Proyecto Fin de Grado
Ingeniería de las Tecnologías de Telecomunicación

Red de sensores Waspmote

Autora: Gloria Martínez Muñoz
Tutor: José Manuel Fornés Rumbao

Dep. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2016

Proyecto Fin de Grado
Ingeniería de las Tecnologías de Telecomunicación

Red de sensores Waspmote

Autora:
Gloria Martínez Muñoz

Tutor:
José Manuel Fornés Rumbao
Profesor titular

Dep. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2016

Proyecto Fin de Grado: Red de sensores Waspmote

Autora: Gloria Martínez Muñoz

Tutor: José Manuel Fornés Rumbao

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2016

El Secretario del Tribunal

A mis abuelos

Agradecimientos

Este trabajo de fin de grado supone el final de una etapa muy especial e importante para mí. A pesar de las dificultades, prevalecen los buenos momentos, las personas y las experiencias y enseñanzas que me llevo de ella. El llegar hasta este punto no habría sido posible sin los docentes, desde mis maestros de primaria hasta los profesores que me han instruído estos últimos años. Me gustaría agradecer a cada uno de ellos y a mi tutor, por darme la oportunidad de realizar este proyecto.

Por supuesto, dar las gracias a mi familia, por apoyarme en todo momento durante estos años tan duros. A mis padres, por sus consejos, por sufrir conmigo los sinsabores y por alegrarse de mis triunfos sabiendo que también son los suyos; a mi hermano, por preocuparse por mí de esa forma especial en que lo hace; a mi abuela, por intentar ayudarme a su manera; a Benjamín, por su paciencia infinita.

Por útlimo me gustaría agradecer la ayuda que durante estos años me han brindado mis compañeros de clase, ya convertidos en amigos. Especialmente a Lourdes, juntas le hemos puesto siempre el toque de humor a todo.

Gloria Martínez Muñoz

Sevilla, 2016

Resumen

La preocupación por el medio ambiente y la limitación de los recursos fósiles han fomentado la generación de energía a partir de fuentes renovables. Sin embargo, la producción de energía mediante plantas solares aún no aporta costes que compitan con las tecnologías tradicionales de las plantas de generación a partir de recursos fósiles. Una pequeña parte de los costes se invierten en sistemas de control y puesta a punto, que tienen un impacto importante en la eficiencia de la producción y los gastos de explotación y mantenimiento de la planta.

Desde hace aproximadamente una década, la tecnología basada en redes de sensores inalámbricos (WSN) se ha introducido con fuerza en el mercado. Estos dispositivos están diseñados tanto para uso doméstico como industrial, aportando gran flexibilidad al despliegue de redes para transmisión de datos en sistemas de monitorización y control.

El sistema desarrollado en este proyecto permite obtener, gracias a una red de sensores, datos necesarios para la predicción de la radiación solar a corto plazo. Esto, unido a una serie de elementos, permitirá desarrollar un sistema que prediga el movimiento de las nubes por el cielo y la irradiancia solar, a fin de automatizar el movimiento de las placas de una central solar térmica.

El proyecto completo busca mejorar la eficiencia de las centrales solares y permitirá actuar sobre el sistema para un mayor control y aprovechamiento de las instalaciones. Se llevará a cabo su aplicación en escenarios reales.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

1.1 Objetivos

En los últimos años se han desarrollado numerosos sistemas de aprovechamiento de la energía solar debido a la preocupación por el medio ambiente, la contaminación y la limitación de los recursos fósiles, que son agotables. No obstante, la generación de este tipo de energía aún no compite con las tecnologías tradicionales que, aunque menos limpias y más dañinas con el entorno, ofrecen beneficios mucho mayores.

Una pequeña parte de las pérdidas generadas durante la fabricación y puesta a punto de una instalación solar típica es ocasionada por los sistemas de control, que representan un papel fundamental en la eficiencia de la producción y los costes de explotación y mantenimiento de la planta. En este sentido, la Automática se convierte en una herramienta indispensable para lograr la integración de la energía solar con el resto de fuentes de generación tradicionales.

La cantidad de radiación solar recibida en una determinada localización tiene una componente previsible, debido a la variación de la misma con las diferentes horas del día, y componentes aleatorias que perturban el funcionamiento de las plantas solares. Una de estas perturbaciones es ocasionada por la presencia de nubes, por lo que resulta fundamental anticiparse a las condiciones climatológicas para lograr una operación eficiente de estas centrales. En la mayoría de ellas, este problema se trata de manera manual, controlando y corrigiendo el funcionamiento en base a la experiencia e intuición de los operarios de la planta.

En el presente trabajo de fin de grado se desarrolla uno de los puntos del proyecto CODISOL, el cual comprende el estudio y desarrollo de un sistema de detección de cúmulos nubosos y predicción de la radiación solar a través de técnicas de control y estimación distribuídas en una red de sensores. Esto permitirá conocer la fracción de campo solar que se verá afectada por la disminución de radiación debido al paso de las nubes y, por tanto, lograr una mayor eficiencia, fiabilidad y aprovechamiento de las instalaciones solares. Los métodos desarrollados podrían también ayudar a introducir la energía solar a algunos mercados de comercialización de energía en los que actualmente no pueden competir. Esto es debido a la gran incertidumbre en la prospectiva de generación futura a corto y medio plazo.

Concretamente, este proyecto tiene como objetivo el estudio y la implantación de la red de sensores que permita la monitorización remota y predicción de la irradiancia solar a corto plazo. Las condiciones de radiación pueden variar sensiblemente de unos puntos a otros en la central, aunque existe una correlación espacial y temporal evidente de este parámetro físico. Es esta correlación la que se pretende explotar: a partir de los datos medidos por los distintos nodos de la red de sensores, se desarrollarán modelos de predicción que contribuyan al objetivo final del proyecto.

El sistema se implantará en dos plantas solares ya disponibles. La primera situada en el departamento de Ingeniería de Sistemas y Automática y la segunda en una de las instalaciones que Abengoa Solar tiene en la Plataforma SOLUCAR.

El Sistema deberá permitir el seguimiento del mes, día y hora en que se realizan las medidas, con la posibilidad de almacenamiento local de los datos en una base de datos para su posterior exportación, de manera que las medidas realizadas sean de utilidad para el resto de grupos que trabajan en el proyecto CODISOL.

1.2 Estructura del trabajo de fin de grado

A continuación, se detalla de manera resumida el contenido de los distintos capítulos y anexos que componen este trabajo:

- Capítulo 1 – Introducción

  Se trata del capítulo actual, en el cual se resumen los motivos que han propiciado el desarrollo del proyecto y los objetivos que con él se persiguen.

- Capítulo 2 – Estado del Arte

  En este capítulo se describen conceptos elementales y conocimientos teóricos, necesarios para comprender este documento y el proyecto que se desarrolla.

- Capítulo 3 – Caso de Estudio

  Esta sección detalla en profundidad el marco donde se quiere aplicar la red de sensores desarrollada.

- Capítulo 4 – La tecnología Zigbee

  Este capítulo profundiza y expone la tecnología de comunicación fundamental que se desarrolla en el proyecto y que permite la comunicación entre los diversos nodos que componen la red de sensores creada.

- Capítulo 5 – Selección de microcontrolador

  Se analizan en esta sección las distintas opciones disponibles en el mercado para construir el sistema requerido, y se justifica la elección realizada.

- Capítulo 6 – Arquitectura hardware

  En esta sección se detallan todos los elementos que conforman cada nodo integrante de la red de sensores, así como sus características y funcionalidades.

- Capítulo 7 – Arquitectura software e implementación

  En este capítulo se muestra en detalle el funcionamiento de los components que forman la red de sensores y su configuración, se explica la solución a la que se ha llegado, la red construída y la programación implementada.

- Capítulo 8 – Resultados obtenidos

  En esta última parte se exponen los resultados logrados a través de la experimentación y puesta en marcha de la red de sensores del proyecto y se comentan los problemas encontrados a lo largo de la realización del mismo, detallando como se han solucionado.

  Se adjunta también el presupuesto invertido en la realización del proyecto.

- Anexo A – Código del programa implementado en los nodos finales

- Anexo B – Código del programa implementado en los nodos enrutadores

- Anexo C – Código del script que vuelca los datos en un fichero

2 ESTADO DEL ARTE

En la actualidad, existen millones de sensores que conforman numerosos sistemas y dispositivos electrónicos (sensores de humedad en estaciones metereológicas, giroscopios en móviles y tabletas, etc.).

Cuando estos dispositivos comenzaron a conectarse a Internet se formó lo que hoy en día se conoce como *Internet de las Cosas*, concepto que se detallará más adelante.

Durante los últimos años, debido a esta tendencia, ha surgido una nueva generación de sensores que son independientes de un sistema electrónico concreto y que incorporan en un mismo dispositivo el transductor (de la o las varables que interesan medir), la alimentación del propio dispositivo y un módulo de comunicación dotado de cierta inteligencia propia, capaz de organizarse a sí mismo y de interconectarse de forma inalámbrica con otros nodos semejantes a él.

Como resultado de este despliegue han surgido las denominadas *redes de sensores inalámbricos* (WSN, Wireless Sensor Networks), que consisten en redes constituídas por multitud de sensores individuales que intercambian información entre sí y/o con un nodo central, sin necesidad de cables y mediante un protocolo de comunicación preestablecido.

Las WSN comenzaron siendo usadas en aplicaciones militares, pero en la actualidad se usan en todo tipo de áreas: la domótica, el sector de la salud, la industria de producción en masa o en el control de la infraestructura urbana. Gracias a esta creciente popularidad y a la búsqueda de un diseño eficiente, ha surgido un área de investigación en los últimos años, debido al gran potencial de estas redes para cubrir grandes áreas con un coste relativamente bajo.

