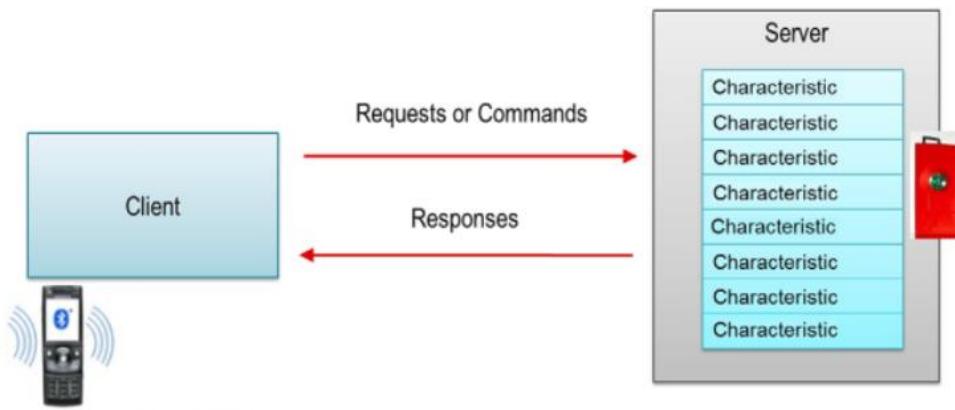


Figura 21 Roles en una conexión BLE [17]

2.3.5.2. Base de datos GATT

Para almacenar los datos en el servidor, se sigue una estructura que va desde la unidad más pequeña de datos (atributos) hasta la agrupación general de éstos (perfils). En la Figura 22 se muestra esta jerarquía.

Los atributos son los grupos básicos de información transferida entre dispositivos.

Una característica estará formada de atributos, y contendrá información sobre el valor del dato, propiedades y configuración. Una característica típica puede estar formada por los siguientes atributos:

- *Characteristic Value*: será el valor del dato que almacena la característica.
- *Characteristic Declaration*: será un descriptor que almacena las propiedades, localización y el tipo de característica.
- *Client Characteristic Configuration*: se utiliza para los procesos de *notify* e *indicate* que se explican en la sección 2.3.5.3.
- *Characteristic User Description*: será una cadena ASCII que ofrece una descripción de la característica.

El protocolo ATT (Attribute Protocol) se utiliza para almacenar los atributos en una tabla, usando IDs de 16-bis para cada entrada. No vamos a entrar en detalle de cómo se gestiona este protocolo.

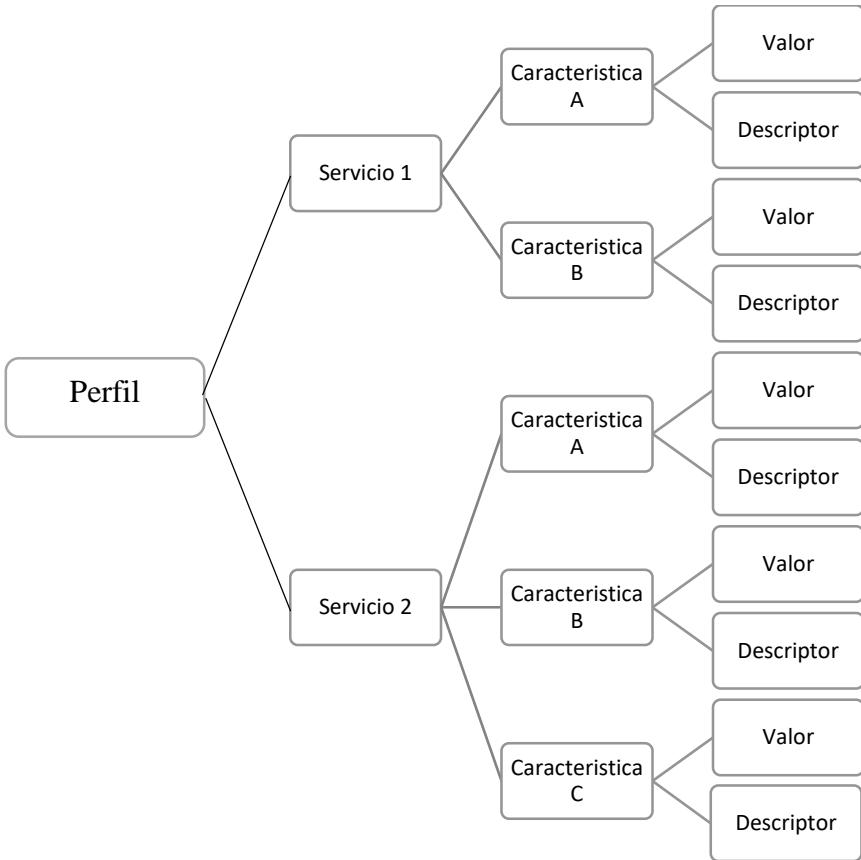


Figura 22 Jerarquía de base de datos GATT

2.3.5.3. Tipo de peticiones GATT

Para gestionar la base de datos GATT, el cliente, utiliza las siguientes peticiones:

- *Read*: solicita una petición de lectura al servidor GATT, de manera que éste devolverá el valor almacenado en ese campo.
- *Write*: se utiliza para modificar el valor de una característica del servidor GATT de forma remota.
- *Write without response*: la funcionalidad es la misma que la de *write*, pero en este caso no espera confirmación en cada escritura. En la Figura 23 se ilustra este proceso.
- *Notify*: es un modo de conexión que permite al servidor mandar notificaciones de manera permanente. El encargado de activar este modo de conexión será el cliente,

mediante la modificación de la característica *Client Characteristic Configuration Descriptor (CCCD)*.

- *Indicate*: realiza las mismas funciones que *notify* pero con notificación por ACK.

Estas peticiones tendrán que ser implementadas en el servidor GATT y variaran en función de las características y nuestros intereses. Se pueden configurar como obligatorias, opcionales o *excluded*.

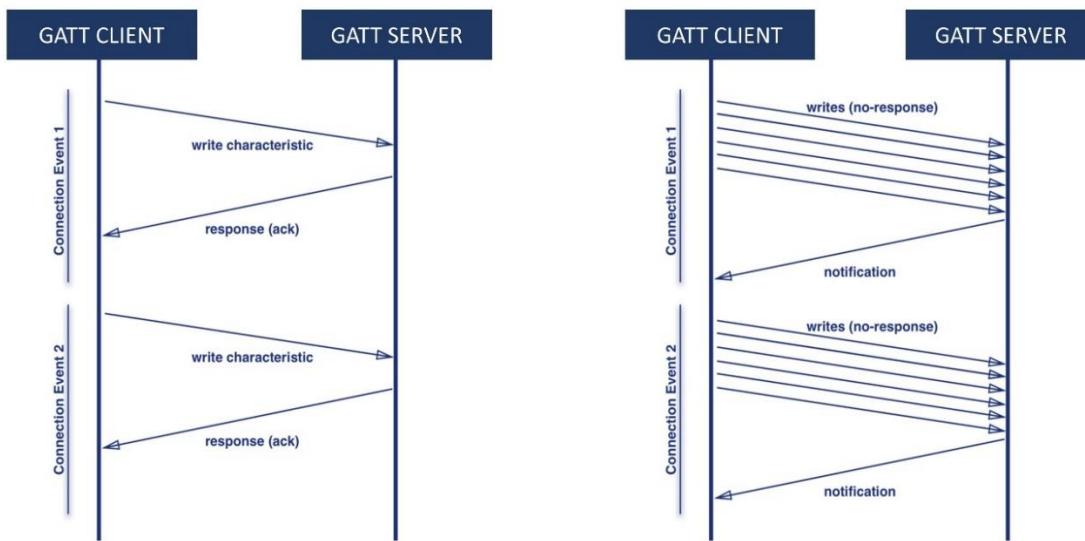


Figura 23 Proceso Write y Write_without_response

2.3.5.4. Conexiones GATT

Las conexiones GATT son exclusivas. Esto quiere decir que, si un periférico se conecta a una central, dejará de enviar mensajes GAP, y no podrá ser descubierto por otras centrales.

En la Figura 24 se muestra el intercambio de mensajes que se producen entre el periférico y la centra en una conexión GATT.

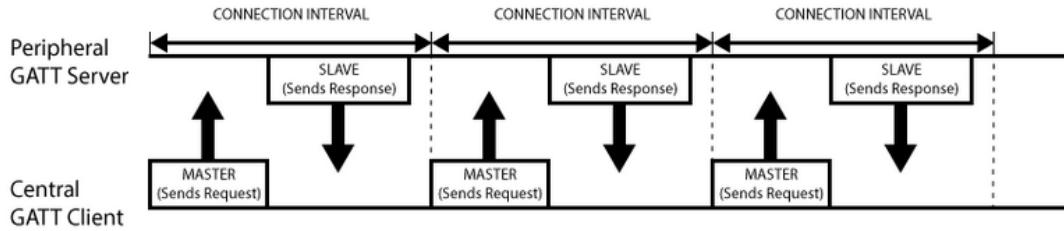


Figura 24 Intercambio de mensajes GATT [13]

Durante la conexión, los equipos enviarán mensajes con una frecuencia que habrá sido determinada al inicio de la conexión, como se explicó en la sección 2.3.4.1 y 2.3.4.2.

Los parámetros principales que entrarán en juego serán los siguientes:

- Intervalo de conexión: El intervalo de conexión será el tiempo entre dos eventos de conexión. Este tiempo puede estar en un rango entre 7,5 ms y 4 s. Servirá a ambos equipos para sincronizar las conexiones y poder determinar cuándo se producirá un envío/recepción. Esto permite que se utilicen canales de frecuencia distintos en cada envío, pero en la selección de éstos. En la Figura 25 se ilustra el comportamiento que deriva de este parámetro. [17]

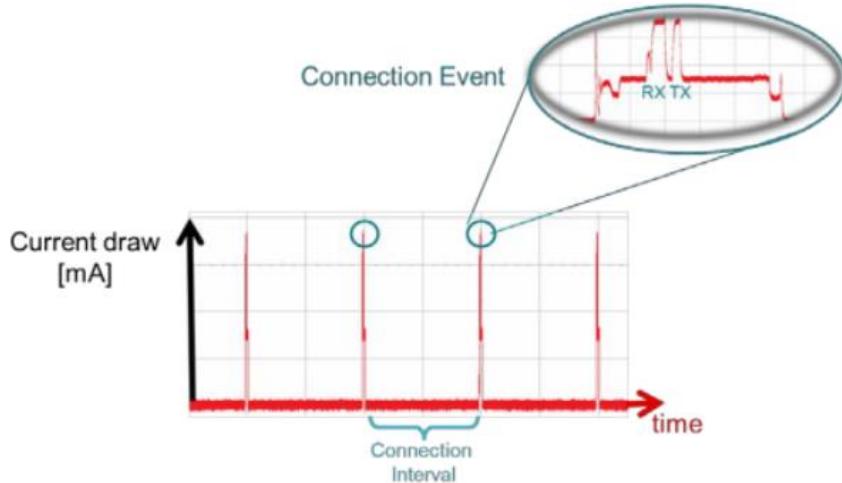


Figura 25 Consumo de potencia en estado de conexión [17]

- *Slave Latency*: este parámetro permite al *slave* (sección 2.3.5.1) saltarse ciertos eventos de conexión. En el caso de que el periférico (o *slave*) no tenga que mandar

datos, podría saltarse intervalos y pasar a modo *sleep*, lo que ahorraría energía. En la Figura 26 se muestra.

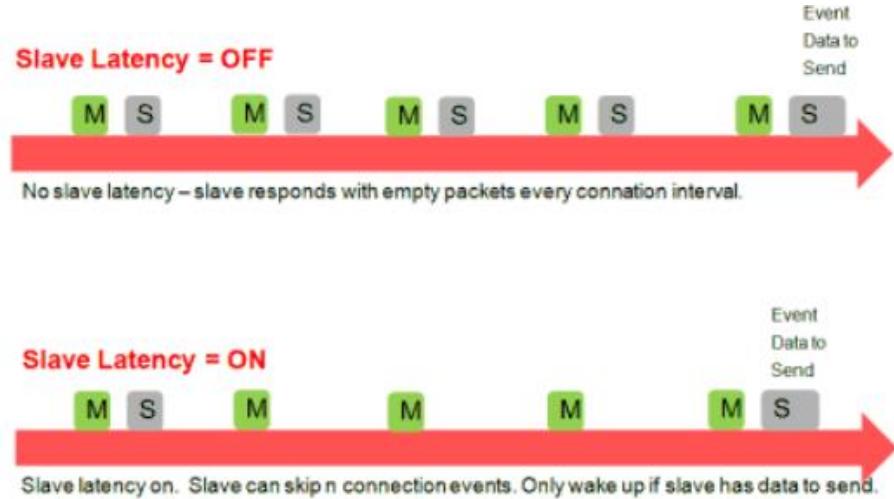


Figura 26 Slave Latency ilustración [17]

2.3.6. Servicios y características

Para facilitar la gestión de la tabla de datos GATT (sección 2.3.5.2), nos apoyamos en el uso de perfiles y servicios.

- **Servicios:** agrupa un conjunto de características. Se utilizan para separar funcionalidades específicas del dispositivo, como gestión de batería, información del dispositivo o control de las pantallas. Un servicio puede tener más de una característica. Cada servicio se identifica con UUID, de uso exclusivo. Existe una serie de UUID predefinidos para facilitar la compatibilidad [18].
- **Perfiles:** contiene un conjunto de servicios. Hace referencia a la funcionalidad global del dispositivo. Existe una colección predefinida de servicios [19] que facilita la compatibilidad con aplicaciones, aunque en nuestro caso tendremos que implementarlo.

En la Figura 27 se muestra gráficamente.

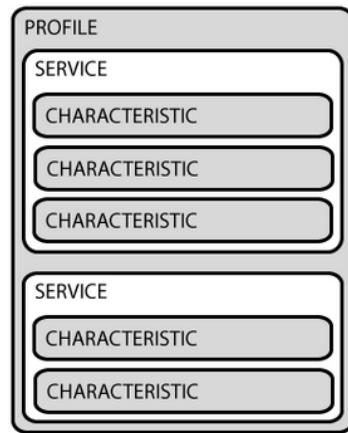


Figura 27 Objetos de transacción GATT [13]

3. Diseño del protocolo

En este capítulo mostraremos las decisiones de diseño que hemos tomado para implementar nuestro protocolo y las razones que nos han llevado a hacerlo.

La estructura de la red se muestra en la Figura 28. Esta red estará formada por:

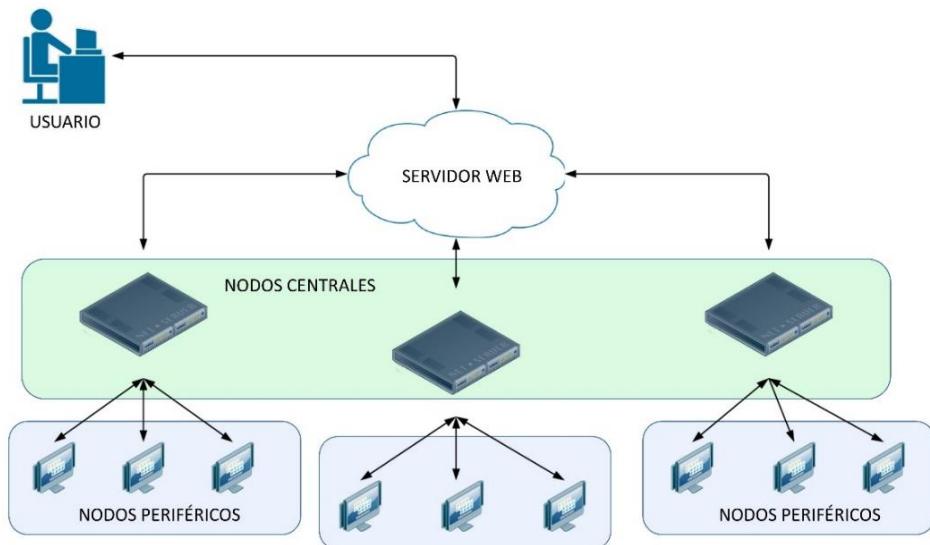


Figura 28 Esquema de la red IoT

- Servidor Web: estará albergado en los nodos centrales y se encargará de controlar el resto de la red. Será el punto de unión entre el usuario y el resto de equipos. Nos podremos conectar a con conexión a internet.

- Nodos centrales: como se ha mencionado, se encargarán de contener el servidor web y de controlar el resto de la red, es decir, los nodos periféricos. Estos dispositivos se incorporarán conexión internet (para la gestión del servidor) y conexión *Bluetooth Low Energy* para la comunicación con los nodos periféricos.
- Nodos periféricos: serán el punto final de nuestra red y se encargarán de albergar las pantallas y mostrar la información que reciba del nodo central. Las comunicaciones con este se harán con el protocolo *Bluetooth Low Energy*.

Nuestra misión será desarrollar la parte de la red que se extiende a partir del servidor web, es decir, los nodos centrales y periféricos.

Para entender mejor este protocolo y las decisiones de diseño se recomienda la lectura comprensiva de la sección 2.3, donde se explica a fondo los fundamentos del protocolo *Bluetooth Low Energy*. Dividiremos este proceso de diseño en dos partes principales:

- Selección de parámetros (sección 3.1): En esta sección entraremos en detalle sobre los parámetros de la conexión principalmente.
- Diseño de servicio y características(sección 3.2): en esta sección diseñaremos un protocolo que permita entender la comunicación entre los nodos periféricos y centrales, y estudiaremos la estructuración y el almacenamiento de los datos.

3.1. Selección de parámetros BLE

3.1.1. Parámetros y roles de descubrimiento BLE

Se recomienda la lectura y compresión de la sección 2.3.4 antes de empezar con este capítulo, en el que nos centraremos en la selección de los parámetros y los roles de los dispositivos durante el proceso de descubrimiento en la conexión *bluetooth low energy*.

El principal parámetro a seleccionar es el intervalo de *advertising* (sección 2.3.4.2), que podría variar desde 20 ms hasta 10,28 s.

Una frecuencia de *advertising* pequeña permitirá un descubrimiento más rápido, pero tendrá como consecuencia un mayor consumo de energía.

Un intervalo grande, aumentará el tiempo de descubrimiento, pero reducirá el consumo del dispositivo.

Como este intervalo se configura en el equipo periférico, en el que el consumo de energía será crítico, seleccionaremos un intervalo grande, de 4 segundos. Se puede apreciar que no hemos elegido el tiempo máximo. Esto se debe a que, aunque el consumo sea crítico, un tiempo de espera muy grande para la conexión podría dar una experiencia de usuario negativa, por lo que elegimos un valor intermedio.

En cuanto a los roles en el proceso de descubrimiento (sección 2.3.4.2), el *slave* será el equipo periférico, mientras que el central será el *master*, determinando los parámetros de la comunicación.

Tabla 1 Parámetros de advertising

Advertising Time	4 s
Slave	Nodo periférico
Master	Nodo central

3.1.2. Parámetros y roles de conexión BLE

Se recomienda la lectura previa de la sección 2.3.5 para la mejor compresión de este apartado, donde vamos a determinar los parámetros y los roles de los dispositivos en la etapa de conexión.

El principal parámetro a seleccionar será el intervalo de conexión (sección 2.3.5.4). Podemos seleccionar un rango de entre 7,5 y 4000 ms.

Seleccionaremos un tiempo mayor para casos en los que se tenga que transmitir poca información y el tiempo no suponga un inconveniente.

En nuestro caso, queremos mandar imágenes (gran cantidad de datos) y queremos hacerlo en el menor tiempo posible, por lo que seleccionaremos un tiempo de 7,5 ms.

El consumo de energía no se verá alterado por la variación de este parámetro, ya que entre intervalos entraremos en modo *sleep* (sección 2.3.5.4).

También definiremos un parámetro *timeout* para que finalice la conexión si no ha habido comunicación en un periodo de tiempo. Este periodo los definiremos como 10 segundos.

En cuanto a los roles en la conexión, el equipo periférico será el encargado de almacenar la información, ya que tiene que mostrarla en las pantallas. Por lo que será el GATT *Server* (sección 2.3.5.1).

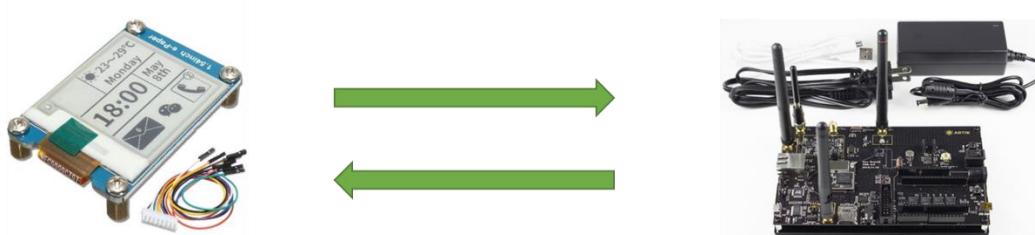
El nodo central actuará como GATT *client* (sección 2.3.5.1).

Tabla 2 Selección de parámetros conexión

Intervalo de conexión	7,5 ms
GATT Server	Nodo periférico
GATT Client	Nodo central

3.1.3. Parámetros seleccionados

A modo de recapitulación, mostramos en la Figura 29 los parámetros elegidos. En la sección 3.1.1 y 3.1.2 se justifican esta elección.



NODO PERIFÉRICO

- Rol descubrimiento: *Slave*
- Rol conexión: GATT *Server*
- Intervalo de descubrimiento: 4 s
- Intervalo de conexión: 7,5 ms

NODO CENTRAL

- Rol descubrimiento: *Master*
- Rol conexión: GATT *Client*
- Intervalo de descubrimiento: 4 s
- Intervalo de conexión: 7,5 ms

Figura 29 Resumen conexión BLE

3.2. Perfil BLE

Una vez concluida la selección de parámetros de nuestras comunicaciones, vamos a establecer el protocolo de comunicación. Esto consistirá en el diseño del perfil BLE que utilizaran los equipo para el envío de información.

En las siguientes secciones estructuraremos los datos a almacenar y definiremos un protocolo de comunicación entre ambos equipos.

3.2.1. Servicios por defecto

El protocolo *BLE* incluye por defecto dos servicios que son necesarios para la correcta identificación de nuestro dispositivo. Estos son *Generic Access* y *Device information*.

3.2.1.1. Generic Access

Este servicio contiene información de nuestro dispositivo. Las características que incluye son simplemente de lectura. Estas características tendrá asignado un valor según las especificaciones de *Bluetooth SIG* [19]. A continuación, se detallan [20]:

Tabla 3 Generic Access Service

CARACTERÍSTICA	DESCRIPCION	UUID
<i>Device Name</i>	Nombre del dispositivo	0x2A00
<i>Appearance</i>	Tipo de aplicación	0x2A01
<i>Peripheral Privacy Flag</i>	Tipo de privacidad del dispositivo	0x2A02
<i>Reconnection Address</i>	Dirección de reconexión	0x2A03
<i>Peripheral Preferred Connection Parameters</i>	Información relacionada con los parámetros de conexión preferidos	0x2A04

En nuestro caso los configuraremos del siguiente modo:

- ***Device name***: seleccionaremos el nombre del dispositivo con la máscara “SCREEN-XXX” donde XXX será un número único para cada dispositivo que se resolverá a partir de la dirección *bluetooth*.
- ***Appearance***: como nuestra aplicación no tiene predefinido ningún tipo de apariencia seleccionaremos el valor 0 (Unknown).
- ***Peripheral Privacy Flag***: sin uso.
- ***Reconnection Address***: asignación automática.
- ***Peripheral Preferred Connection Parameters***: los parámetros que especificamos en la sección 3.1.2.

3.2.1.2. Device information

Este servicio aporta información al fabricante o vendedor de nuestro dispositivo. Las características que engloba son: [21]

- *Manufacturer Name String*
- *Model Number String*
- *Serial Number String*
- *Hardware Revision String*
- *Firmware Revision String*

- *Software Revision String*
- *System ID*
- *IEEE 11073-20601 Regulatory Certification Data List*
- *PnP ID*

3.2.2. SPP Service

Para poder transmitir las imágenes, implementaremos nuestro propio servicio. Este servicio se encargará de simular una conexión serie en *bluetooth low energy*. Hasta la fecha no se han detallado especificaciones oficiales de perfiles que cumplan esta función.

A continuación explicamos el proceso a seguir para la actualización de una imagen de pantalla (Figura 30):

- 1) La conexión será establecida por el nodo central, que previamente habrá realizado un escaneo de los dispositivos. Una vez conectados, habilitará la opción *notify* para indicar al receptor que se va a iniciar la transmisión.
- 2) Se dividirá la imagen en paquetes, los cuales irán siendo enviados por desde el nodo central al periférico. El tamaño de estos paquetes se justifica en la sección 3.2.4.
- 3) Una vez concluido el envío, se indicará con la operación *notify off*, que indicará al nodo periférico que puede actualizar la pantalla con la imagen recibida.

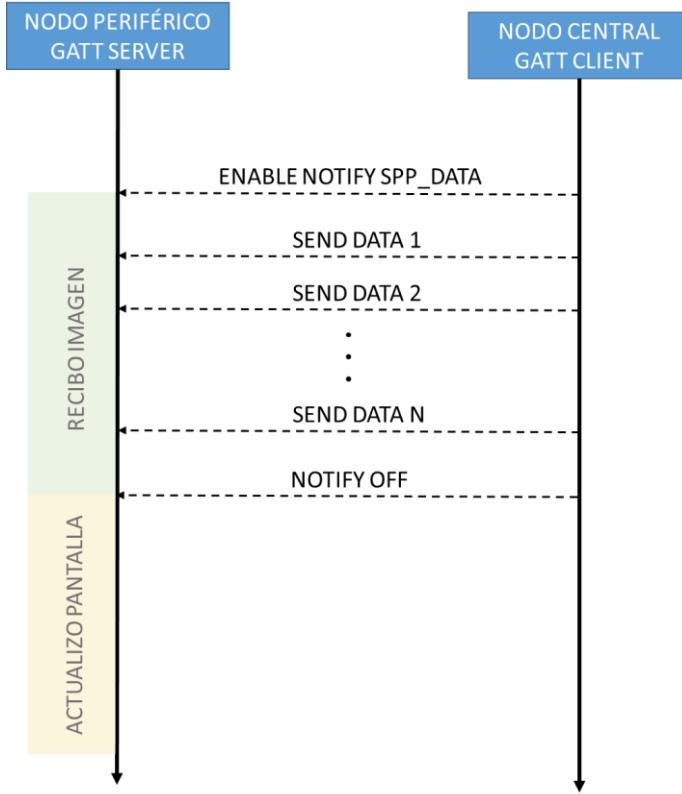


Figura 30 Protocolo envío imagen

3.2.2.1. Características del servicio SPP

En la versión final, este servicio incluye una única característica (SPP_Data). Ésta tendrá una longitud variable en formato hexadecimal, y se encargará de recibir los paquetes en los que dividiremos la imagen.

Con este proceso en mente, y lo detallado en la sección 2.3.5.3 diseñaremos nuestra característica con las siguientes propiedades:

- Notify: habilitaremos esta característica para anunciar que se va a enviar una imagen y poder preparar ambos equipos para el correcto envío/recepción. Una vez finalizado el envío lo indicaremos desactivando la notificación.
- Write: nos servirá para mandar los paquetes de la imagen.

Tanto el servicio como la característica, se identificaran con un UUID único que se muestra en la Tabla 4.

Tabla 4 Direcciones UUID Servicio

Nombre	UUID	Propiedades	Longitud
SPP Service	4880c12c-fdcb-4077-8920-a450d7f9b907	Servicio	-
SPP Data	fec26ec4-6d71-4442-9f81-55bc21d658d6	Write y Notify	250 (hex)

3.2.3. Almacenamiento imagen

Para poder almacenar la imagen en el nodo periférico, y poder registrar los parámetros de la transmisión, definiremos un protocolo de almacenamiento.

Éste consistirá en una variable de tipo *struct* que se almacenará en una posición de memoria fija y que contendrá los siguientes datos (Figura 31):

- *Data*: será un vector parametrizable donde almacenaremos la imagen. La longitud de éste dependerá del tamaño de pantalla. En nuestro caso utilizamos 4736 Bytes para almacenar la imagen, valor calculado en la ecuación 3, sección 3.2.4.
- *Pointer*: variable que estará apuntando a la última posición del vector en la que se almacenaron datos. Este nos permitirá saber dónde incluir los paquetes que nos vayan llegando.
- *Num_bytes_received*: almacenaremos la cantidad de información que hemos recibido.
- *Num_packets_received*: contendrá el número de paquetes que hemos recibido.

```
/*
 * STRUCT TO SAVE THE IMG
 */
typedef struct
{
    char data[image_length];      // Vector to storage data
    uint32 pointer;              // Pointer to control the storage
    uint32 num_bytes_received;   // Num of bytes count
    uint32 num_packets_recived; // Num of packets count
} image_struct;
```

Figura 31 Struct de imagen

Los datos de la conexión se mostrarán al usuario por el canal de *debug*. En la sección 5.1.1 se amplía esta información.

3.2.4. Tamaño de paquete

En esta sección vamos a realizar un estudio teórico de los tiempos de envío mínimos que conseguimos con el protocolo diseñado.

El tamaño máximo de paquetes (MTU) que podemos mandar para la versión de *stack BLE* que utilizamos (*Blue Gecko devices 2.8*) es de 250 bytes [22], aunque por defecto asignaremos un tamaño de 247 bytes.

Hay que tener en cuenta que en estos paquetes también se envía información relativa a la conexión *BLE* (PDU), por lo que el tamaño de los paquetes de imagen será variable en función a esta información, y lo calcularemos como se muestra en la ecuación 2. En la Figura 32 se muestra gráficamente este cálculo.

$$\text{Paquete imagen enviado} = \text{MTU} - \text{payload_bluetooth} \quad (2)$$

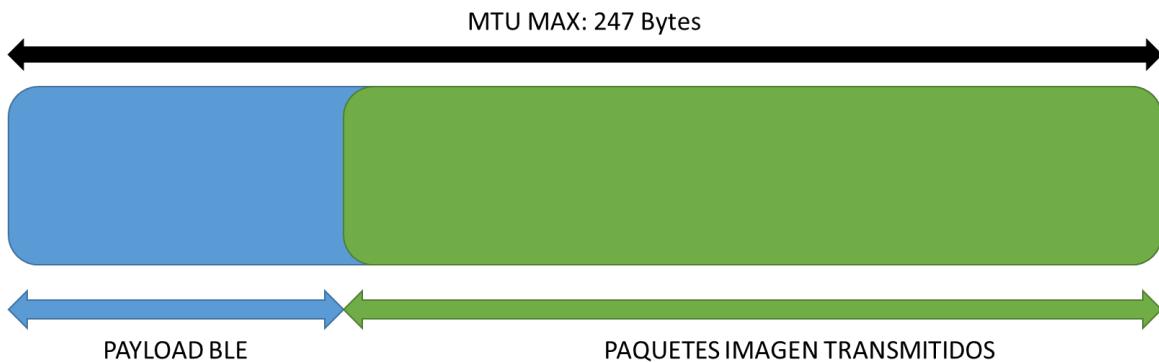


Figura 32 Tamaño de paquetes de imagen

Calculamos el tamaño de imagen para la resolución que estamos utilizando (296x128). En la sección 4.1 detallamos el porqué de esta resolución, que tendrá que ver con el tamaño de pantalla elegido.

$$\text{Tamaño imagen} = 296 \times 128 \text{ bit} = 4736 \text{ Bytes} = 4,625 \text{ KBytes} \quad (3)$$

Para el envío utilizaremos la operación de *write* (sección 2.3.5.3), lo que generará un ACK por parte del periférico. El esquema de envío de paquetes en un intervalo de conexión genérico se muestra en la Figura 33 [22]. El dispositivo central enviará un paquete de datos, a lo que el periférico responderá con una confirmación, por lo que serán necesarios al menos dos intervalos de conexión para cada envío.