A fin de comprender la situación tecnológica actual de la que se parte, en este capítulo van a tratarse los siguientes conceptos:

- Internet de las cosas.
- Redes inalámbricas.
- Programación remota.

2.1 Internet de las cosas.

El Internet de las Cosas [1] o *Internet of Things*, en inglés, por cuyas siglas suele ser referido normalmente como IoT, es una idea basada en la existencia de una capa de conectividad digital para cosas existentes. Con el término “cosas” nos referimos a cualquier participante real o virtual, ya sean objetos cotidianos, seres humanos, datos virtuales o agentes de software inteligente.

Se trata de la construcción de un mundo en el que cada objeto tiene una identidad virtual propia y capacidad potencial para integrarse e interactuar de manera independiente en la red con cualquier otro individuo, ya sea una máquina o un humano. El objetivo que se persigue es crear un entorno en el que la infomación básica proveniente de cualquiera de los actores autónomos conectados pueda ser compartida eficientemente con otros, en tiempo real.

Con la aparición del internet de las cosas, se está llevando a cabo una transformación en la que se están creando

nuevos modelos de negocio, productos y compañías. Se espera que esta idea traiga consigo numerosos beneficios en aspectos como la optimización de la cadena de abastecimiento, efectividad de costos, mejoras en la experiencia de consumidores, beneficios en aspectos de seguridad y servicios de emergencia, etc.

Obviamene, para que la idea del internet de las cosas sea una realidad, es necesaria una evolución en la tecnología que soporta Internet (por ejemplo, en el volumen de información que se intercambia simultáneamente), además de cambios en el paradigma de lo cotidiano.

Siempre y cuando las empresas, los gobiernos, los organismos de normalización y las universidades trabajen de manera conjunta, el Internet de las Cosas seguirá avanzando de la manera en que lo hace.

2.1.1 Historia del IoT.

El IoT, tal y como hoy en día se conoce, surgió en el Instituto Tecnológico de Massachusetts (MIT). Concretamente en el Auto-ID Center, un grupo fundado en 1999, que realizaba investigaciones sobre la identificación por radiofrecuencia en red (RFID) y las tecnologías de sensores emergentes.

Tras esta aparición, surgen diferentes definiciones sobre el concepto de Internet de las Cosas, aunque la más extendida es la siguiente:

> *"IoT es sencillamente el punto en el tiempo en el que se conectaron a Internet más cosas u objetos que personas"*

En 2003 había aproximadamente 6300 millones de personas en el planeta, y había 500 millones de dispositivos conectados a Internet, lo que significa que había una media de menos de un dispositivo (0,08) por persona. Por tanto, la cita anterior no puedía ser considerada válida para el año 2003, pues la cantidad de dispositivos conectados era relativamente pequeña.

Sin embargo, en 2010 el crecimiento explosivo de los smartphones y las tablets elevó a 12500 millones la cantidad de dispositivos conectados a Internet. A su vez, la población mundial se elevó a 6,8 mil millones. El número medio de dispositivos conectados pasaba a ser superior a 1 por persona (1,84 exactamente), por primera vez en la historia.

Profundizando aún más en estas cifras, podría estimarse que el Internet de las Cosas "nació" en algún momento entre los años 2008 y 2009, tal y como se muestra en la siguiente imagen:

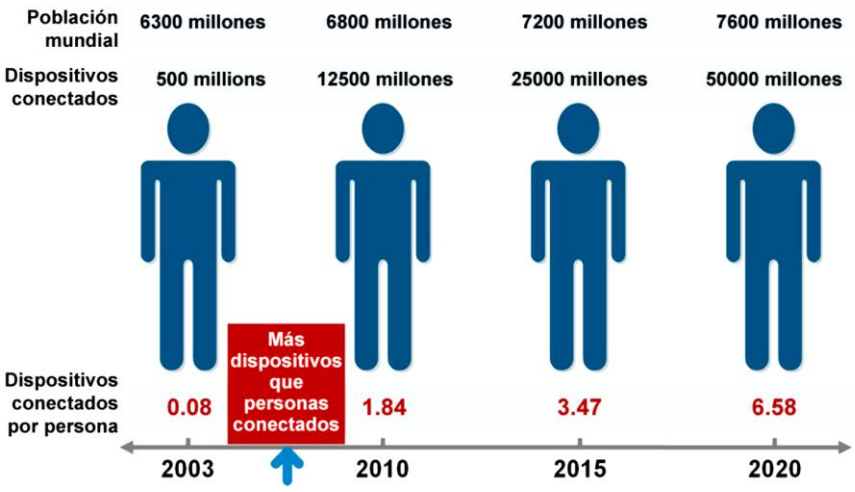

Ilustración 1: Fuente: IBSG de Cisco

Nada permanence estático, sobre todo cuando hablamos de internet, y el Internet de las Cosas está en pleno proceso de desarrollo. El IBSG de Cisco estima que habrá unos 50 000 millones de dispositivos conectados en 2020.

2.1.2 Desarrollo del IoT

El internet de las cosas está formado por multitud de redes diferentes y distintos fines, por ejemplo, para controlar el funcionamiento del motor en los automóviles, en edificios para el control del sistema de seguridad, calefacción e incluso iluminación, etc. Por ello, el IoT puede considerarse una red de redes. De esta forma, a medida que evoluciona el internet de las cosas, las redes estarán conectadas con la incorporación de funciones de análisis, seguridad y administración, para permitir que el internet de las cosas sea una herramienta aún más poderosa.

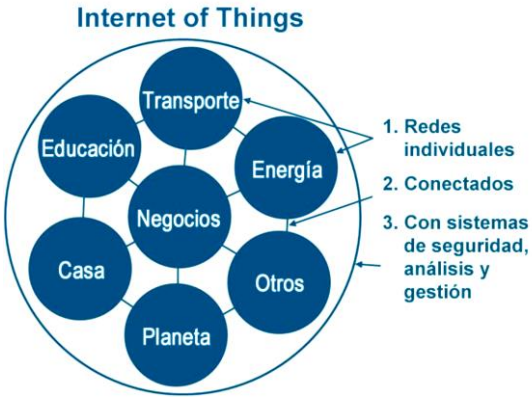

Ilustración 2: Fuente: IGSB de Cisco

Es necesario mencionar la importancia que ha cobrado el internet de las cosas en la sociedad durante estos últimos años. Desde su aparición, Internet (entendido como la capa física o red formada por switches, routers y otros equipos, y no como la Web) había seguido un proceso continuo de desarrollo y mejora, pero podría decirse que no con muchos cambios.

Sin embargo, el internet de las cosas ha sido la primera evolución real de Internet, lo que está provocando su expansión hacia lugares inalcanzables hace algunos años, y que se convierta en algo sensorial. Por ello están apareciendo aplicaciones revolucionarias que tienen el potencial de mejorar sustencialmente la manera de vivir, aprender, trabajar y entretenerse de las personas.

Los datos son la materia prima que se procesa para que las personas consigan información. A partir de dicha información y de su difusión y compartición se adquieren los conocimientos, que unidos a la experiencia hacen que nazca la sabiduría. El IoT aumenta significativamente la cantidad de datos a procesar. Este hecho, unido a la capacidad de Internet para comunicar dichos datos, permiten a la sociedad avanzar a pasos agigantados, por lo que hoy en día es de vital importancia su existencia para la progresión humana.

2.1.3 Retos y características del IoT

El concepto de IoT está principalmente impulsado por los continuos progresos en microelectrónica y las tecnologías de redes. Los retos científicos y técnicos requieren distintas compentencias a distintos niveles, que son:

- **Nivel tecnológico**: engloba los retos relacionados con la integración de objetos de red inteligentes bajo fuertes restricciones de energía y medioambiente.
- **Nivel de comunicación y redes**: abarca los retos relacionados con las redes masivas, seguras y dinaámicas y los servicios de provision ubícua.
- **Nivel de inteligencia**: agrupa todos los retos relacionados con la fusion de datos y el descubrimiento de

servicios donde los datos recolectados por nodos individuales son requeridos por usuarios distribuídos.

En el nivel tecnológico se desarrolan las funcionalidades que permiten la iteracción entre las "cosas", tales como la identificación, la toma de medidas, el almacenamiento, la actuación y otras actividades relacionadas con la interconexión. La conexión entre el mundo real y el digital require la capacidad por parte del mundo digital de tomar medidas del mundo real y actuar en consecuencia. Existen numerosas tecnologías que soportan el IoT. Están basadas particularmente en dispositivos inalámbricos, pequeños y baratos, con capacidad para tomar medidas, comunicarse, actuar y procesar información y señales de forma avanzada.

Las características fundamentales de la tecnología IoT pueden resumirse en las siguientes:

El IoT es una solución global y en tiempo real

Por una parte, la tecnología IoT está basada e Internet o en otras redes basadas en áreas amplias. Es por ello que el alcance del internet de las cosas no tiene límites físicos.

Por otra parte, la comunicación de datos dse realiza en tiempo real o casi de ese modo, lo que lo diferencia de otras sistemas web y bases de datos tradicionales.

Es una tecnología inalámbrica que puede proporcionar datos acerca de los alrededores.

Esta característica incrementa significativamente la riqueza de información que puede obtenerse. Incluso pueden recogerse medidas en entornos y situaciones en las que no resultaría sencillo proceder sin este tipo de tecnología, tales como el seguimiento de animales o en situaciones climatológicas difíciles.

Es capaz de monitorizar remotamente el ambiente y rastrear objetos.

Combinando el uso de sensores con otras tecnologías como GPS o la detección de objetos, se puede seguir la pista de cualquier objeto en ambientes interiores o exteriores. Esta visibilidad constante de los objetos monitorizados permite responder instantáneamente a cualquier evento excepcional, distribuyendo la información y compartiéndola a través de múltiples organizaciones y usuarios.

En definitiva, no importa qué definición se tome del internet de las cosas, el requisito básico de éste es proporcionar conectividad directa o indirecta con cualquier "cosa" inteligente habilitada para la red en cualquier momento y desde cualquier lugar.

Comparando esto con la definición de internet ("tener conexión con cualquier ordenador en cualquier momento y desde cualquier lugar"), la única diferencia entre el internet de las cosas e Internet es que los dispositivos finales en el IoT son cualquier cosa inteligente en lugar de solo ordenadores.

2.1.4 Arquitectura orientada a servicios del internet de las cosas

La arquitectura orientada a servicios (Service-oriented architecture - SoA) en la que se basa el internet de las cosas, define tres protagonistas: el productor de servicios, el consumidor y el registro de servicios. Las entidades que participan en una arquitectura SoA están limitadas a un único rol para cada momento en una determinada iteracción.

Los productores publican sus servicios en uno o más registros. Los consumidores usan herramientas de búsqueda en los registros para encontrar el servicio deseado.

Esta arquitectura está diseñada para soportar de forma global múltiples usos, aplicaciones y fuentes de datos. Proporciona una plataforma unificada para recolectar, compartir, procesar y consultar datos a los sensores, y

permitir a éstos unirse y abandonar el sistema sin que afecte al resto de componentes que lo forman.

2.1.5 Limitaciones y barreras para el IoT

A pesar de todos los beneficios que puede aportar el internet de las cosas, y de la cantidad de sistemas desarrollados para hacerlo possible, aún presenta inconvenientes que pueden retrasar su desarrollo.

El potencial completo del IoT podrá desbloquearse cuando las pequeñas redes de cosas conectadas se conviertan en una gran red que se extienda por industrias y organizaciones.

Para ello, desarrolladores e ingenieros aún tienen que afrontar numerosos retos. Los más significativos son la cobertura de Internet alrededor del mundo, la implementación de IPv6, la seguridad, la energía de los sensores y los acuerdos sobre los estándares utilizados.

Cobertura de Internet.

El éxito del internet de las cosas depende completamente de la disponibilidad de internet en cualquier lugar. Hoy en día, alrededor del 50% de la población mundial aún no tiene acceso a internet, por lo que es necesario que Internet siga creciendo. Para hacerlo accesible a todos los lugares del mundo es necesario proporcionar recursos e infraestructuras, y esto es algo que las grandes compañías no llevan a cabo a no ser que haya un claro inicio de generar beneficios. Es por ello que los gobiernos deben entrar en escena y, bajo la situación económica actual, no será una decisión inmediata por parte de ninguno. Sin embargo, se espera que con el tiempo suceda.

Por otra parte, sería conveniente que el acceso a internet fuera gratuíto y estuviera permanentemente disponible, algo mucho más dificil de llevar a cabo. No solo las instalaciones son caras, sino también el mantenimiento que éstas precisan para continuar funcionando.

Implementación de IPv6. Con el crecimiento de uso de dispositivos, el mundo se quedó sin direcciones IPv4 en el año 2010. Este hecho no afectó a los usuarios de Internet pero puede frenar potencialmente el avance del IoT, debido a los miles de millones de direcciones IP públicas que pueden necesitar los sensores. Es por ello que se necesita una implementación definitiva de IPv6, que además facilita la gestión de las redes gracias a capacidades de configuración automáticas y ofrece mejores características de seguridad.

Seguridad de las comunicaciones. Bajo la situación actual del planeta, en el que el terrorismo se convierte en algo cotidiano, el acceso a internet sin ningún tipo de medidas de seguridad supone un gran riesgo. Es necesaria la seguridad tanto en internet como en los dispositivos inalámbricos utilizados, que pueden usarse de forma maliciosa, con propósitos ilegales. El reto en este caso es desarrollar las contramedidas apropiadas. Estas contramedidas pueden consistir en desarrollar detectores de sensores capaces de reconocer redes maliciosas.

Energía de los sensores. Las redes están compuestas por sensores que deben ser autosuficientes para que puedan alcanzar su pleno potencial, ya que cambiar sus baterías de forma manual puede convertirse en algo inviable. Por ello es necesario un mecanismo para que los sensores generen electricidad a través del medio ambiente (vibraciones, luz solar, flujo del aire, etc.).

Además, las baterías y elementos antiguos de los dispositivos contienen materiales tóxicos que suponen un riesgo para el medioambiente y no sería conveniente cambiarlos constantemente, por lo que la reutilización de materiales es la opción más deseable. Con ello puede extenderse la vida de los dispositivos, minimizar la contaminación, dar acceso a los productos a personas que no pueden afrontar los gastos de nuevos dispositivos, etc.

Estándares. Es necesario un mayor avance en lo referido a la seguridad, privacidad, arquitectura y comunicaciones. El IEEE es solo una de las organizaciones que luchan para solucionar estos problemas, asegurándose de que los paquetes puedan enviarse a cualquier tipo de red.

Cabe destacar que a pesar de las barreras anteriormente citadas, ninguna de ellas es insalvable. Son muchas más las ventajas que aporta el internet de las cosas que los inconvenientes.

2.1.6 El futuro del IoT

Las aplicaciones del internet de las cosas son, como ya hemos dicho, infinitas. Abarcan todo tipo de sectores y son aplicables en cualquier ámbito. Por ello cualquier perfil profesional puede beneficiarse de su uso, y es por ello que los proveedores de servicios y otros organismos debe ofrecer aplicaciones que aporten valor tangible a la vida de las personas.

En un futuro no demasiado lejano, se hará uso del internet como un andamio para soportar y transmitir las sensaciones de las personas. El internet de las cosas consistirá en millones de dispositivos de medida electrónicos embebidos: termostatos, medidores de presión, detectores de polución, cámaras, micrófonos, sensores de glucosa, electroencefalogramas, etc. Gracias a ello se podrán sondear y monitorizar desde ciudades hasta especies en peligro de extinción, la atmósfera, conversaciones, carreteras, barcos o incluso nuestros propios cuerpos.

A medida que el IoT vaya evolucionando se irán produciendo cambios que ya hoy en día pueden percibirse:

- Los móviles o dispositivos de usuario serán continuamente bombardeados con información y servicios de la ciudad.
- Será necesario ampliar la infraestructura de comunicaciones de la que se dispone actualmente y mejorar en las tecnologías de Big Data para los servicios en tiempo real.
- La información deberá estar filtrada para que los usuarios solo reciban aquello que pueda interesarles, es decir, información personalizada.
- Cloud Computing: la información deberá estar disponible de manera global y distribuída.

Los dispositivos hablarán directamente con procesos y no entre ellos.

2.2 Redes sensoriales o Wireless Sensor Networks

Las redes de ordenadores se han convertido en una parte fundamental en la vida diaria de gran parte de la población mundial. Facilitan la disponibilidad de información y servicios y los ponen al alcance de cualquier persona en la red, independientemente de su localización. Existen varios tipos de redes de ordenadores, como las redes de área personal (PAN – Personal Area Networks), las redes de área local (LAN – Local Area Networks), las redes de área metropolitan (MAN – Metropolitan Area Networks) o las redes de área amplia (WAN – Wide Area Networks). Todos estos tipos de redes establecen la comunicación entre enlaces cableados, lo que facilita la confiabilidad, pero aporta otro tipo de inconvenientes, como los altos costes de instalación.

Las tecnologías de comunicación inalámbricas proporcionan la solución a estos obstáculos, conectando los dispositivos a través de ondas de radio, infrarrojos o cualquier otro tipo de medio inalámbrico.

Las redes de sensores [2] están formadas por un conjunto de sensores autónomos distribuidos de manera ad hoc, que se comunican por medio de infraestructuras inalámbricas. Dichos sensores trabajan en conjunto para medir fenómenos físicos o medioambientales como la presión, la temperatura o partículas contaminantes, a fin de procesar los datos obtenidos y obtener resultados relevantes.

Se trata de una tecnología emergente que puede considerarse como la tecnología subyacente del internet de las cosas, ya que aportan información muy relevante al resto de nodos que las componen.

Las redes sensoriales no se limitan al aire como medio físico de comunicación, sino que también engloban cualquier medio de transmisión inalambrico como el agua o la luz. Estos medios son utilizados, por ejemplo, en la comunicación basada en señales ópticas o en señales de radiofrecuencia (como es el caso de ZigBee).

2.2.1 Elementos de una red de sensores

Una WSN está formada por multiples nodos, ya sean varios, cientos o miles de ellos, en donde cada uno está conectado a uno o más dispositivos. Los nodos destinados a hacer mediciones son comunmente llamados nodos sensores. Aquellos que se encargan de retransmitir datos reciben el nombre de routers. Por último, existe un

nodo encargado de intercambiar la información con otras redes, llamado estación base o nodo sumidero, similar a una pasarela en las redes de ordenadores tradicionales. El usuario o desarrollador de la red puede controlarla a través de la estación base.

La siguiente imagen ilustra la arquitectura de comunicación de una red de sensores inalámbrica, así como el papel de los diferentes nodos que la componen.