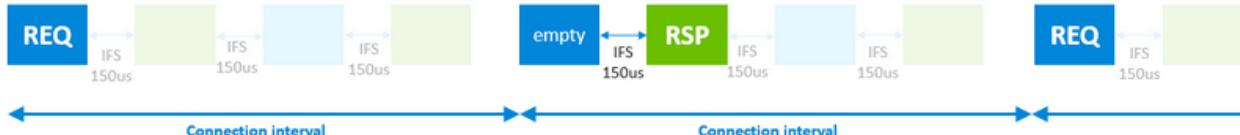


Figura 33 Envío de paquetes write without response

El *payload* máximo para la operación de *write*, es de 3 bytes [22].

El tiempo de envío entre paquetes es de 7,5 ms (sección 2.3.5.4).

Por lo que el tiempo máximo de envío de imagen será:

$$\begin{aligned}
 \text{Tiempo mínimo} &= \frac{\text{Tam imagen}}{\text{Tam paq}} \times 2 * t \text{ paq} \\
 &= \frac{4625 \text{ KBytes}}{247 - 3 \text{ Bytes}} \times (2 * 7,5) \text{ ms} = 0,29 \text{ s}
 \end{aligned} \tag{4}$$

La tasa de envío máxima que podemos conseguir según estos cálculos son:

$$\begin{aligned}
 \text{Tasa de envio máxima} &= \frac{1000 \text{ ms}}{2 * 7,5 \text{ ms}} * 244 \text{ Bytes} \\
 &= 16266 \text{ byte/sec} = 130133 \text{ bps}
 \end{aligned} \tag{5}$$

Podemos concluir que son tiempo más que aceptables para nuestra aplicación, teniendo en cuenta que trabajamos con un protocolo diseñado para envío de datos reducido.

En la sección 6.1.2 se contrasta estos resultados con los obtenidos en la implementación.

4. Selección de Hardware

En este apartado realizaremos un estudio sobre la elección de los dispositivos que nos permitirán implementar la red diseñada en la sección 3.

Para ello, abordaremos el estudio desde tres direcciones:

- Pantallas: hemos decidido trabajar con los modelos de la familia *Pervasive Display* [23], ya que ofrecían amplios recursos de desarrollo y librerías que nos facilitaban la utilización de éstas. En la sección 4.1 entraremos en detalle en el funcionamiento y la puesta a punto de este dispositivo.



Figura 34 Eink Pantallas

- Nodos periféricos: para las funciones de nodo periférico, hemos elegido el modelo de ARTIK 020 [24] del fabricante *Samsung*. Estos dispositivos incorporan la

funcionalidad *Bluetooth Low Energy*, y nos ofrecen un abanico de herramientas de desarrollo muy amplio. En la sección 4.2 se detallan sus características.

- Nodos centrales: para los nodos centrales hemos elegido los dispositivos ARTIK-520 [25], que se estudiarán en la sección 4.3. Estos dispositivos son ideales para nuestra función de nodo central, ya que disponen de un sistema operativo que nos permitirán gestionar el servidor web y a la misma vez llevar el control de las pantallas. Además, incorporan conectividad *Wifi* (para comunicarnos con el servidor) y BLE (para comunicarnos con las pantallas).



Figura 35 Samsung ARTIK

Para la comprobación del protocolo hemos utilizado un *sniffer* de paquetes BLE. En la sección 4.4 expondremos las principales características de este dispositivo.

4.1. Pervasive Display

En esta primera implementación trabajamos con el modelo 2.87" Aurora Mb (V231) [23]. Este es un tipo de *Electronic Paper Display* (EPD) que se basa en el principio de funcionamiento explicado en la sección 2.2. Utilizaremos la versión en blanco y negro, sin colores. [26]

En la Figura 36 se muestra esta pantalla. En cuanto a las dimensiones, la mostramos en la Figura 37. Es una pantalla de un grosor muy reducido y muy manejable, lo que la hace realmente atractiva para nuestra aplicación.



Figura 36 Pantalla E Ink Pervasive Display [23]

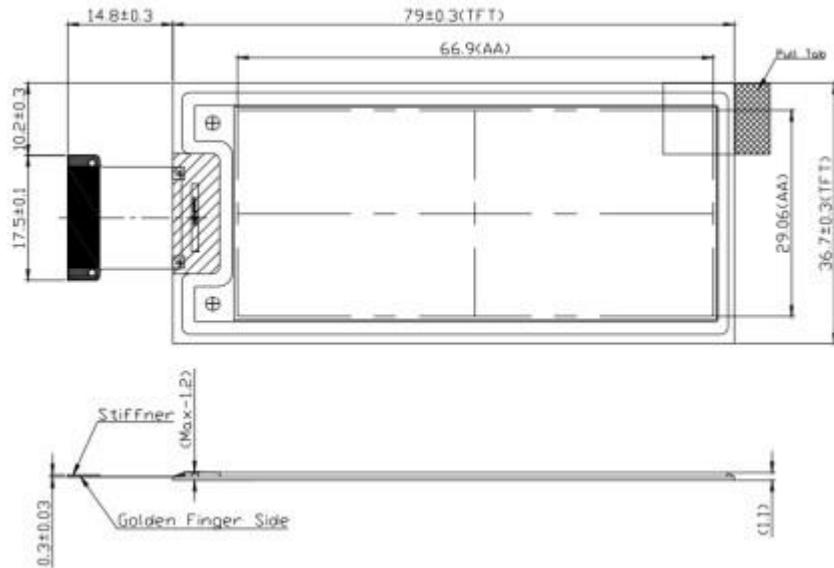


Figura 37 Dimensiones E-Ink Pervasive Display

Este modelo que hemos seleccionado, tiene una resolución de 296 x 128 (Figura 38) [26]. Para futuras implementaciones, se pretende utilizar varios tamaños de pantalla, pero en este proyecto solo trabajaremos con esta resolución.

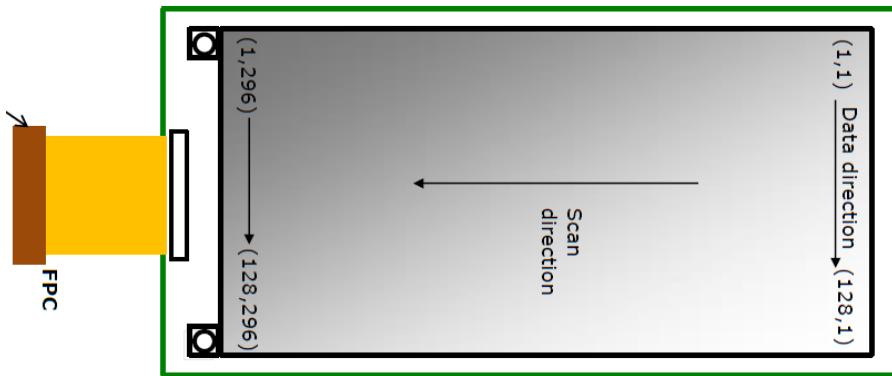


Figura 38 Resolución pantalla E Ink [23]

Para la programación de la pantalla, se utiliza un módulo SPI que se gestiona a través de un driver de la pantalla. (Figura 39).

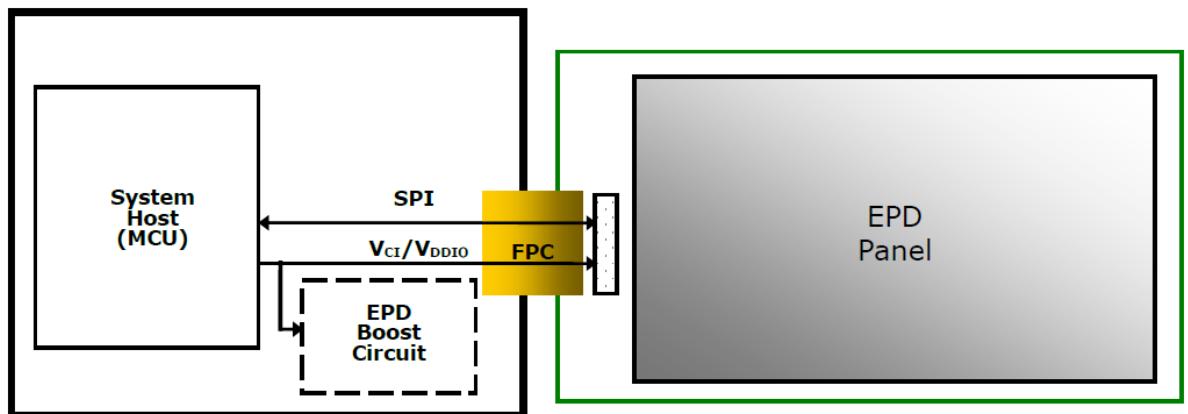


Figura 39 Driver EPD pantalla [23]

Para simplificar esta gestión, utilizaremos un kit de extensión, que nos facilitará este proceso. Utilizaremos el *EPD Extension Kit Gen 2* [27] de *Pervasive Display*. Este kit nos facilitará el proceso de actualización de la pantalla, ya que se encargará de gestionar los mensajes que se manda por SPI a la pantalla a través del uso de librerías. En la Figura 40 y Figura 41 se muestra una imagen de este kit. Además, nos ofrece todo el *software* necesario para esta gestión. En la sección 5.1.2 profundizamos en este tema.



Figura 40 Extensio Board frontal



Figura 41 Extension Board reverso

4.2. Atik-020

Para la implementación de los nodos periféricos hemos seleccionado el modelo ARTIK 020, del fabricante Samsung ARTIK [24]. Esto es un módulo de desarrollo de aplicaciones *IoT* basadas en el protocolo *Bluetooth Low Energy*.

Nos ofrece una amplia gama de herramientas de desarrollo, así como librerías y ejemplos, por lo que es una elección ideal para nuestro proyecto. En la sección 5.1 profundizaremos en esta herramientas.

En la Figura 42 mostramos el dispositivo. En esta primera versión, utilizaremos la placa de desarrollo (a la derecha), aunque en un futuro se podrá diseñar una PCB para reducir el tamaño, utilizando el módulo reducido (izquierda).



Figura 42 ATIK 020

Enumeramos las principales características de estos dispositivos que se tuvieron en cuenta a la hora de elegirlos:

- **Procesador:** ARM Cortex-M4 con DSP y punto flotante. Para nuestra aplicación será más que suficiente. Es importante que dispongamos de DSP para futuros tratamientos de imagen en tiempo real.
- **Memoria:** 32 kB RAM, 256 kB memoria flash. Como en nuestro caso solo vamos a implementar el protocolo *BLE*, sin SO, será más que suficiente.
- **Módulos:** ofrece una amplia gama de módulos (ADC, DMA, PWM...). Para esta primera implementación necesitaremos el protocolo UART para poder realizar las funciones de *debug*, las entradas y salidas de la pantalla, y el canal SPI para poder comunicarnos con ésta.
- **Bluetooth:** ofrece un módulo que opera en la frecuencia BLE (2.4 GHz) con modulación GFSK. Este ofrece una potencia de radiación máxima de 10,5 dBm. En la sección 6.1.1.1 se hará un estudio sobre esta potencia y el impacto que tiene en nuestra aplicación.

4.3. Artik 520

Para la implementación de los nodos centrales utilizaremos el modelo de ARTIK 520. Este es un módulo que implementa *system-on-Module* (SOM). Se utiliza también para aplicaciones *IoT* que requieran varios protocolos de comunicación. Como ya hemos mencionado, nosotros utilizaremos *Wifi* o *Ethernet* para comunicarnos con el servidor. Para las comunicaciones con las pantallas y nodos periféricos utilizaremos *Bluetooth Low Energy*.

Este módulo utiliza un sistema operativo que permitirá la gestión del dispositivo conectándonos a través del puerto serie con una terminal. Los sistemas operativos con los que trabaja son *Fedora* y *Ubuntu*. Nosotros trabajaremos con el sistema operativo *Ubuntu*. En la Figura 43 se muestra el dispositivo.



Figura 43 Artik 520 Module [25]

Las principales características que se han tenido en cuenta a la hora de la elección de este módulo son:

- **Procesador:** utiliza un procesador Dual Core ARM Cortex-A7 [28].
- **Memoria:** memoria dinámica DRAM de 512 MB y memoria flash de 4 GB.
- **Radio:** implementa los protocolos WLAN, BT, BLE y ZigBee. Como ya hemos mencionado, nosotros utilizaremos BLE y WLAN para nuestra aplicación.

- **Seguridad:** seguridad con autenticación punto a punto.

Las principales funcionalidades de modulo se muestran en la Figura 44.

Este módulo ofrece un respaldo de herramientas de desarrollo y librerías realmente útil, lo que también nos resultó atractivo a la hora de hacer la elección. Además, es de la misma familia, *Samsung*, que el dispositivo periférico.

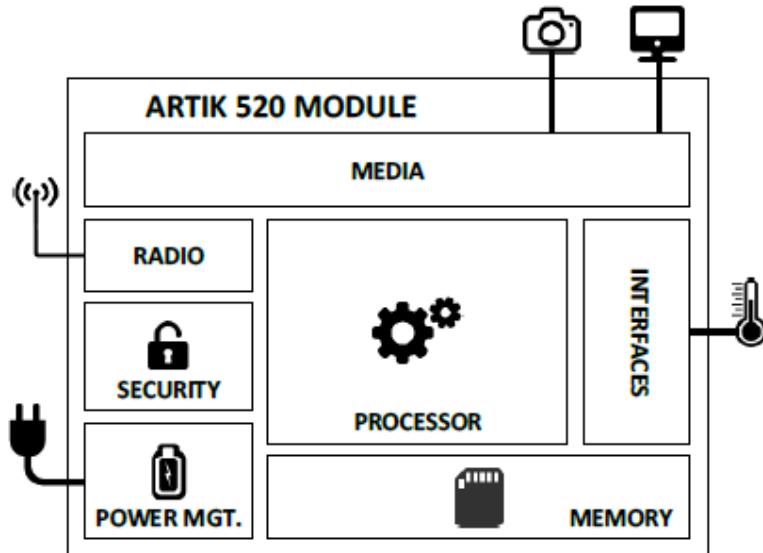


Figura 44 Funcionalidades ARTIK 520 [25]

Otro de los aspectos que más nos llamó la atención de este dispositivo, es la seguridad. Este módulo está considerado como uno de los más seguros en el mercado de nodos *IoT* [29], y como comentamos en la sección 2.1 es uno de los retos a tener en cuenta en el desarrollo de esta tecnología.