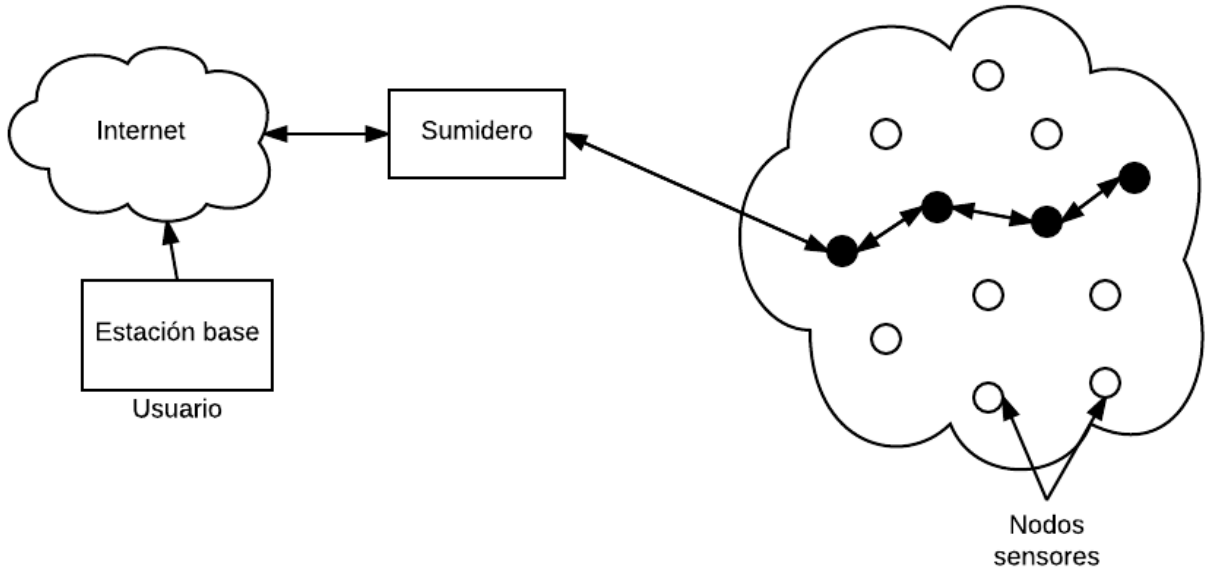

Ilustración 3: Arquitectura de comunicación de una WSN

Los datos son recolectados por los nodos sensores y retransmitidos al nodo sumidero, que se encuentra conectado a internet o a otro tipo de redes. A través de estas otras redes, los datos obtenidos pueden ser finalmente recibidos por alguna aplicación.

Los nodos sensores no tienen por qué tener una localización fija. De hecho, lo más usual es que las redes de sensores se desarrollen en entornos móviles y cambiantes y los dispositivos puedan encontrarse desplegados de manera aleatoria.

Por otra parte, cada nodo sensor o mota está compuesto por una serie de elementos. Más o menos complejos, todos tienen una serie de funcionalidades y componentes en común. Estos elementos son los sensores, un microcontrolador, un transceptor radio y una fuente de alimentación, normalmente una batería.

Cada mote dispone una unidad sensorial compuesta por uno o varios sensores, dispositivos hardware que producen respuestas medibles ante cambios en condiciones físicas. Estos elementos generan una señal analógica continua que es digitalizada por un convertidor digital analógico y para ser posteriormente enviada al controlador. En la unidad de procesamiento hallamos el microcontrolador que realiza tareas, procesa los datos recogidos y controla la funcionalidad del resto de elementos. Además, los datos son almacenados en la memoria del dispositivo, que en algunos casos puede ser extraíble. El transceptor radio es fundamental para la comunicación con otros dispositivos por medio de su antena interna.

Según la complejidad del nodo sensor, también puede disponer de otros elementos opcionales, como son un generador de energía para evitar el reemplazo periódico de la batería, un sistema de localización o un sistema de movilidad.

En la siguiente imagen podemos ver un esquema de la arquitectura interna de un nodo sensor.

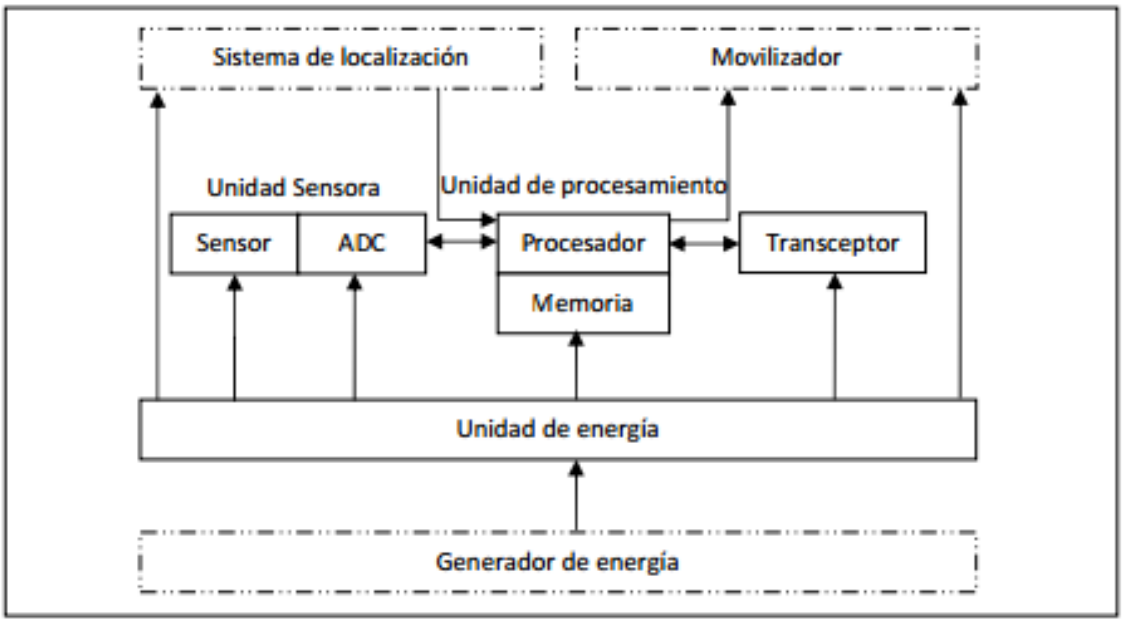

Ilustración 4: Componentes de un nodo sensor

2.2.2 Características de una red de sensores

Una red de sensores se caracteriza por las ventajas que ofrece el trabajo en conjunto de los nodos frente a las medidas proporcionadas por un sensor individual. Algunas de ellas son:

- Rango de sensibilidad mayor: un conjunto de nodos puede abarcar una mayor superficie de operación que uno solo.
- Redundancia: múltiples nodos cercanos unos a otros pueden incrementar la tolerancia a fallos.
- Mejora de la precisión: los nodos pueden colaborar entre sí y combinar sus datos obtenidos para mejorar la precisión de los datos medidos.
- Funcionalidad extendida: los nodos no solo pueden actuar como sensores, también ofrecen servicio de reenvío.

Además de las ventajas que ofrecen, las WSN poseen una serie de características peculiares:

- Están formadas por sensores, los cuales tienen un bajo consumo, memoria limitada y energía restringida debido a su pequeño tamaño.
- Pueden desarrollarse en condiciones ambientales extremas y pueden ser propensos a ataques.
- Aunque se desarrollan de manera ad hoc, los nodos necesitan auto organizarse y auto repararse, además de afrontar constantes reconfiguraciones.

2.2.3 Requisistos necesarios de una red de sensores

A la hora de diseñar una red de sensores, es necesario tener en cuenta una serie de factores y retos de diseño que

deben cumplirse. Algunos de ellos son los siguientes:

- **Auto organización**: los nodos desplegados deben formar una topología que permita establecer rutas por las que mandar los datos obtenidos. Los nodos necesitan conocer su lugar en esta topología, pero en muchos casos resulta inviable que la asignación se realice de forma manual, debido al gran número de nodos. Por ello, es fundamental que sean capaces de formar una topología por sí mismos. Este proceso no solo debe ejecutarse cuando la red se forma por primera vez, sino que las motas deben tener la capacidad de reconfigurarse ante cambios o fallos en el sistema.
- **Heterogeneidad**: los dispositivos desarrollados pueden ser de distinto tipo y aún así ser capaces de colaborar entre ellos. Deben utilizarse los protocolos y mecanismos adecuados para lograr que la comunicación sea exitosa aunque los dispositivos no sean iguales.
- **Procesamiento distribuído**: los algonitmos utilizados deben ser centralizados, así como el procesamiento debe estar distribuído en diferentes nodos. Además, deben utilizarse técnicas de compresión y agregación de datos, a fin de ahorrar costes de transmisión. El uso de una estrategia de comunicación, petición y análisis de datos distribuída es un factor muy importante en una WSN.
- **Coordinación a gran escala**: los sensores necesitan colaborar entre ellos para producir los resultados requeridos. Cuando las redes están formadas por un gran número de nodos, resulta muy complicado y, en algunos casos imposible, llevar a cabo las tareas de mantenimiento y configuración. Por este motivo la escalabilidad es un objetivo fundamental que debe cumplir toda red de sensores.
- Utilización de sensores de manera que produzcan el mayor resultado usando la menor energía posible. El problema de **eficiencia energética** puede abordarse de varias maneras. Por un lado, deben optimizarse tanto el hardware como el software embebido, incluyendo los algoritmos de enrutamiento para minimizar el consumo. Por otro lado, se puede lograr la reducción de la energía optimizando la administración de la misma, tanto a nivel de componentes hardware como a nivel de red.
- **Comunicación de bajo ancho de banda**: los datos deben ser transferidos eficientemente entre los distintos sensores que compongan la red.
- **Comunicación segura**: Los riesgos de seguridad son inevitables en una red de sensores, debido a su naturaleza inalámbrica. Por ello es necesario proporcionar los mecanismos adecuados para proteger a la red de ataques o pérdida de datos. Normalmente los datos viajan encriptados por esta razón. También es necesario proporcionar a los nodos medios de protección contra ataques físicos.
- **Evitar de interferencias**: la coexistencia de la red de sensores con otras redes que trabajen en bandas de frecuencia similares puede provocar la existencia de interferencias que dificulte la comunicación entre los nodos o el envío de datos. Además, es muy común que se produzcan errores debido al ruido que poseen las medidas obtenidas y los fallos que puedan producirse en los nodos. El uso de mecanismos para evitar interferencias es fundamental para lograr una comunicación satisfactoria, pero puede no ser eficiente en redes de gran tamaño debido a las restricciones de capacidades de los sensores.
- **Computación en tiempo real**, realizada de manera rápida cada vez que se genera un nuevo dato.