4.4. Sniffer nRF51822

Como se ha mencionado, utilizamos una herramienta de captura de paquetes para el *debug* de nuestra infraestructura y la monitorización de mensajes BLE utilizaremos, el sniffer de Bluefruit LE nRF518822 [30].

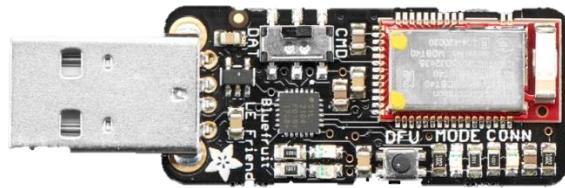


Figura 45 nRF518822

Este dispositivo viene programado con un *firmware* especial que nos permite convertirlo en un *Bluetooth low energy sniffer*. Con ello podremos capturar datos entre la conexión de dos dispositivos y mostrarlos por *Wireshark* [31].

Para la uso del dispositivo, nos hemos apoyado en la aplicación de *Nordic nRF* [32]. Este nos ofrece herramientas y drivers para poder lanzar el sniffer a través de su software (Figura 46) o a través de la aplicación *Wireshark* (Figura 47)

```
C:\Users\Salva\Desktop\ble-sniffer_win_1.0.1\ble-sniffer_win_1.0.1_1111_Sniffer.exe

Discovering and connecting to sniffer hardware...

Commands:
f          Erase the COM Port preset.
w          Start Wireshark, the primary viewer for the sniffer.
x/q        Exit
u          Launch User Guide (pdf)
s          Get support
CTRL-R    Re-program firmware onto board

+-----+
| Sniffer was previously at COM3 .
| Connecting to Sniffer at COM3 .
| If you want to use a new board, erase the preset with 'f',
|     then program with CTRL-R.
+-----+

Is the sniffer software already running somewhere else?
Try plugging out the hardware. Wait 5 seconds before plugging back in.
See the User Guide for Troubleshooting guide. (press 'u')
.
```

Figura 46 Nordic nRF

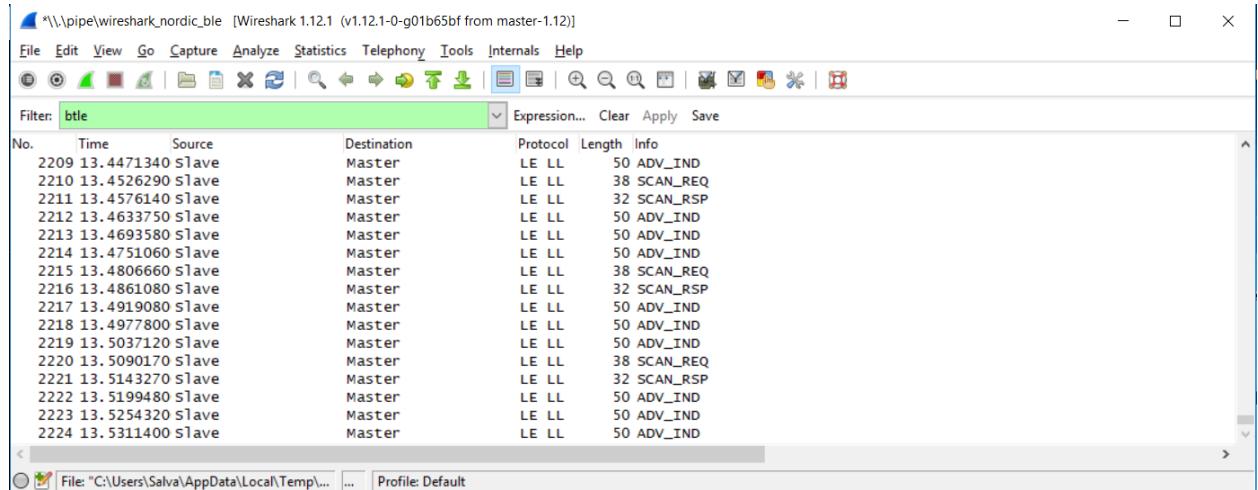


Figura 47 Ejemplo sniffer wireshark

Esta herramienta ha sido realmente útil, ya que nos ha permitido poder examinar que ocurría en nuestras comunicaciones con todo nivel de detalle.

5. Implementación del sistema

Una vez diseñado nuestra infraestructura, y habiendo seleccionado los dispositivos que vamos a utilizar, llega el turno de la implementación.

En esta sección mostraremos los recursos y herramientas utilizados para llevar a cabo nuestro sistema, así como las decisiones de diseño que se han ido tomando durante este proceso.

La sección estará dividida en dos partes principales:

- Nodos periféricos (sección 5.1): Esta sección se centrará en los nodos que incluyen las pantallas. Se mostrarán las herramientas de desarrollo, herramientas de *debug* (sección 5.1.1), la actualización de pantalla (sección), la gestión de eventos del dispositivo (sección 5.1.3), modos de energía (sección 5.1.4) y planificación de tareas (sección 5.1.5).
- Nodos centrales: (sección 5.2). En este punto se tendrá en cuenta la comunicación con el resto de la red *IoT*, realizando dos funciones principales: búsqueda de dispositivos (sección 5.2.1) y selección y envío de imagen (sección 5.2.2).

Iremos comentando durante todo el proceso las herramientas que hemos utilizado y los problemas que nos hemos ido encontrando, y como se han solucionado.

5.1. Nodos periféricos

Para la programación del ARTIK 020, hemos utilizado *Simplicity Studio IDE* de Silicon Labs [33] (Figura 48). Este *software* nos ofrece un entorno de desarrollo basado en Eclipse 4.5 con multitud de funcionalidades y recursos.

Simplicity Studio™ 4

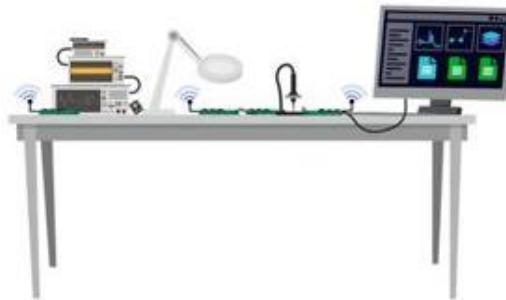


Figura 48 Simplicity Studio

De entre todos los recursos que se ofrecen, hemos utilizado principalmente:

- Las librerías que proporcionaba el IDE y el API de referencia de Silicon Labs [16, 34]
- El compilador y herramientas de *debug*.
- El modo de energía (Figura 49): como venimos comentando, el ahorro de energía es la principal especificación de los nodos periféricos. Por lo que esta herramienta nos ha sido realmente útil, ya que nos permite monitorizar el voltaje e intensidad consumida por nuestra placa en tiempo real. Además, nos ofrece una estimación de la potencia media consumida en el tiempo, lo que será realmente útil para la estimación de la autonomía del dispositivo (sección 6.1)

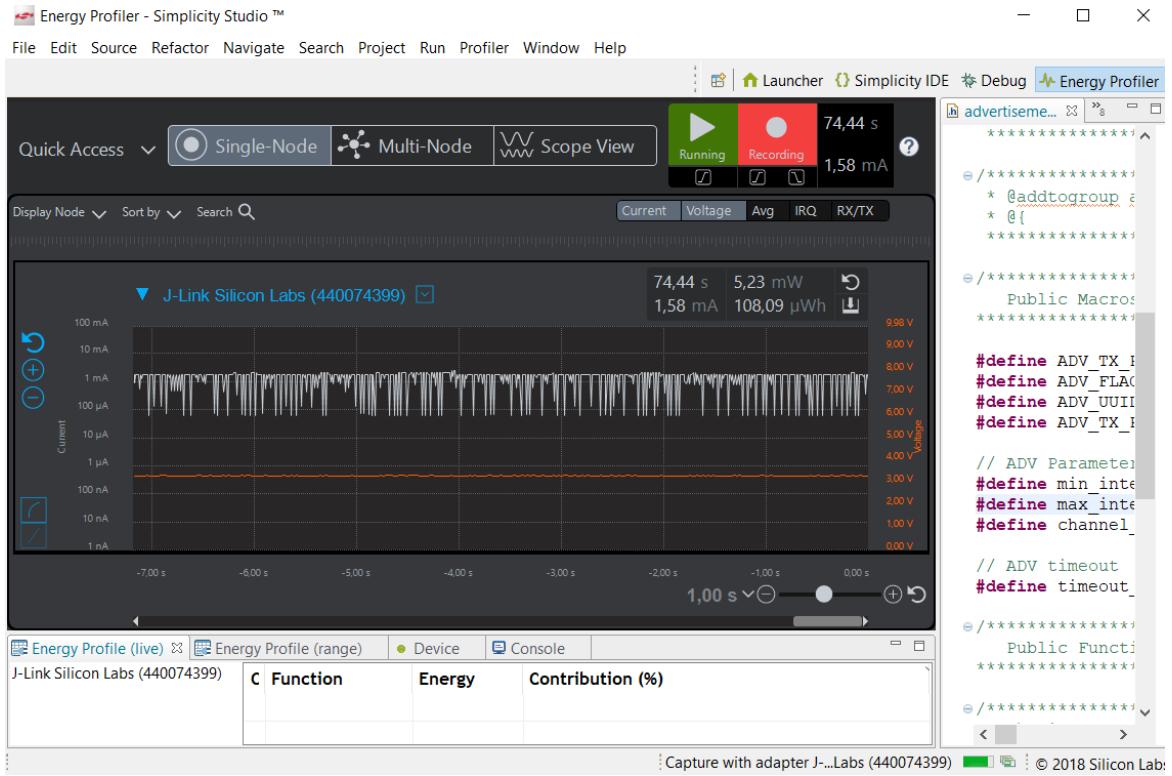


Figura 49 Energy profiler Simplicity Studio

5.1.1. Serial debug

Para poder mostrar la información del estado que se encuentra nuestro dispositivo, parámetros de conexión y descubrimiento, errores e información de la transmisión, decidimos habilitar un canal por puerto serie donde iremos mostrando estos mensajes.

Para ello habilitamos un puerto COM en el ARTIK020 y redirigimos los mensajes con *printf* por este puerto, redirigiendo la salida de *stdout* a través de la salida SWO de la placa. Mostramos los pasos que seguimos:

1. Añadir al proyecto las librerías de puerto serie: “retargetio.c”, “retargetserial.c” y “retargetserial.h”.
2. Añadir librerías a nuestro código (“#include stdio.h” “#include retargetserial.h”).
3. Habilitar puerto VCOM: (“HAL_VCON_ENABLE(1)”).
4. Inicializar función (“Retarget_serialInit()”).

Esta funcionalidad se podrá eliminar en un futuro para reducir costes de consumo de ejecución de tareas, pero para el desarrollo es realmente útil. En la Figura 50 se muestra una captura de una sesión de monitorización.

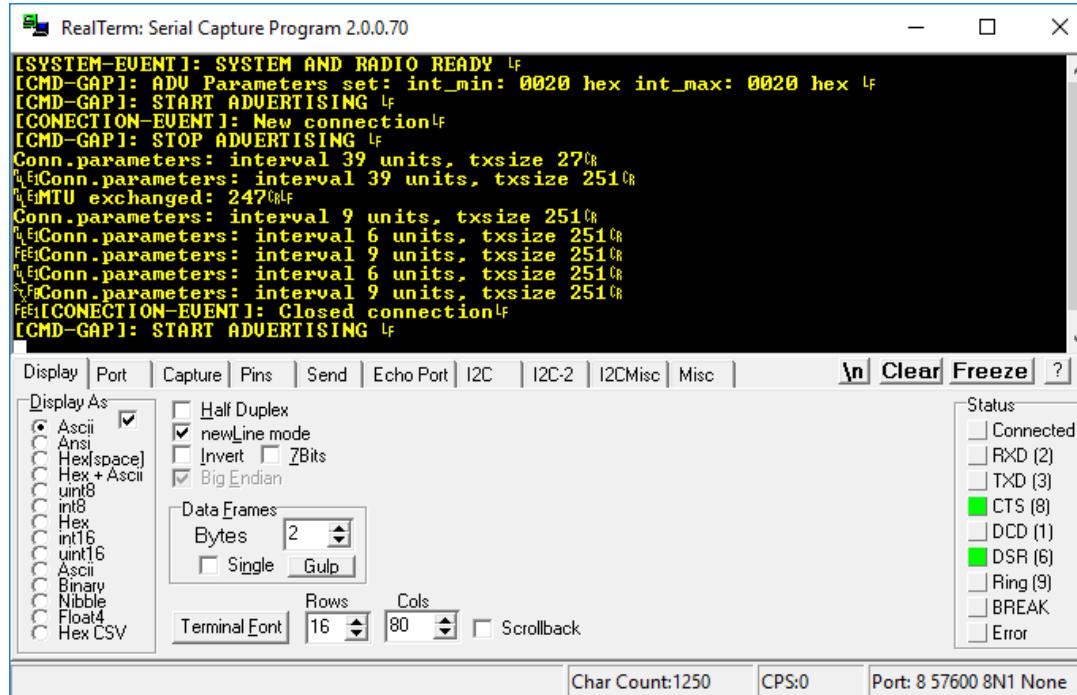


Figura 50 Ejemplo de captura serial

5.1.2. Actualización de pantalla

En esta sección mostraremos el primer paso de la implementación, la actualización de una imagen en la pantalla.

Para ello, utilizamos un Kit de extensión, *EPD Extension Kit Gen 2* [35]. Este kit nos facilitaba la tarea de actualización de pantalla, ya que dispone de librerías y *drivers*, que simplifican el proceso de actualización, como se adelantó en la sección 4.1.

Para la puesta a punto de las pantallas y la familiarización con su uso, utilizamos el software *PDi Apps v1.30*, facilitado por Pervasive Display [35]. En la Figura 51 se muestra una captura de este programa.

Este nos ofrecía unas herramientas de gestión de las pantallas en las que podemos destacar:

- Actualización directa del contenido de las EPD.
- Comprobación de algunos parámetros de las pantallas, como la temperatura.
- Conversión de imágenes a formato *BitMap*, que es el formato con el que trabajamos las imágenes en las librerías.

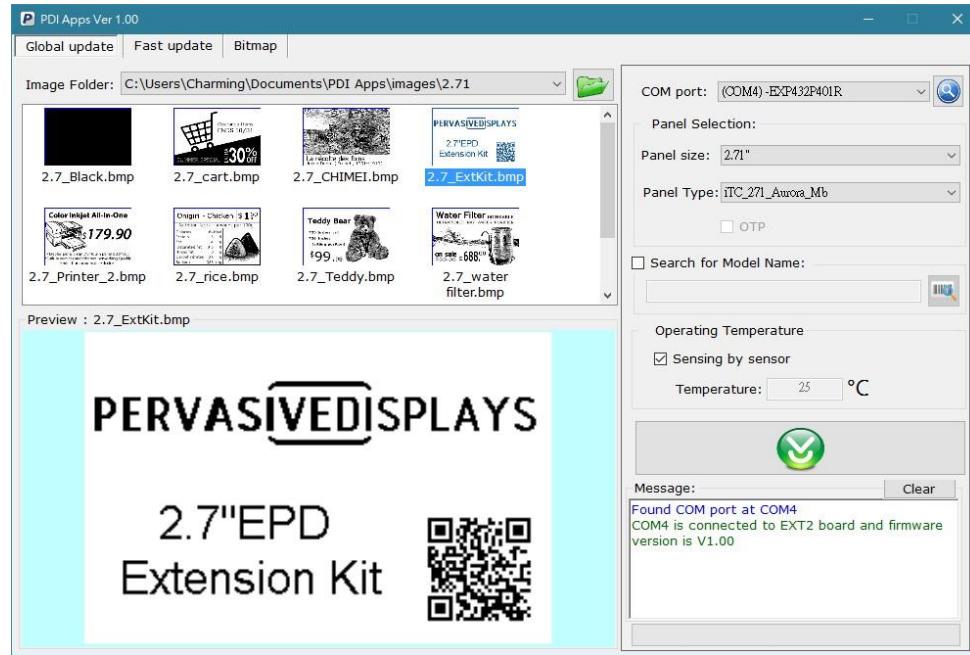


Figura 51 PDI Apps Software

Para la actualización de la pantalla desde nuestro dispositivo, consultamos la documentación del kit [27] y realizamos la asignación de pines que se muestran en la Figura 52.