2.2.4 Campos de aplicación de una red de sensores

Las aplicaciones de redes de sensores inalámbricas pueden ser clasificadas en dos categorías: monitorización remota y localización de objetos móviles. Ambas pueden dividirse en multitud de aplicaciones, tanto para entornos cerrados como abiertos.

Entre las aplicaciones de monitorización de áreas, podemos encontrar la monitorización del medioambiente y hábitats naturales, la agricultura de precisión, en la que se evalúan las condiciones óptimas para producer de manera eficiente y con el menor impacto medioambiental; el control de la climatización en interiores, la supervivencia militar, el establecimiento de alarmas inteligentes, etc.

En cuanto a las aplicaciones para monitorizar objetos, podemos centrarnos en áreas como la monitorización estructural, la eco-fisiología, el mantenimiento basado en condiciones, aplicaciones para diagnósticos médicos, tratamientos de salud o monitorización de señales vitales y anomalías, el mapeado de terrenos urbanos o aplicaciones relacionadas con el automotor.

Exceptuando la monitorización de la localización de objetos móviles, cuyos datos necesitan ser actualizados en tiempo real, el resto de aplicaciones de monitorización remota normalmente se basa en realizar medidas periódicas y mandando las muestras de datos de tres maneras: periódicamente en intervalos de tiempo predefinidos, como resultado tras la ocurrencia de algún evento específico o el alcance de un umbral de alguna variable, o como respuesta a la petición de un usuario.

2.3 La programación OTAP

El concepto de OTAP [3] (Over the Air Programming) o *programación inalámbrica* se refiere a los diversos mecanismos de distribución de nuevo software o actualizaciones. Estos métodos hacen posible que, desde una ubicación o nodo central, se envíen actualizaciones al resto de componentes de una red.

La programación OTA se popularizó mucho con la reprogramación de dispositivos móviles como smartphones: es muy común hoy en día actualizar el firmware de nuestros teléfonos móviles sin necesidad de conectarlo a un ordenador, simplemente haciendo uso de una red Wi-Fi o 3G.

Más recientemente, con los nuevos conceptos de redes inalámbricas de sensores y el Internet de las cosas, donde las redes pueden estar formadas por cientos o miles de nodos, la OTAP ha sufrido otro enfoque. En estos casos se puede aplicar utilizando bandas de frecuencia sin licencia y con bajo consumo y transmisión de datos, usando protocolos como 802.15.4 y Zigbee.

3 CASO DE ESTUDIO

En este trabajo de fin de grado se va a desarrollar uno de los puntos que componen un proyecto real mayor, el proyecto CODISOL (Estimación y Predicción Distribuida de la Radiación para Control de Campos Solares). Éste comprende el estudio y desarrollo de un sistema piloto de detección de nubes y predicción de la radiación solar a corto plazo para el control de centrales solares.

Tal y como se ha explicado anteriormente, la producción de energía renovable y limpia mediante plantas solares aún no ofrece grandes beneficios si se compara con otras fuentes de energía más tradicionales como el carbón y otros recursos fósiles. Una parte de los gastos son generados por el mantenimiento y control de estas plantas, debido a que las condiciones climatológicas varían continuamente. Resulta de vital importancia anticiparse a ellas para lograr una operación eficiente en las centrales, ya que en ellas se llevan a cabo procesos con un tiempo de respuesta relativamente lento y las medidas correctoras tardan un tiempo en hacer efecto.

Las plantas solares ocupan una gran extensión de superficie y están compuestas por cientos de elementos: heliostatos en el caso de la energía solar térmica o seguidores solares en las centrales de generación fotovoltaica. Dichos elementos deben ser controlados individualmente o en grupos para conseguir un buen funcionamiento de la central en tiempo real.

Ejemplo de campo solar

En la mayoría de las plantas solares este tipo de problemas se trata de manera manual, y son los operarios de las centrales los encargados de recalibrar y corregir el funcionamiento de las mismas en base a su experiencia e intuición.

Para cambiar esta metodología, resulta de vital importancia la capacidad de anticipación a la radiación solar disponible en el campo, debido a que es una de las variables más importantes para realizar un buen control de la planta. El control mejoraría de manera sustancial si se dispone de un módulo de control en adelanto y de

predicción, en el que la variable principal sea la radiación solar. De esta forma puede buscarse una manera de automatizar el problema y prescindir de la labor manual de los operarios.

Con el sistema desarrollado en el proyecto podrá predecirse la fracción de campo solar que se verá afectada por la disminución de radiación debido al paso de las nubes por el cielo, anticipándose a las perturbaciones. Con ello se consigue optimizar y lograr una mayor eficiencia en la operación y producción de las centrales solares, manteniendo la producción lo más cerca posible de la generación diaria prevista.

Para hacer posible este proyecto, se pretende desarrollar un sistema completo de detección de cúmulos nubosos y predicción de la radiación solar a corto plazo. Para ello, resultan de especial interés las redes de sensores, la utilización de imágenes de satélite y el procesamiento de imágenes de vídeo para la estimación de la evolución de la cobertura de nubes.

Concretamente, se propone el empleo de cámaras localizadas en varios puntos de las instalaciones para la captura de la evolución de la cobertura de las nubes, junto con sensores complementarios que capten las medidas de radiación, viento, temperatura, etc. Todos los elementos se encontrarán conectados entre sí y utilizarán medidas de consenso para mejorar las predicciones.

Tan importante es la predicción del movimiento de las nubes como la naturaleza de las mismas, ya que diversos tipos de nubes afectan de forma diferente a la radiación recibida. Es por ello que se emplearán técnicas de clasificación estadística basadas en procesamiento visual de nubes.

Los datos que proporcione la red implementada, serán la base de los modelos físicos y algoritmos que se usen para hacer las predicciones. Éstos serán diseñados sabiendo que, aunque las condiciones de radiación solar pueden variar localmente de forma sustancial de unos puntos a otros, existe una clara correlación espacial y temporal de este parámetro. Gracias a esta correlación pueden explotarse los modelos de predicción que se pretenden desarrollar.

A fin de validarlos, los resultados se aplicarán en dos instalaciones solares. Una de ellas ubicada en los laboratorios del departamento de Ingeniería de Sistemas y Automática de la Escuela Técnica Superior de Ingeniería de Sevilla. La segunda batería de pruebas se llevará a cabo en una de las instalaciones de la empresa Abengoa Solar España, en la plataforma SOLUCAR.

Ilustración 5: Instalación solar situada en los laboratorios de la ETSI

Ilustración 6: Plataforma Solúcar

Lo novedoso de este proyecto es la aplicación del gran potencial que ofrecen las redes de sensores a este propósito. La implantación del mismo va a realizarse en dos etapas:

1. En una primera fase se ensayarán estrategias de estimación de la radiación solar, evaluando el comportamiento de la estrategia de estimación distribuída y comparando los resultados con métodos clásicos no basados en redes. Se pretende estudiar la distribución óptima de las cámaras y sensores sobre la instalación, a fin de maximizar el rendimiento y los recursos de energía solar.
2. En una segunda etapa se implantarán las técnicas de control y estimación sobre redes de sensores inalámbricos, sustituyendo el control clásico centralizado por otro completamente distribuído, evaluando su comportamiento y robustez.

A través de la realización de mi parte del proyecto se obtendrán datos necesarios para realizar la primera de las etapas.

4 LA TECNOLOGÍA ZIGBEE

Zigbee [4] [5] es un estándar de comunicaciones inalámbricas diseñado por la ZigBee Alliance [6], de fácil implementación por cualquier tipo de fabricante. Dicha organización, formada en 2002, está compuesta por cientos de miembros pertenecientes tanto al sector de la industria y el desarrollo software como instaladores y fabricantes de equipos, y está abierta a cualquiera que quiera formar parte de ella.

Zigbee se basa en el estándar IEEE 802.15.4 de redes inalábricas de área personal (Wireless Personal Area Network, WPAN) para contruir su capa física y MAC, y define un conjunto de protocolos para la formación de redes de corto alcance que requieren comunicaciones seguras con baja tasa de envío de datos y maximización de la vida útil de las baterías.

Se trata, por tanto, de un sistema idóneo para redes domóticas, especialmente pensado para reemplazar la proliferación de sensores individuales. Además, en este aspecto no tiene una fuerte competencia por parte de otras tecnologías existentes, como WiFi o Bluetooh, como se verá en una de las secciones siguientes.

4.1 El estándar IEEE 802.15.4

El estándar IEEE 802.15.4 fue definido por el comité de estándares IEEE 802 y lanzado en 2003. Define el nivel físico y el control de acceso al medio de redes inalámbricas de área personal, sin llegar a definir los requisitos de las capas más altas. La actual revisión del estándar se aprobó en 2006.

4.1.1 La realación entre los estándares Zigbee e IEEE 802.15.4

Una de las formas más comunes de establecer comunicación de red, ya sea inalámbrica o cableada, es utilizar el concepto de las capas de red. Cada capa es responsable de ciertas funciones en la red y, normalmente, pasan datos y comandos solo a las capas directamente superiores o inferiores a ellas.

Dividir un protocolo de red en capas tiene numerosas ventajas. Por ejemplo, si el protocolo cambia con el tiempo, es fácil reemplazar o modificar únicamente la capa afectada por el cambio, en lugar de reemplazar el protocolo completo. Además, a la hora de desarrollar una aplicación, las capas más bajas del protocolo son independientes de ésta.

Las capas del protocolo Zigbee se muestran en la siguiente figura. Están basadas en el modelo OSI (Open System Interconnect).

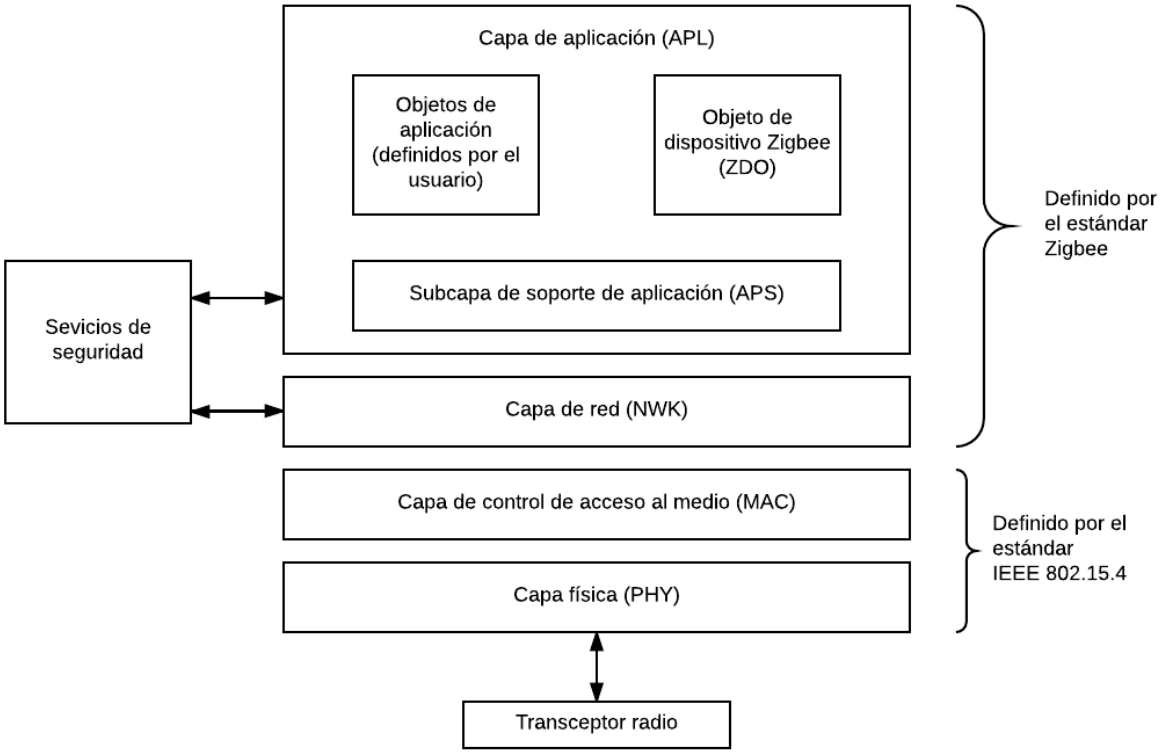

Ilustración 7: Capas del protocolo de comunicación inalámbrica Zigbee

Tal y como se muestra, las dos capas más bajas están definidas por el estándar IEEE 802.15.4. Es la base sobre la cual se define la espeficiación de ZigBee, cuyo propósito es ofrecer una solución completa para las redes WPAN, construyendo los niveles superiores de la pila de protocolos que el estándar no cubre. Por tanto, el estándar Zigbee solo define las capas de red, aplicación y seguridad, y adopta las capas física y MAC del estándar IEEE 802.15.4. Es por ello que cualquier dispositivo compatible con Zigbee se ajusta también al 802.15.4.

El estándar 802.15.4 fue desarrollado de forma independiente a Zigbee, y es posible construir redes inalámbricas de corto alcance solo basándose en él, sin implementar capas de red específicas de Zigbee. En este caso, los usuarios desarrollarían sus propias capas de red y aplicación sobre las capas físicas y MAC de 802.15.4. Este modo de personalizar las capas es normalmente una solución más sencilla que implementar el protocolo Zigbee y se usa solo en aplicaciones muy específicas.

Una ventaja de personalizar capas propias es la baja memoria que se necesita para implementar el protocolo completo, lo que puede resultar en una reducción de costes también. Sin embargo, implementar el protocolo Zigbee completo asegura la interoperabilidad con soluciones de otros proveedores.

4.1.2 Frecuencias de operación y tasas de datos del estándar IEEE 802.15.4

El estándar especifica que se puede trabajar en tres bandas ISM:

- 868 – 868.6MHz (la banda de 868MHz). Esta banda es usada en Europa en multitud de aplicaciones, incluyendo comunicaciones inalámbricas de corto alcance
- 902 – 928MHz (la banda de 915MHz). Se usa mayoritariamente en el norte de América para usos industriales, médicos y científicos.
- 2400 – 2483.5MHz (la banda de 2.4GHz). Tiene los mismos usos que la banda anterior, pero se usa en el resto del mundo.

La siguiente tabla muestra más detalles sobre las bandas citadas:

Frecuencia (MHz)	Número de canales	Tipo de modulación	Tasa de bit (Kbps)	Tasa de símbolo	Método de extensión
868 – 868.6	1	BPSK	20	20	DSSS binario
902 – 928	10	BPSK	40	40	DSSS binario
2400 – 2483.5	16	O-QPSK	250	62.5	Ortogonal 16-array

Tabla 1: Frecuenias de operación y tasas de datos del estándar IEEE 802.15.4

El estándar IEEE 802.15.4 requiere que, si el transceptor utilizado soporta la banda de 868MHz, debe soportar también la de 915MHz, y viceversa. Es por ello que ambas bandas van siempre incluídas juntas como la banda de operación de 868/915MHz. Un transceptor de la banda de 2.4GHz puede soportar las dos bandas anteriores. Sin embargo, no es un requisito del estándar.

La banda de 2.4GHz está aceptada mundialmente y posee la mayor tasa de datos y número de canales. Es por estas razones por lo que la opción más corriente es que los fabricantes desarrollen transceptores que trabajen en esta banda. Sin embargo, el estándar 802.11b trabaja en la misma frecuencia, por lo que la coexistencia de ambos puede ser fuente de problemas en el desarrollo de algunas aplicaciones. Esto, unido al hecho de que las bajas frecuencias penetran mejor en entornos cerrados y con obstáculos, provoca que algunos fabricantes prefieran las bandas de 868/915MHz.

4.2 La arquitectura ZigBee

Para llevar a cabo la configuración de una red Zigbee [7] [8], es necesario conocer los papeles y roles que poseen los dispositivos implicados en ella, así como las topologías que pueden formarse.

4.2.1 Tipos de dispositivo

Existen dos tipos de dispositivos en una red inalámbrica basada en los estándares Zigbee e IEEE 802.15.4: dispositivos con todas las funciones (FFDs – Full-function devices) y dispositivos con funciones reducidas (RFDs – Reduced-function devices). Un FFD es capar de realizar todas tareas descritas en el estándar IEEE 802.15.4 y puede desempeñar cualquier rol en la red. Por otra parte, un RFD tiene capacidades limitadas; puede comunicarse solo con un dispositivo FFD. Los dispositivos con funciones reducidas están pensados para aplicaciones simples, ya que sus tamaños de memoria y energías consumidas suelen ser menores.

4.2.2 Roles de los dispositivos

En una red 802.15.4, un dispositivo FFD, con todas las funcionalidades, puede desempeñar tres roles diferentes: coordinador, coordinador PAN o dispositivo. Un *coordinador* es capaz de retransmitir mensajes. Si además el coordinador es el controlador principal de la red de área personal, estaremos hablando de un *coordinador PAN*. Si el dispositivo no actúa como coordinador, es decir, no puede reenviar mensajes, es llamado simplemente *dispositivo.*

En el estándar Zigbee encontramos ligeras diferencias de terminología. Un *coordinador* Zigbee es equivalente a un coordinador PAN en el IEEE 802.15.4, mientras que el equivalente a un simple coordinador es llamado *router.* Finalmente, un *dispositivo final* hace referencia a aquellos que no son ni router ni coordinador, a un dispositivo.