El siguiente paso fue cablear la pantalla a nuestro dispositivo. En la Figura 53 mostramos este cableado.

```
#define SPIMOSI_PIN          6
#define SPIMOSI_PORT          gpioPortC
#define SPIMISO_PIN            7
#define SPIMISO_PORT           gpioPortC
#define SPICLK_PIN              2
#define SPICLK_PORT             gpioPortC
#define EPD_BUSY_PIN            3
#define EPD_BUSY_PORT           gpioPortF
#define PWM_PIN                  5
#define PWM_PORT                 gpioPortF
#define EPD_RST_PIN               1
#define EPD_RST_PORT              gpioPortA
#define EPD_PANELON_PIN          10
#define EPD_PANELON_PORT         gpioPortC
#define EPD_DISCHARGE_PIN         2
#define EPD_DISCHARGE_PORT        gpioPortA
#define EPD_BORDER_PIN             11
#define EPD_BORDER_PORT            gpioPortB
#define Flash_CS_PIN                4
#define Flash_CS_PORT              gpioPortF
#define EPD_CS_PIN                  9
#define EPD_CS_PORT                 gpioPortC
```

Figura 52 Cableado pantalla

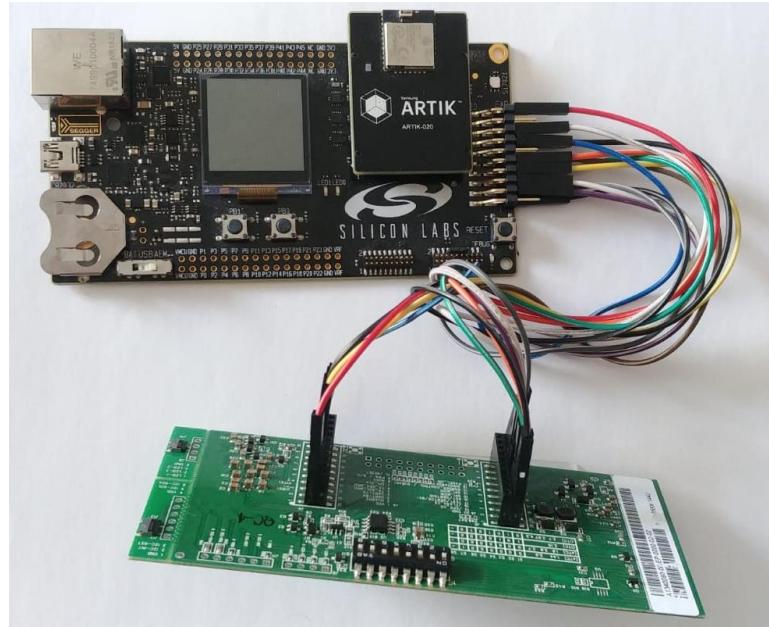


Figura 53 Asignación de pines

5.1.3. Gestión de eventos

En este apartado vamos a comentar como se han gestionado los eventos BLE. Este análisis surgió con la necesidad de mantener la conexión en las actualizaciones de pantalla. Esta tarea requiere un tiempo elevado en comparación con la gestión de eventos BLE, por lo que, si no lo gestionamos correctamente, tendríamos que reiniciar las conexiones en cada actualización, con las consecuencias de inestabilidad que eso acarrea.

Nuestro sistema implementará la máquina de estados que se muestra en la Figura 54.

El sistema permanece en estado *advertising* como estado inicial, esperando a que se reciba una petición de conexión. Los parámetros de *advertising* son los descritos en la sección 3.1.1

Cuando se registra este evento de conexión, establecemos la conexión con los parámetros descritos en la sección 3.1.2.

Una vez realizada la conexión, esperaríamos a que el nodo central indique el inicio de la transmisión de la imagen, recibiendo el evento *notify on* como se describió en la sección 3.2.2.

En este punto, pasaríamos al estado *SPP_MODE*, donde estaríamos recibiendo y almacenando la imagen.

Cuando la transmisión se ha completado, el nodo central deshabilitará el modo *notify* indicándonos que podemos pasar al modo de *Change_Screen*, donde actualizaremos nuestra pantalla. Una vez finalice este proceso, volveríamos al estado *advertising*.

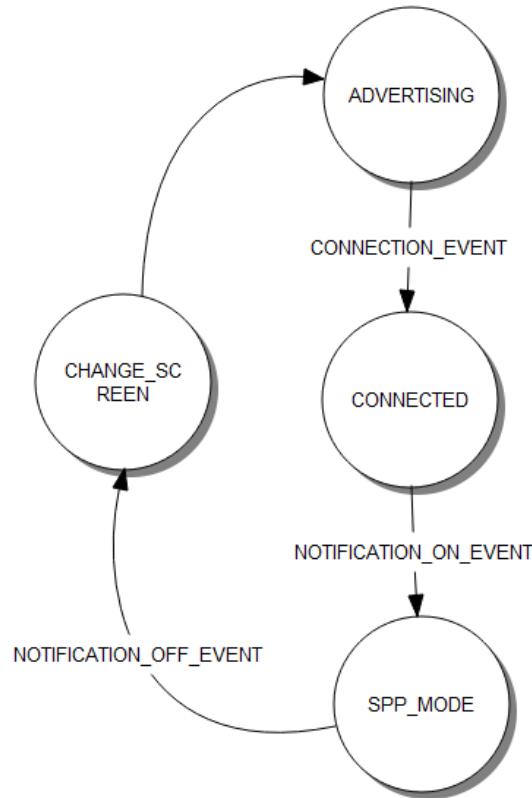


Figura 54 Máquina de estados ARTIK020

Existen dos formas de atender estos eventos:

- Gestión de eventos bloqueante: este modo se implementa con una función bloqueante, que espera la llegada de eventos a la cola para pasarlo a la función que los gestiona (*event handler*) [34]. La ventaja de esta gestión es que los modos de energía (sección 5.1.4) de nuestro dispositivo se gestionan de manera automática, garantizando el mínimo consumo. La desventaja es que no podemos ejecutar tareas entre la gestión de eventos, pues es bloqueante.

- Gestión de eventos no bloqueante: almacena los eventos en una cola que va procesando los eventos en cualquier momento, sin causar bloqueo de código [34]. Requiere la gestión manual de modos de energía (sección 5.1.4), con la ventaja de que nos facilita la gestión de tareas.

En nuestro caso alternaremos ambos modos en función del estado en que se encuentra nuestro sistema. En la sección 5.1.5 se detalla esta planificación.

5.1.4. Modos Energía del sistema

El objetivo de este apartado es mostrar los modos de energía en los que puede funcionar nuestro dispositivo, para facilitar la comprensión de la planificación que realizaremos en los siguientes apartados. Estos son [36]:

- Modo Activo (EM0): en este modo el procesador obtiene y ejecuta las instrucciones de la *flash* o la RAM y todos los periféricos de baja energía se pueden habilitar.
- *Sleep Mode* (EM1): el reloj estará desactivado, mientras que los dispositivos periféricos (incluida la RAM y la *flash*) mantienen su funcionamiento. Podemos recopilar datos de éstos sin el uso del procesador, a través del DMA y el *peripheral réflex system* (PRS).
- *Deep Sleep Mode* (EM2): en este modo se desactiva el oscilador de alta frecuencia, dejando un reloj de 32 kHz y el reloj de tiempo real para los periféricos.
- *Stop Mode* (EM3): adapta el consumo para mantener un tiempo de activación muy corto y responder a las interrupciones externas. El oscilador de baja frecuencia estará deshabilitado.
- *Shutoff Mode* (EM4): este modo toda la MCU estará desactivada, dejando solo una posibilidad de *wake up* con un reset.

5.1.5. Planificación tareas

En este apartado mostraremos la planificación de tareas de nuestro sistema. Haremos una planificación en tiempo real, de manera que podamos acotar el tiempo en que se realizaran estas tareas. Nuestro dispositivo realizará tres tareas principales:

- Gestión de eventos de conexión (*stack*): se encargarán de procesar los eventos que vayan ocurriendo para la gestión de la conexión y envío. Se estarán ejecutando constantemente.
- Recepción y almacenamiento de imagen (T1): será el proceso de recibir y almacenar la imagen que queremos mostrar. Esta tarea se iniciará cuando el nodo central active el modo *notify*, como se detalló en la sección 3.2.2.
- Actualización de pantalla (T2): una vez recibida y almacenada la imagen, pasaremos a actualizar la pantalla. Esta tarea se iniciará cuando se indique el fin de la transmisión de la imagen, con *notify off*.

La planificación que realizaremos se muestra en la Figura 55. Como vemos estaremos en modo de gestión de eventos de conexión será bloqueante (sección 5.1.3) hasta que se inicie la actualización de la pantalla. Cuando se ejecutan las tareas 1 y 2 pasamos al modo de gestión de eventos no bloqueante (sección 5.1.3). De este modo, podremos mantener la conexión independientemente del tiempo que dure la actualización de la pantalla.

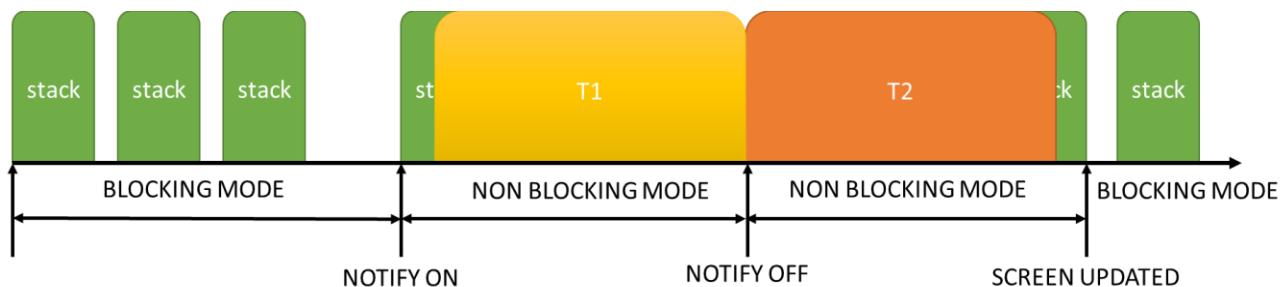


Figura 55 Planificación de tareas periférico

Como se detalló en la sección 5.1.3, en el modo de operación no bloqueante tendremos que gestionar los modos de energía en los que opera nuestro sistema. En nuestro caso nos

mantendremos en el modo EM1 (sección 5.1.4) en la ejecución de las tareas 1 y 2, regresando al modo EM2 el tiempo restante, para minimizar el consumo al máximo.

5.2. Nodos centrales

En este apartado mostramos el desarrollo de los nodos centrales de nuestra red. Las características de dispositivos se puede estudiar en la sección 4.3.

Para la implementación en el ARTIK 520 (sección 4.3), hemos utilizado ARTIK IDE de Samsung, con entorno Eclipse. [37] (Figura 56). Este entorno nos ofrece librerías SDK, herramientas y los drivers que necesitamos para gestionar nuestro módulo.

Durante todo el proceso también hemos ido utilizando documentación y ejemplos que están disponibles en la página web del *Samsung Artik Modules* [38], y que nos ayudaron sobre todo en los primeros pasos de la implementación.



Figura 56 ARTIK IDE

Las funciones que realizaran este dispositivo, las podemos dividir en dos:

- Búsqueda de dispositivos BLE (sección 5.2.1): se encargará de realizar una búsqueda de los dispositivos BLE para poder detectar las pantallas que están disponibles.
- Envío de la imagen a un determinado dispositivo (sección 5.2.2): una vez escaneados los dispositivos disponibles, seleccionaremos el periférico deseado y procederemos con el envío de la imagen, siguiendo el proceso que se detalló en la sección 3.2.2.

Estas dos tareas se implementarán en programas independientes que serán llamados por el usuario a través del servidor web. Además, tendremos que implementar estas comunicaciones con este servidor web que se encargará de gestionar la red, para lo que utilizaremos el formato JSON [39].

5.2.1. Búsqueda de dispositivos

Como hemos mencionado, la primera tarea será descubrir los dispositivos que están a nuestro alcance y que nos permitirán conectarnos a ellos para mandar las imágenes.

Este primer programa se encargará de realizar esa tarea. Para ello, se seguirá el siguiente proceso (Figura 57):

1. En primer lugar, inicializaremos el módulo *Bluetooth* de la placa y nos aseguraremos de que esté disponible para ser utilizado. En caso contrario, mostraremos un mensaje de error por la salida *stderr*.
2. Despues iniciaremos el escaneo de dispositivos durante un tiempo parametrizable que se ha predefinido a 10 segundos. La idea de esta parametrización es la posibilidad de ser modificado desde el servidor en futuras implementaciones.

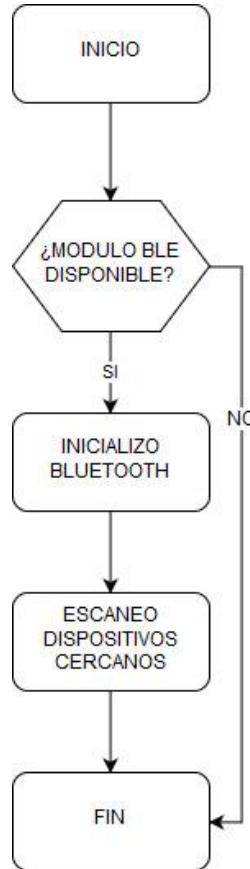


Figura 57 FlowChart Busqueda BLE

Una vez obtenidos los dispositivos, mostraremos la salida en formato JSON [39], para facilitar las comunicaciones con el servidor web (Figura 58). Los parámetros que mostraremos son:

- Id: será un número de identificación dentro de nuestra búsqueda.
- MAC: dirección física del dispositivo encontrado.
- Screen: número de pantalla que gestiona el dispositivo.
- RSSI: intensidad de la señal.
- Bat: nivel de batería.
- Initcode: código de inicialización del dispositivo. Código que mostraremos en pantalla en el primer arranque de ésta, para poder identificarla.

```

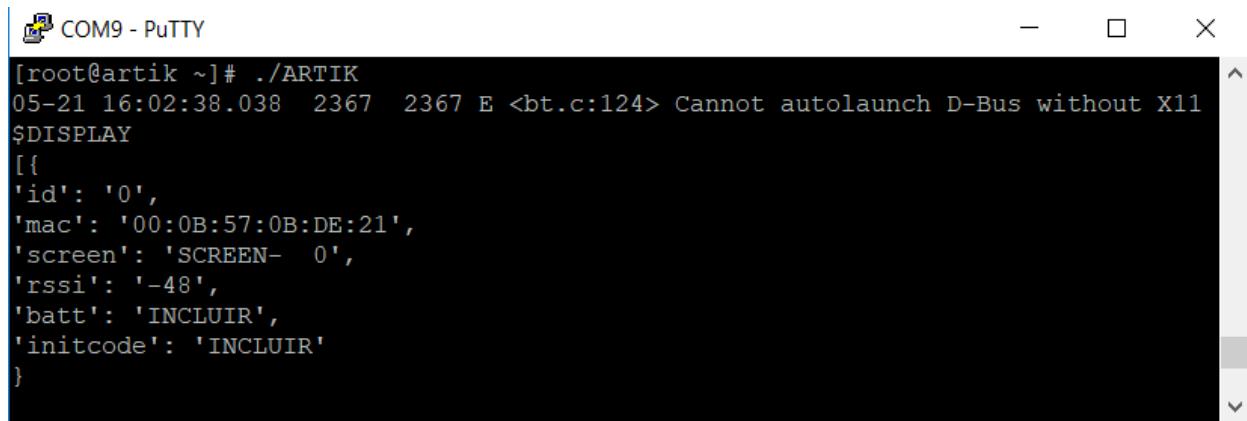
    "devices": [
        {
            "id": "0",
            "mac": "3DF2C9A6B34F",
            "screen": "27bw",
            "rssи": "-68",
            "batt": "90",
            "initcode": "000369"
        },
        {
            "id": "1",
            "mac": "3DF2C9A6B36D",
            "screen": "27bw",
            "rssи": "-46",
            "batt": "56",
            "initcode": "000325"
        },
        {
            "id": "2",
            "mac": "3DF2C9A6B33B",
            "screen": "27bw",
            "rssи": "-87",
            "batt": "78",
            "initcode": "000326"
        }
    ]

```

Figura 58 Formato salida escaneo dispositivos

En la Figura 59 se muestra un ejemplo de una búsqueda de dispositivos. Como se ha mencionado anteriormente, los mensajes de error se redirigen por la salida *stderr*. La información del programa se mostrará por la salida *stdout*. Esto hará que los posibles errores que lance el

sistema operativo, que no tienen que ver con nuestro programa, no afecten al correcto funcionamiento de éste.



```
root@artik ~]# ./ARTIK
05-21 16:02:38.038 2367 2367 E <bt.c:124> Cannot autolaunch D-Bus without X11
$DISPLAY
[{
'id': '0',
'mac': '00:0B:57:0B:DE:21',
'screen': 'SCREEN- 0',
'rssi': '-48',
'batt': 'INCLUIR',
'initcode': 'INCLUIR'
}]
```

Figura 59 Búsqueda dispositivos ARTIK 520

5.2.2. Envío de imagen

En este apartado detallaremos como se ha implementado el programa que se encargará de enviar la imagen del nodo central al periférico.

La imagen a enviar estará alojada en nuestro dispositivo, tarea de la que se encargará el servidor web. Los parámetros de entrada a nuestro programa serán la dirección MAC a la que queremos enviar la imagen (previamente escaneada como se detalla en la sección 5.2.1) y la ruta donde se ubica ésta imagen.

En la Figura 60 se muestra el diagrama de flujo del programa. Los pasos a seguir a la hora de enviar una imagen son:

- 1) Adquisición de imagen desde el fichero indicado: La imagen estará alojada en un fichero .txt que será generado por el servidor web en formato ASCII. En nuestro programa realizaremos la conversión de este fichero a formato Hexadecimal, almacenándolo en un vector.
- 2) Comprobación e inicialización del módulo *bluetooth* del dispositivo: Nos aseguraremos de que el módulo BLE de nuestro dispositivo se encuentra disponible para su uso, y en tal caso, lo inicializaremos. Si por algún motivo no fuera posible, enviaremos un mensaje de error por la salida del programa *stderr*.

- 3) Emparejamiento y conexión con la dirección MAC introducida: Una vez inicializado, iniciaremos la conexión con el dispositivo. Nuevamente, en caso de error, se indicará por la salida *stderr*. En caso de conexión satisfactoria, comenzaríamos con el proceso de envío de la imagen.
- 4) Envío de imagen: como se detalló en la sección 3.2.2, el primer paso será activar el modo *notify*. Hecho esto se empezaría a mandar paquetes de 244 Bytes (sección 3.2.4) hasta que la imagen haya sido transmitida por completo.
- 5) Finalización de envío: una vez concluido el envío, se procederá a indicar la finalización (*notify off*) y finalizar la conexión BLE.

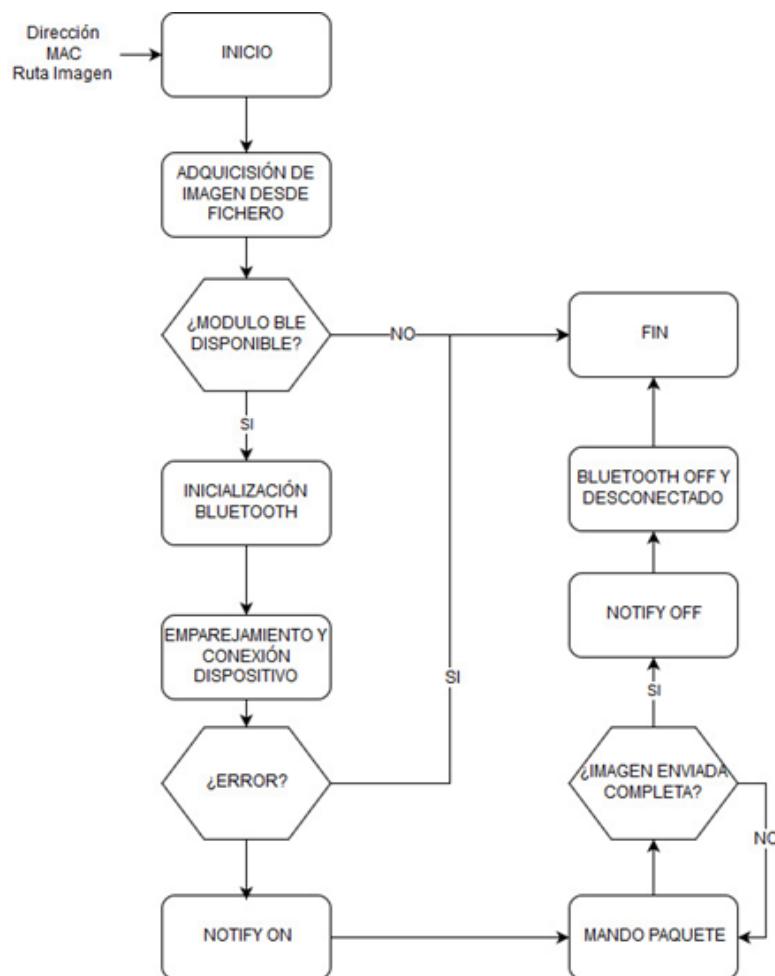


Figura 60 FlowChart Envío imagen

6. Resultados y conclusiones

Para concluir con el proyecto, iniciamos este apartado en el que mostraremos los resultados, conclusiones, presupuesto y trabajo futuro.

En primer lugar analizaremos el grado de cumplimiento de las expectativas, así como los resultados obtenidos, en la sección 6.1. Centraremos este análisis en dos direcciones: análisis del consumo (sección 6.1.1) y tiempo de envío de imagen (sección 6.1.2).

Continuaremos este apartado con una estimación económica del proyecto (sección 6.2) para estimar los costes que tendría un modelo comercial de este sistema. Del mismo modo, también analizaremos los costes de desarrollo que ha supuesto esta implementación.

Para concluir, en la sección 6.3, comentaremos los resultados de nuestro proyecto y las sensaciones generales con las que terminamos, e introduciremos posibles líneas de trabajo futuro para la continuación de éste.

6.1. Resultados

Como se mencionó en la sección de objetivos (1.1), las especificaciones de nuestro proyecto se podían resumir en dos principalmente:

- Minimizar al máximo el consumo para poder aumentar la autonomía del dispositivo.
- Reducir los tiempos de envío al máximo para mejorar la experiencia de usuario.

Por lo que centraremos el análisis de los resultados en calificar el grado de cumplimiento de esos objetivos iniciales.

6.1.1. Análisis de consumo

Para la estimación del consumo, nos centraremos en el nodo periférico, que será el crítico en este sentido. El nodo central se pretende alimentar en la red eléctrica y no tendría limitaciones de consumo, por lo que vamos a obviar este estudio.

Hay dos factores principales que afectan al consumo en las comunicaciones BLE, la potencia de transmisión y el tiempo de transmisión o recepción (rx o tx). Los tiempo de transmisión ya han sido detallados en la sección 3.1.3, por lo que nos centraremos en la potencia de transmisión.

La cantidad de potencia requerida dependerá de la distancia entre el central y el periférico. A mayor potencia de transmisión, mayor distancia entre dispositivos. Esta distancia se puede ver reducida por interferencias que provendrán principalmente de dos fuentes: los objetos que se interpongan entre los equipos y por el tráfico a 2,4 GHz que interfiera en la conexión. [40]

Es importante mencionar que para las estimaciones de consumo, tendremos que desactivar la inicialización del canal de *debug* que habilitamos para desarrollar nuestra aplicación (sección 5.1.1), ya que influirá en el consumo.

Para la estimación de la vida de la batería, tomaremos como referencia una batería recargable Li-Ión con una capacidad de 4000 mAh (3.7V/14.40 Wh) [41]. En la Figura 61 se muestra ésta elección. Las dimensiones son (10 cm x 7 cm x 0,5 cm).



Figura 61 Batería de alimentación

Dividiremos este estudio en dos subapartados, que coinciden con los dos estados en los que se puede encontrar nuestro sistema: modo *Advertising* (sección 6.1.1.1) y modo actualización de pantalla (sección 6.1.1.2).

En el último apartado, sección 6.1.1.3, faremos una estimación de la autonomía de nuestra red para un caso de aplicación real.

6.1.1.1. Consumo en modo Advertising

En este apartado vamos a analizar el consumo del nodo periférico en el modo *advertising*. Para este estudio, y durante todo el proceso, utilizaremos la herramienta ofrecida por el software *Simplicity Studio* de análisis energético, introducido en la sección 5.1.

En primer lugar, vamos a realizar un estudio de la duración de la batería en función de la potencia de radiación de la antena, como se ha comentado en la introducción el apartado. Para ello, utilizaremos la siguiente expresión:

$$Autonomía = \frac{P_{batería} (Wh)}{P_{dispositivo} (W)} \quad (6)$$

Para ello, iremos calculando el consumo medio en función de la potencia de transmisión. Analizaremos cuatro valores intermedios en el rango de 0 dBm (mínima) a 10,5 dBm (máxima) [42]. En la Tabla 5 se muestra estos cálculos.

Tabla 5 Consumo de potencia modo advertising en función de TX power

TX Power	Potencia consumida	Autonomía
0 dBm	134,7 µW	107.246 h = 4468 días = 12,24 años
2,5 dBm	173,56 µW	82968 h = 3457 días = 9,47 años
7,5 dBm	212,87 µW	67646 h = 2818 días = 7,72 años
10,5 dBm	258,08 µW	55796 h = 2324 días = 6,36 años

En la Figura 62 representamos estos resultados. Como era de esperar, la duración de la batería es inversamente proporcional al nivel de radiación de la antena, de manera lineal. Esto habrá que tenerlo en cuenta a la hora de instalar nuestra red, y entender que trabajaremos siempre con el nivel de radiación más bajo que se permita, para poder alargar la vida de la batería.

Por el momento hemos seleccionado la máxima potencia de radiación (10,5 dBm), que nos ofrece un alcance de 25-40 m en un edificio con paredes e interferencias de la red (2,4 GHz). De todos modos, este valor se ha seleccionado de manera parametrizada, de modo que, en un futuro, se pueda mostrar la opción de modificarlo desde el servidor web desde el servidor web.

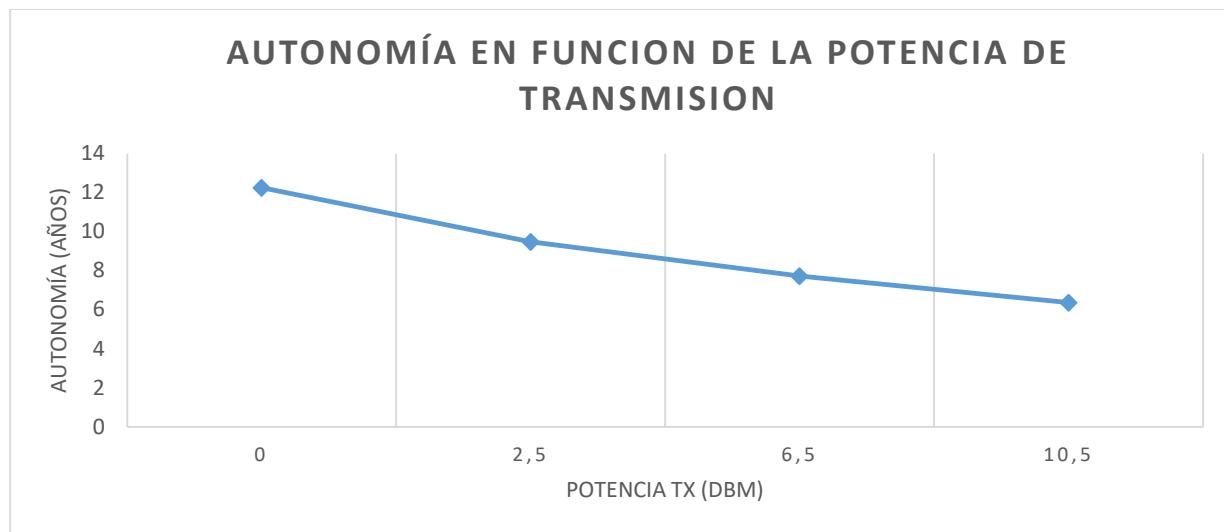


Figura 62 Autonomía en función de la potencia

A modo de ejemplo, en la Figura 63 Consumo para transmisión de 0 dBmFigura 63 se muestra la captura de consumo para una potencia de transmisión de 0 dBm (mínima potencia), mientras que en la Figura 64 mostramos para 10,5 dBm (máxima potencia). En el margen superior derecho podemos ver estos valores de potencia media.