El siguiente esquema ilustra las diferentes posibilidades anteriormente citadas:

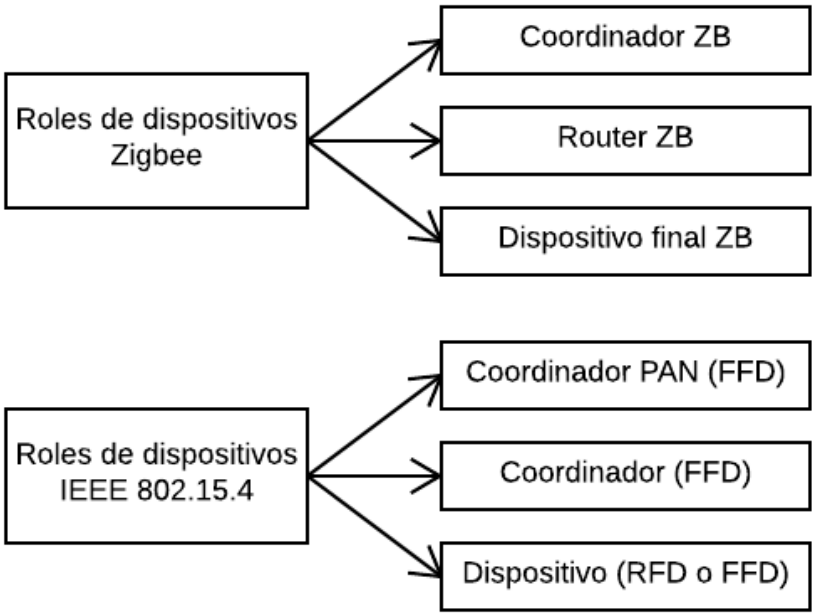

Ilustración 8: Roles de dispostitivos en los estándares IEEE 802.15.4 y Zigbee

4.2.3 Topologías de una red Zigbee

Existen varias posibilidades a la hora de formar una red Zigbee:

Topología en estrella

Cada dispositivo final puede comunicarse solo con el nodo central, que actúan coordinador. El coordinador es el responsable de seleccionar un identificador PAN único que no esté siendo usado por ninguna otra red en su región de influencia, en la que su radio puede comunicarse con otras radios.

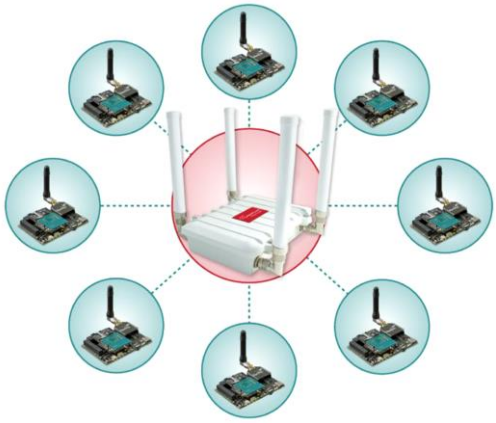

Ilustración 9: Topología Zigbee en estrella

Topología peer-to-peer (punto a punto)

Cada dispositivo final puede comunicarse con cualquier dispositivo que forme parte de la red, siempre y cuando se encuentren lo suficientemente cercanos como para establecer un enlace de conexión entre ellos. Cualquiera de los dispositivos que forme parte de esta topología puede desempeñar el papel de coordinador, por lo que todos ellos disponen de sus funcionalidades al completo. Sin embargo, pueden existir también dispositivos con

funciones reducidas en esta topología, si solo se comunica con otro dispositivo en particular (coordinador o router) de la red.

Este tipo de redes permite establecer distintas formas de red, definiendo restricciones en los dispositivos que la componen sobre el modo en el que pueden comunicarse. Si no hay ningún tipo de restricción, se trata de una **topología en malla**. Podemos encontrar también la **topología en árbol**, en la que el los dispositivos finales, que actúan como hojas, se comunican con routers, que forman las ramas. Cada router reenvía los datos recibidos y los suyos propios al coordinador de la red, encargado del establecimiento inicial de la misma.

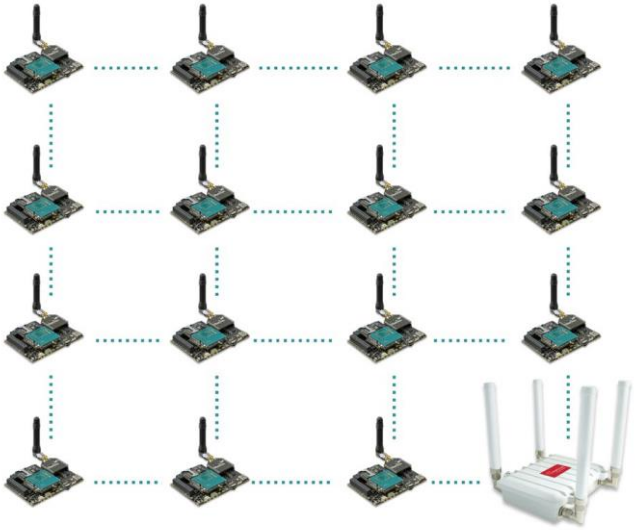

Ilustración 10: Topología Zigbee en malla

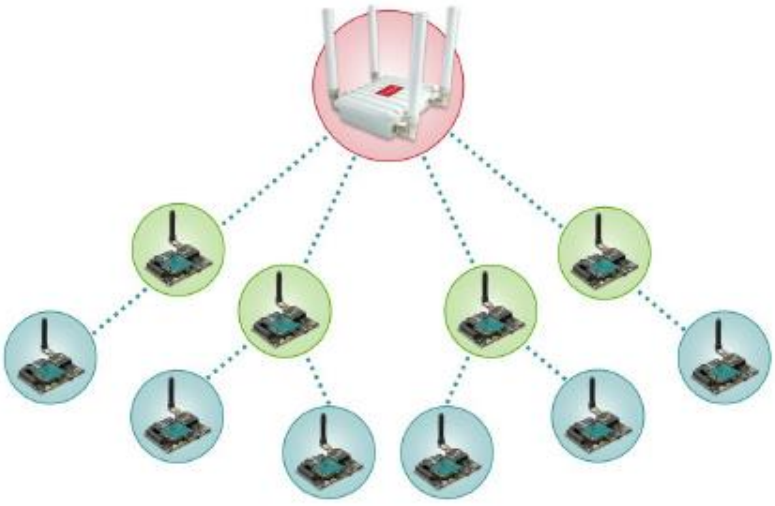

Ilustración 11: Topología Zigbee en árbol

Como puede observarse, los dispositivos sombreados de color azul actúan de dispositivo final, mientras que los sombreados de color verde tienen el rol de router. Los rojos actúan de coordinador de la red.

4.2.4 Las capas del protocolo ZigBee

Como se ha comentado en un apartado anterior, las capas del protocolo Zigbee están basadas en el modelo de

referencia OSI, por lo que existen siete capas. Aunque, como ya se ha indicado, Zigbee solo implementa aquellas que son esenciales para conseguir una comunicación inalámbrica de baja tasa de datos y poca energía consumida: las capas de red y de aplicación. Las capas más inferiores están definidas por el estándar 802.15.4, mientras que la capa de seguridad está definida por ambos estándares.

Una red que implementa todas las capas indicadas anteriormente se considera una red inalámbrica Zigbee.

En la siguiente figura se muestran cada una de ellas:

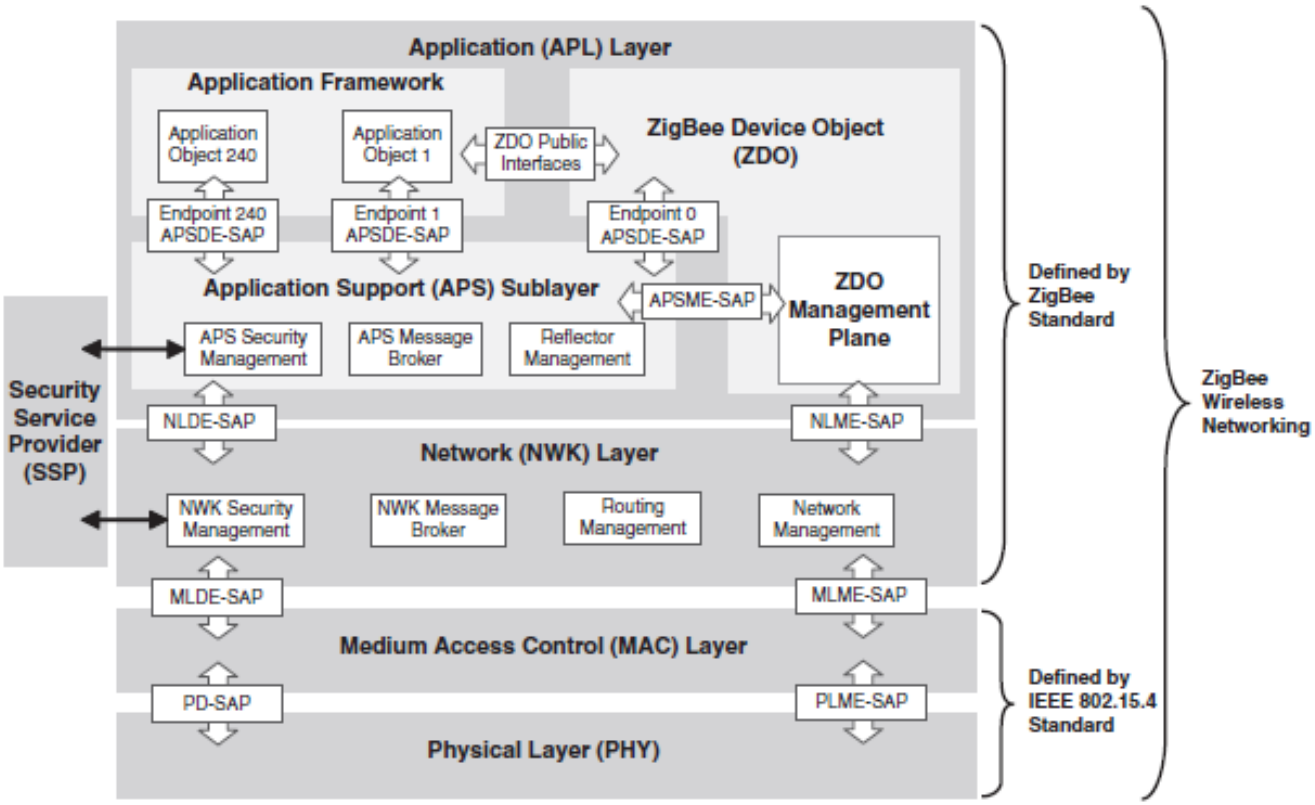

Ilustración 12: Capas del estándar Zigbee

Cada capa se comunica con las capas adyacentes mediante los puntos de acceso al servicio (SAPs). Se trata de un término conceptual mediante el cual una capa de un protocolo puede hacer peticiones de servicio a otra capa.