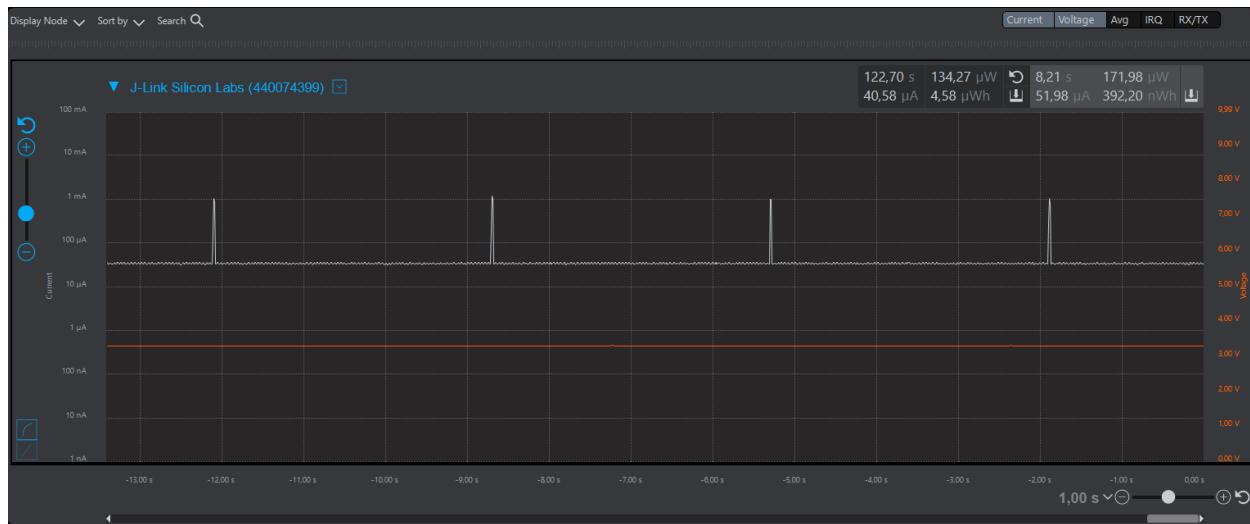


Figura 63 Consumo para transmisión de 0 dBm

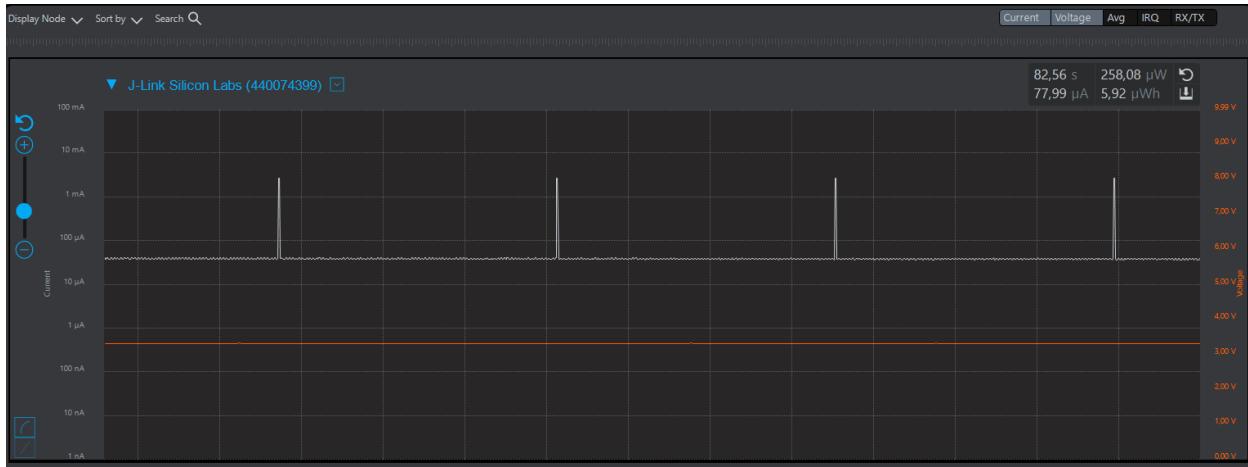


Figura 64 Consumo para transmisión de 10,5 dBm

6.1.1.2. Consumo en modo de actualización de pantalla

En este apartado vamos a analizar el consumo que acarrea la operación de actualización de pantalla, entendiendo por esto, el proceso que transcurre desde iniciamos la conexión, hasta que la imagen ha sido mostrada en la pantalla.

Aunque también influirá la potencia de transmisión, no la tendremos en cuenta, ya que en las actualizaciones de pantalla trabajamos en el orden de mA. Por lo que todas las estimaciones se harán para la potencia de transmisión máxima fijada (10,5 dBm).

En la Figura 65 se muestra este cálculo

$$P_{\text{envío-actualización}} = 29,64 \text{ mW} \quad (7)$$

$$T_{\text{actualización}} = 9,13 \text{ s} \quad (8)$$

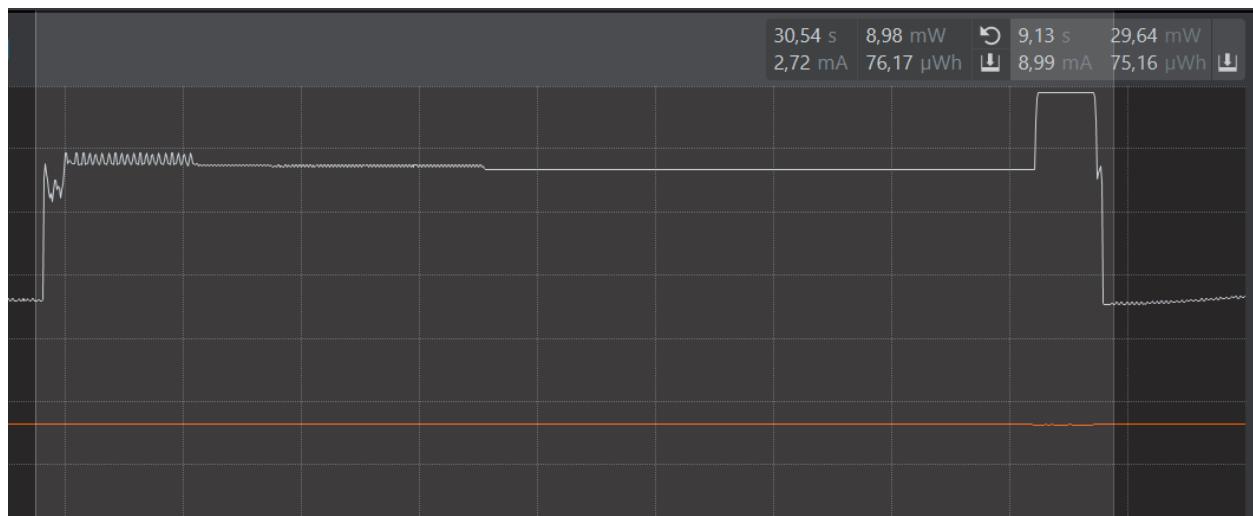


Figura 65 Envío y actualización pantalla

6.1.1.3. Consumo en situación real

En este apartado vamos a estudiar la autonomía del dispositivo combinando ambos modos de operación, el modo *advertisement* (sección 6.1.1.1) y el modo de transmisión y actualización de pantalla (sección 6.1.1.2).

Para ello, vamos a utilizar la batería descrita en la sección 6.1, de 14,4 Wh. Además, vamos a suponer que, en un uso normal del sistema, se producirían 4 actualizaciones de pantalla diarias de media. En la Figura 66 se muestra este escenario.

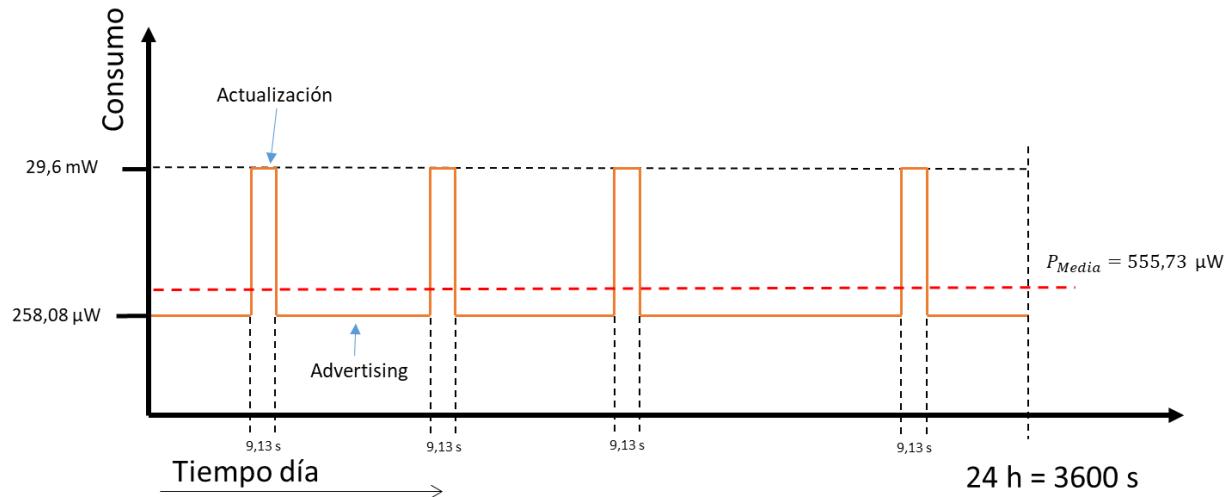


Figura 66 Consumo medio diario

La potencia media consumida por nuestro sistema en este escenario, la podemos calcular con la siguiente expresión con la siguiente expresión:

$$\begin{aligned}
 & P_{media}(W) \\
 &= \frac{258,08 \mu W * (3600s - 4 * 9,13s) + 29,6 mw * 4 * 9,13s}{3600 s} \\
 &= 555,73 \mu W
 \end{aligned} \tag{9}$$

El cálculo de la autonomía de nuestro sistema en este escenario se muestra en la siguiente expresión:

$$\begin{aligned}
 Autonomía sistema &= \frac{P_{batería} (Wh)}{P_{media}(W)} = \frac{14,4 Wh}{555,73 \mu W} = 25911 h \\
 &= 2,95 \text{ años}
 \end{aligned} \tag{10}$$

Se puede observar que es un tiempo bastante bueno, ya que, durante aproximadamente 3 años, no nos tendremos que preocupar por la batería de nuestro sistema. Además, este tiempo se

puede ver aumentado reduciendo la potencia de radicación como se detalló en la sección 6.1.1.1. Por lo que podemos valorar este resultado como positivo.

Durante todo el proceso ha sido un aspecto que se ha tenido muy en cuenta, y podemos garantizar que hemos obtenido el consumo más óptimo que la tanto la aplicación, como la tecnología, nos permite hoy día.

6.1.2. Tiempo de envío de imagen

En esta sección vamos a analizar el proceso de envío de imagen, descrito en la sección 3.2.4. Haremos una comparación con los resultados teóricos (sección 3.2.4). Del mismo modo, comprobaremos que la implementación del protocolo diseñado (sección 3), cumple las especificaciones.

Empezaremos por el proceso de envío de la imagen desde el nodo central al nodo periférico. En la sección 3.2.2 se detalla este proceso, que abarcará desde el inicio de la transmisión, hasta que esta imagen este almacenada en nuestro dispositivo periférico.

En la Figura 67 podemos observar que transcurre un tiempo de 185 ms para establecer la conexión y un tiempo de 1,05 s para transmitir la imagen. Estos resultados difieren de los calculados teóricamente en la sección 3.2.4.

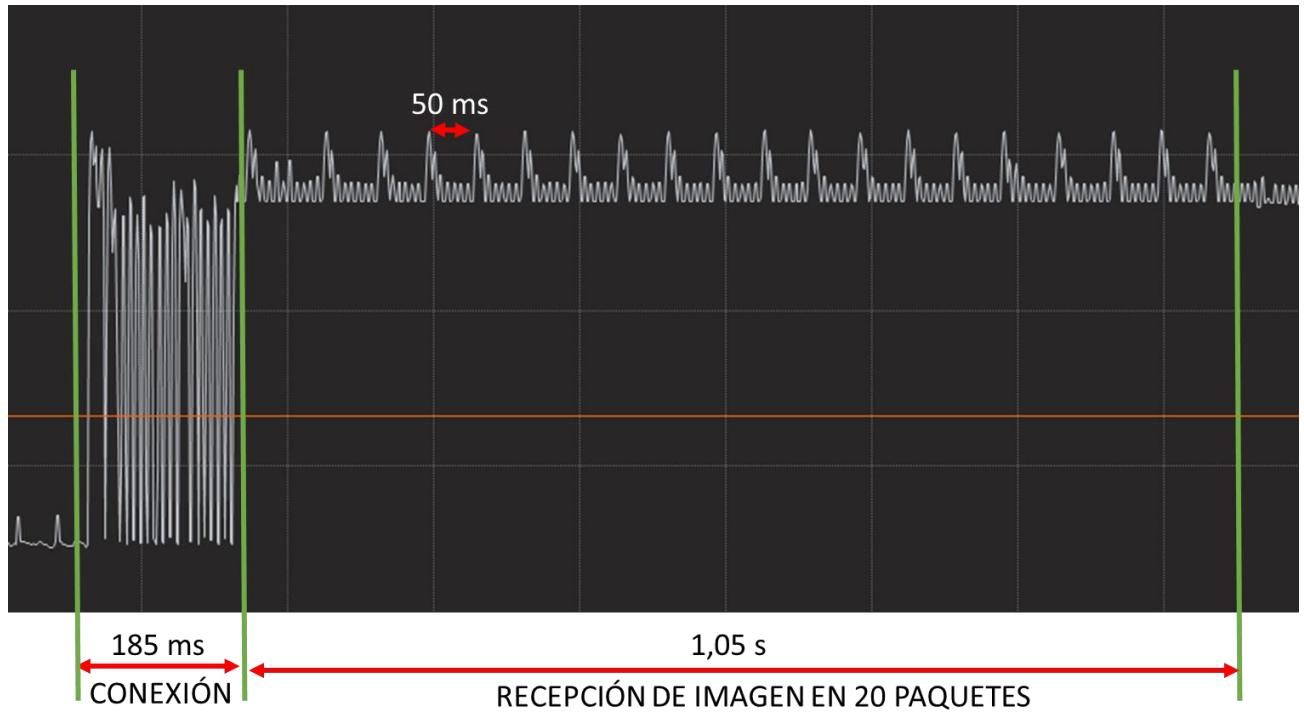


Figura 67 Señal transmisión de imagen

A continuación, vamos a explicar a que se deben estas diferencias. Podemos ver que el tiempo entre paquetes es de 50 ms. Queremos aclarar este concepto para no confundirlo con el intervalo de conexión de 7,5 ms. En la Figura 68 se muestra una captura de la sesión de envío de imagen, obtenida con el *sniffer* (sección 4.4) y *Wireshark*.

Se puede observar que el intervalo de conexión es de 7,5 ms (0x06), pero como se detalló en la sección 3.2.4, se necesitará más de un intervalo de conexión para enviar un paquete, ya que utilizamos la operación *write*. De hecho, podemos observar en la columna de *Time* que el tiempo de envío entre dos paquetes es 7,5 ms.

Filter: btle							Expression...	Clear	Apply	Save
No.	Time	Source	Destination	Protocol	Length	Info				
24	10.273226000	Slave	Master	L2CAP	42	Rcvd Connection Parameter Update Request				
25	10.278913000	Master	Slave	LE LL	35	Control Opcode: LL_FEATURE_RSP				
26	10.284688000	Master	Slave	LE LL	35	Control Opcode: LL_FEATURE_RSP				
27	10.290395000	Slave	Master	LE LL	35	Control Opcode: LL_FEATURE_RSP				
28	10.296341000	Master	Slave	LE LL	38	Control Opcode: LL_CONNECTION_UPDATE_REQ				
29	10.302181000	Slave	Master	LE LL	35	Control Opcode: Unknown				
30	10.308112000	Master	Slave	SMP	37	Rcvd Pairing Request: Bonding, No MITM,]				
31	10.313400000	Slave	Master	LE LL	26	Empty PDU				

< |

```
# Frame 28: 38 bytes on wire (304 bits), 38 bytes captured (304 bits) on interface 0
# Nordic BLE sniffer meta
# Bluetooth Low Energy Link Layer
    Access Address: 0x492345d7
    Data Header: 0x0c13
    Control opcode: LL_CONNECTION_UPDATE_REQ (0x00)
    Window Size: 2
    Window offset: 0
    Interval: 6
    Latency: 0
    Timeout: 3200
    Instant: 9
```

Figura 68 Captura wireshark intervalo de conexión

Si ampliamos más la señal (Figura 69), podemos apreciar que son necesarios 7 intervalos para la transmisión de un paquete, lo que hace el tiempo de 50 ms. Este tiempo, multiplicado por los 20 paquetes en los que dividimos nuestras imágenes (4,7 Kbytes / 244 Bytes), dan como resultado los 1,09 segundos que necesitamos para el envío de la imagen.

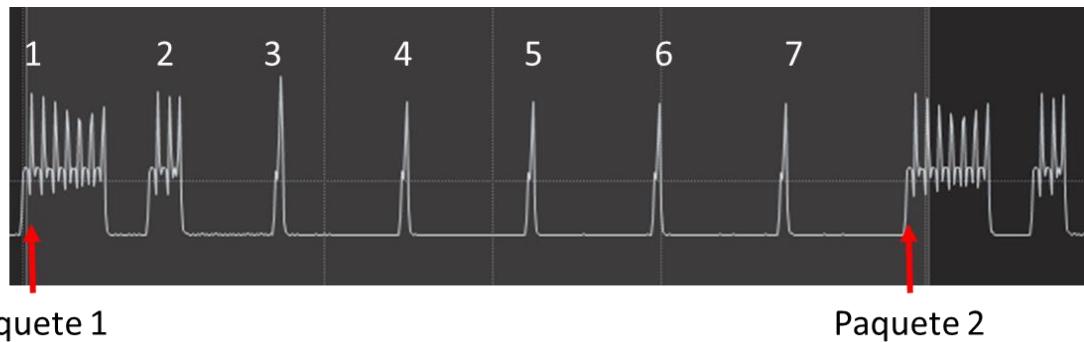


Figura 69 Señales entre dos paquetes

6.2. Presupuesto

En este apartado haremos un estudio económico de nuestro sistema. Empezaremos por la estimación de los costes que tendría una implementación comercial. Para ello, vamos a suponer que queremos utilizar una red con 10 pantallas y un nodo central. En la Tabla 6 se muestra este presupuesto.

Hemos de mencionar que, aunque durante todo el proyecto hemos trabajado con el kit de desarrollo, en esta estimación económica mostramos el precio del *System on chip* que se utilizaría a la hora de lanzar al mercado este producto.

Tabla 6 Presupuesto modelo comercial

Componente	Descripción	Cantidad	Coste
ARTIK-520	Nodo central de nuestra red	1	48,38 € [43]
ARTIK-020	Nodo periférico de nuestra red	10	5,74 € [44]
Pervasive Display 2,87"	Pantalla tinta Low Energy	10	29,60 € [45]
		TOTAL	401,78 €

En la Tabla 7 se muestra una estimación del coste de desarrollo de nuestro producto, tanto de las herramientas como de la mano de obra.

Tabla 7 Presupuesto de diseño y desarrollo

Concepto	Cantidad	Coste
ARTIK-520 Kit desarrollo	1	85,40 € [46]
ARTIK-020 Kit desarrollo	1	84,99 € [47]
Pervasive Display 2,87"	1	29,60 € [45]
Bluefruit LE Sniffer nRF51822	1	20,46 € [48]
Horas Ingeniero Junior	20h/sem*12 meses	10.240,00 €

	TOTAL	10.460,45 €
--	--------------	-------------

6.3. Conclusiones y trabajo futuro

Iniciamos esta sección para hacer una valoración general de nuestro proyecto y establecer las posibles líneas de trabajo futuro para la continuación de éste.

Empezamos este proyecto enfrentándonos a un reto principal: el envío de “grandes” cantidades de datos a través de *bluetooth low energy*, un área con poca exploración.

Prueba de ello, es la constante actualización de las versiones de SDK por parte de los fabricantes que se han ido produciendo durante el desarrollo de este, en los que hemos vivido más de dos actualizaciones de librerías importantes. Durante todo el proyecto hemos tenido que ir adaptándonos a estos cambios e introduciéndolos en nuestras versiones, evolucionando al mismo ritmo que los fabricantes.

Aun así, hemos podido concluir nuestra tarea con bastante éxito y muy satisfechos, ya que hemos sido capaces de reducir el tiempo de envío de las primeras versiones (en torno 30 s) a unos 1 s, lo que ha supuesto una reducción del tiempo de envío del 96,6 %.

Además, hemos sabido combinar la línea de reducción de tiempos sin aumentar el consumo de nuestro sistema, siendo muy conscientes de ello. Cada decisión de diseño se ha tomado con mucha conciencia en este aspecto, pudiendo llegar a afirmar sin lugar a dudas, que nuestro sistema es lo más eficiente energéticamente que la tecnología permite, y sin duda, un producto revolucionario en este aspecto.

Destacar la robustez y confianza que tenemos en nuestro sistema, ya que, desde el primer momento, hemos diseñado desde cero nuestros programas, entendiendo y analizando cada paso que dábamos. Esto ha hecho que el camino haya sido un poco más lento, pero sin duda, mucho más fiable, lo que nos brinda una confianza en nuestro sistema abismal. Del mismo modo que antes, podemos afirmar que sabemos lo que ocurre en nuestro sistema en todo momento sin lugar a dudas.

Por lo que, como se puede comprobar, la valoración del proyecto es más que positiva, tanto a nivel de resultados como de sensaciones.

Pasamos ahora a comentar las principales líneas de desarrollo en las que se podría enfocar nuestro producto:

- Seguridad: la seguridad es un apartado que hemos decidido incorporar en futuras innovaciones de nuestro sistema. Puede parecer que al tratarse de una aplicación de muestra de información este aspecto no es crítico. Pero como se vio en la sección 2.1, con la revolución de *IoT* y la interconexión de todos los dispositivos, es realmente importante pararse a analizar este punto. Aunque no suponga un riesgo directo a nuestra aplicación, podría comprometer a otras con las que esté interconectadas. Una de los primeros pasos que se podrían dar para avanzar en esta dirección, es el de dotar a nuestros equipos de contraseñas de acceso.



Figura 70 Seguridad IoT

- Tamaño de pantallas: en esta primera implementación del proyecto se ha utilizado un solo tamaño de pantalla. Para futuras versiones, se pretende incorporar la compatibilidad del sistema con distintos tamaños de pantalla, por lo que todo el proyecto se ha diseñado para facilitar esta inclusión.



Figura 71 Distintos tamaños de pantalla

- Funcionalidades: otra de las direcciones en las que se podría avanzar, sería la de añadir funcionalidades a nuestro servidor, de manera que nos permita seleccionar más parámetros de nuestro servidor. Estas funciones podrían ser la selección del tiempo de escaneo o la selección de la potencia de transmisión de los dispositivos. Durante todo el desarrollo se ha tenido esto en cuenta, y se ha diseñado todos los parámetros conscientemente para facilitar esta ampliación de funcionalidades.

- Empaquetado: como venimos mencionando, la implementación de todo el sistema se ha ido haciendo sobre kits de desarrollo. Para una última versión comercial, para reducir espacio y coste, se podría plantear el uso de *System-On-Chip* con la integración en PCB, añadiendo los subsistemas de alimentación y protección. Del mismo modo se podría diseñar una estructura de empaquetado y protección de componentes.



Figura 72 Packaging E-Ink

Referencias

- [1] M. Conner, Sensors empower the "Internet of Things".
- [2] Bluetooth. Available: <https://es.wikipedia.org/wiki/Bluetooth>.
- [3] J. A. Arévalo, «El "Internet de las cosas... ",» *DesiderataLAB*.
- [4] K. L. In Lee, «The Internet of Things (IoT): Applications, investments, and challenges for enterprises,» *Science Direct*.
- [5] D. Evans, «IBSG de Cisco,». Available:
https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf.
- [6] «Gartner,». Available: <https://www.gartner.com/en>.
- [7] I. d. l. cosas, «Wikipedia,». Available:
https://es.wikipedia.org/wiki/Internet_de_las_cosas.
- [8] G. Osborne. Available: https://es.wikipedia.org/wiki/George_Osborne.
- [9] «EIoT,». Available: <http://www.summitdata.com/blog/understanding-the-enterprise-internet-of-things/>.
- [10] «E-Ink,». Available: <https://www.eink.com/about-us.html>.
- [11] Wikipedia, «Pantallas Tinta Electronica,». Available:
https://es.wikipedia.org/wiki/Tinta_electr%C3%B3nica.

- [12] R. Heydon, «Bluetooth low Energy: The developer's Handbook,» Prentice Hall.
- [13] F. T. M. S. R. S. y. D. F. Jacopo Tosi Orcid, Review, Performance Evaluation of Bluetooth Low Energy: A Systematic.
- [14] Texas, «Texas Instruments,». Available:
<http://www.ti.com/lit/ug/swru271f/swru271f.pdf>.
- [15] I. t. B. L. Energy, «adafruit,». Available:
<https://learn.adafruit.com/introduction-to-bluetooth-low-energy/introduction>.
- [16] S. Labs, «Bluetooth Smart Software API Reference Manual,» [En línea]. Available: <https://www.silabs.com/documents/login/reference-manuals/bluetooth-api-reference.pdf>.
- [17] T. Instruments, «Bluetooth Low Energy Software Developer's guide,». Available: <http://www.ti.com/lit/ug/swru271g/swru271g.pdf>.
- [18] B. services, «<https://www.bluetooth.com/specifications/gatt/services>,».
- [19] B. SIG, «Profiles specifications,». Available:
<https://www.bluetooth.com/specifications/adopted-specifications>.
- [20] B. SIG, «Generic Access Service,» Available:
https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.service.generic_access.xml.
- [21] B. SIG, «Device Information Service,». Available:
https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=244369.
- [22] S. Labs, «Throughput with Bluetooth Smart technology,». Available:
https://www.silabs.com/community/wireless/bluetooth/knowledge-base.entry.html/2015/08/06/throughput_with_blue-Wybp.

- [23] P. Display, «Application Note for 2.87" Aurora Mb (B231),» . Available: www.pervasivedisplays.com/LiteratureRetrieve.aspx?ID=234394.
- [24] S. ARTIK, «ARTIK 020 Bluetooth Module Data Sheet,» . Available: <https://developer.artik.io/downloads/52e56827-4584-4838-973b-677a9827734f/download>.
- [25] ARTIK-520, «ARTIK 520 Datasheet,» . Available: <https://developer.artik.io/downloads/e249f7ed-2307-4376-8741-935112a2e7d7/download>.
- [26] P. Display, «Product Specifications 2.87" TFT EPD Panel,» . Available: www.pervasivedisplays.com/LiteratureRetrieve.aspx?ID=234374.
- [27] P. Displays, «EPD Extension Kit Gen 2 (EXT2),» . Available: http://www.pervasivedisplays.com/kits/ext2_kit.
- [28] A. Developer, «Cortex A7,» . Available: <https://developer.arm.com/products/processors/cortex-a/cortex-a7>.
- [29] A. 5. Brief, «Samsung Artik 520,» . Available: <https://developer.artik.io/downloads/188f2da0-5669-402d-94d6-76d944ae3f2c/download>.
- [30] BlueFruit, «Bluetooth Low Energy nRF51822,» . Available: <https://www.adafruit.com/product/2269>.
- [31] Wireshark.. Available: <https://www.wireshark.org/>.
- [32] N. Semiconductor, «nRF Sniffer,» . Available: <https://www.nordicsemi.com/eng/Products/Bluetooth-low-energy/nRF-Sniffer>.
- [33] S. Labs, «Simplicity Studio,» Available: <https://www.silabs.com/products/development-tools/software/simplicity-studio>.

- [34] S. Labs, «Silicon Labs Bluetooth C Application Developer's Guide,». Available: <https://www.silabs.com/documents/login/user-guides/ug136-ble-c-soc-dev-guide.pdf>.
- [35] P. Display, «EPD Extension Kit Gen 2 (EXT2),». Available: http://www.pervasivedisplays.com/kits/ext2_kit.
- [36] S. Labs, «EFM32 Ultra Efficient Energy Modes,». Available: <https://www.silabs.com/products/mcu/32-bit/efm32-energy-modes>.
- [37] Samsung, «ARTIK IDE,». Available: <https://developer.artik.io/documentation/artik-05x/getting-started/prepare-ide.html>.
- [38] S. A. Modules.. Available: <https://developer.artik.io/documentation/artik-05x/getting-started/prepare-ide.html>.
- [39] «JSON introducing,». Available: <https://www.json.org/>.
- [40] S. Labs, «Optimizing Current Consumption in Bluetooth Low Energy Devices,». Available: https://www.silabs.com/community/wireless/bluetooth/knowledge-base.entry.html/2017/08/25/optimizing_currentc-C4uw.
- [41] B. P. d. L. 4000mAh, «Amazon,». Available: https://www.amazon.es/Pol%C3%ADmero-compatible-MediaPad-Springboard-sustituye/dp/B00CFFQ81S/ref=pd_sbs_107_3?encoding=UTF8&pd_rd_i=B00CFFQ81S&pd_rd_r=6441ea8b-7b7d-11e8-8726-d91f82847179&pd_rd_w=KfEER&pd_rd_wg=d0Dbl&pf_rd_i=desktop-dp-sims&pf_rd_m=A1AT7Y.
- [42] S. Labs, «EFR32BG1 Blue Gecko Bluetooth® Low Energy,». Available: <https://www.silabs.com/documents/login/data-sheets/efr32bg1-datasheet.pdf>.

- [43] M. Electronics, «Samsung ARTIK 520,». Available: <https://www.mouser.es/ProductDetail/Samsung-ARTIK/SIP-005AYS001?qs=Mv7BduZupUgCPFrWapUh%2fQ%3d%3d>.
- [44] M. Electronic, «ARTIK-020-AV2,». Available: <https://www.mouser.es/ProductDetail/Samsung-ARTIK/ARTIK-020-AV2?qs=sGAEPiMZZMve4%2fbfQkoj%252bISGNdi489jOCDYpohV1SRY%3d>.
- [45] DigiKey, «Pervasive Display,». Available: <https://www.digikey.com/product-detail/en/pervasive-displays/E2287FS091/E2287FS091-ND/7596295>.
- [46] Arrow, «Artik 520,». Available: <https://www.arrow.com/en/products/sip-kitnxb001/samsung-electronics>.
- [47] Digi-Key, «ARTIK 020 BLUETOOTH SMART KIT,». Available: <https://www.digikey.com/product-detail/en/samsung-semiconductor-inc/SIP-KITSLF001/1683-1005-ND/6231212>.
- [48] M. Electronic, «Bluefruit LE Sniffer,». Available: <https://www.mouser.es/ProductDetail/485-2269>.
- [49] RFID, «Wikipedia,» 2005.. Available: <https://es.wikipedia.org/wiki/RFID>.
- [50] J. O. a. J. P. C. Gomez, «Overview and Evaluation of Bluetooth Low Energy,» *MDPI*.
- [51] K. Townsend, «Introduction to Bluetooth Low Energy,» *adafruit learnig system*.
- [52] G. characteristics,
«<https://www.bluetooth.com/specifications/gatt/characteristics>,»

- [53] Cyprees, «Bluetooth Low Energy,». Available:
<http://www.cypress.com/file/179851/download>.
- [54] «Pervasive Display,». Available: <http://www.pervasivedisplays.com/>.
- [55] B. SIG, «Generic Access Service,». Available:
https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.service.generic_access.xml.
- [56] S. LABS, «Bluetooth Smart Software API Reference,». Available:
<https://www.silabs.com/documents/login/reference-manuals/bluetooth-api-reference.pdf>.
- [57] R. Heydon, Bluetooth low Energy: The developer's Handbook.
- [58] «Samsung Artik Modules,». Available: <https://developer.artik.io/>.