A continuación se explican más detalladamente cada una de las subcapas y protocolos que conforman la capa de aplicación de Zigbee. El resto de capas son comúnmente conocidas y no resulta de especial relevancia su explicación en este apartado, ya que además, están definidas por el estándar IEEE 802.15.4.

- **ZigBee Device Object (ZDO):** Este protocolo es responsable de toda la administración del dispositivo, además de las seguridad y políticas. Se comporta como una aplicación que reside dentro del nodo Zigbee y tiene su propio perfil al que pueden acceder los dispositivos finales y otros nodos Zigbee. Básicamente representa el tipo de nodo Zigbee del dispositivo.
- **Plano de administración ZDO:** Este plano extiende las capas de red y de soporte de aplicación, permitiendo al ZDO comunicarse con ellas cuando se ejecuten tareas internas del dispositivo. Además, permite lidiar con peticiones de aplicaciones que estén relacionadas con el acceso a la red o la seguridad.
- **Subcapa de soporte de aplicación (APS):** Esta capa es responsable de comunicarse con la aplicación en cuestión, pasando la información necesaria del mensaje a través de un punto de acceso al servicio o SAP.
- **Framework de aplicación:** Contiene los objetos de aplicación y facilita la interacción entre las aplicaciones y la subcapa de soporte a la aplicación a través del punto de acceso al servicio.

- **Plano de seguridad (SSP):** Proporciona servicios de seguridad como el trtamiento de claves seguro o la encriptación de datos. Para ello usa funciones proporcionadas por el hardware del nodo.

4.3 Estudio comparativo de ZigBee con otras tecnologías

El concepto de usar comunicaciones inalámbricas para enviar información y realizar tareas de control dentro de una casa o fábrica no es algo nuevo. Existen muchos estándares para redes inalámbricas de corto alcance. Entre ellos cabría destacar las tecnologías que comparten la banda de frecuencia de 2,4GHz con Zigbee, como el IEEE 802.11 (WLAN – Wireless Local Area Network) o Bluetooth, así como protocolos propietarios tipo MiWi. Cada uno de estos estándares tiene sus ventajas para aplicaciones particulares. El estándar Zigbee está especialmente desarrollado para aplicaciones con necesidades de bajo coste en implementación, baja velocidad de datos y un consumo energético mínimo.

En los apartados siguientes se realiza una comparación de los aspectos más relevantes de Zigbee y otras tecnologías. Esta comparativa nos ayudará a enteder de qué forma se diferencia Zigbee del resto de estándares existentes, y la razón por la cuál se ha escogido este protocolo como forma de establecer la comunicación en el presente proyecto.

4.3.1 Zigbee versus Wifi

El estándar IEEE 802.11 es, en realidad, una familia de estándares. En este caso hablaremos del estándar IEEE 802.11b, ya que opera en la banda de 2.4 GHz, la misma en la que operan Bluetooth y Zigbee.

Este estándar posee una alta tasa de datos (mayor que 11Mbps). Los dispositivos que forman este tipo de redes deben permanecer activos casi de forma permanente y permiten el intercambio de gran cantidad de datos. Es por ello que el consumo energético es mucho más elevado. Su aplicación más común es proporcionar conexión inalámbrica a internet. El rango de cobertura que ofrece en interiores es, normalmente, entre 30 y 100 metros.

Los sistemas que implementan esta tecnología requiren de unos 0.6 watios de potencia. La máxima potencia de señal que se transmite puede ser superior a 20 dBm.

Una de las desventajas a las que se enfrenta actualmente WiFi es la seguridad. Un elevado porcentaje de redes son instaladas por administradores de sistemas y redes por su simplicidad de implementación, sin tener en consideración la seguridad y, por tanto, convirtiendo las redes en redes abiertas y cuya información circula desprotegida por ellas.

4.3.2 Zigbee versus Bluetooth

Bluetooth y Zigbee tienen mucho en común: ambos son protocolos de redes de alcance personal. Sin embargo, Bluetooth tiene una tasa de transmisión mucho mayor (1 Mbps) y su rango de cobertura en interiores alcanza típicamente entre 2 y 10 metros, ya que está más orientado a eliminar la necesidad de cables para comunicarse entre dispositivos electrónicos y accesorios, como puede ser el caso de un teléfono móvil y unos auriculares inalámbricos. Además, permite el intercambio de archivos entre ellos.

Otra diferencia significativa es el consumo. Los dispositivos que soportan la tecnología Bluetooth consumen más energía que los módulos Zigbee, ya que los primeros deben enviar información de forma continuada a la red para mantener el sincronismo.

4.3.3 ¿Por qué Zigbee?

Zigbee es el estándar con menor tasa de datos, por lo que no sería una buena opción para proporcionar conexión inalámbrica a internet o una gran calidad a la hora de conectar unos auriculares. Sin embargo, este estándar

puede proporcionar la mayor potencia y se trata de la mejor solución en cuanto a eficiencia, ya que permite transmitir y recibir información reduciendo costes de implementación a base de simplificar los protocolos de comunicación y reduciendo la tasa de datos.

La principal ventaja que presenta el protocolo Zigbee frente a otros es que reduce el tiempo en el que la comunicación vía radio permanece activa y, por lo tanto, reduce considerablemente el consumo. En el caso de un nodo final, solamente necesita estar activo durante el envío o recepción de datos. No sería el caso de los routers o coordinadores, que deben permanecer continuamente activos a la espera de datos y, por consiguiente, consumirán notablemente más energía que los anteriores.

Por otra parte, debido al amplio abanico de posibilidades que ofrece Zigbee y sus numerosas aplicaciones, existen multitud de proveedores que aportan soluciones basadas en este estándar. Es importante que dichos dispositivos sean capaces de interactuar entre ellos independientemente de su origen de fabricación. En otras palabras, los dispositivos deben ser interoperables. Este aspecto es otra de las grandes ventajas que ofrece Zigbee frente a otras tecnologías. Los dispositivos basados en zigbee pueden interactuar con otros incluso si los mensajes se encuentran cifrados por cuestiones de seguridad.

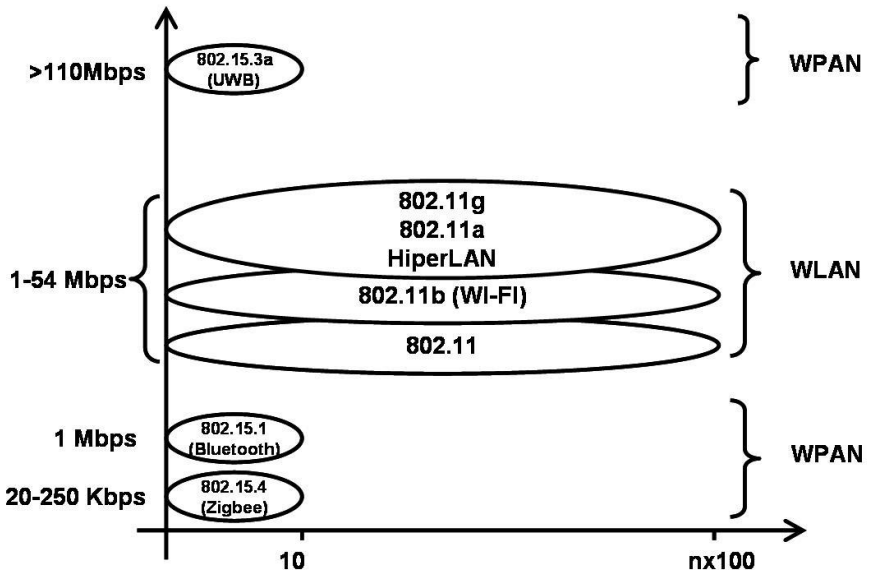

Ilustración 13: Comparativa de anchos de banda de las tecnologías inalámbricas más comunes

5 SELECCIÓN DEL MICROCONTROLADOR

En la actualidad, existen multitud de microcontroladores y dispositivos disponibles en el mercado, de diversas características y pensados para diferentes aplicaciones. A la hora de escoger el dispositivo a emplear en una red de sensores, será necesario tener en cuenta diversos factores como la plataforma en la que se implementan, la tecnología de comunicación que usan, la aplicación donde pretende emplearse, etc.

En este capítulo se estudian algunas de las posibilidades disponibles en cuanto a sistemas operativos, transceptores o plataformas de sensores. Asímismo, se detalla la solución finalmente escogida y los motivos que han propiciado esta decisión.

5.1 Plataformas de nodos sensores

Existen multitud de plataformas y fabricantes relacionados con el internet de las cosas y las redes de sensores. A continuación se habla solo de algunas de las muchas posibilidades disponibles en el mercado.

5.1.1 Arduino

Arduino [9] es una plataforma abierta centrada en el desarrollo de productos electrónicos, orientada a la experimentación por parte de todo tipo de usuarios, desde los más inexpertos a grandes programadores. Sus placas, fáciles de programar y de bajo coste, ofrecen un amplio abanico de posibilidades. Desarrolla varios tipos de placa con distintas características en función de las necesidades del usuario. Cada una de ellas es totalmence configurable y adaptable gracias a la multitud de módulos y elementos que pueden conectarse para ofrecer funcionalidades extra, orientadas tanto a desarrollos para principiantes, el internet de las cosas o wearables. Además, al tratarse de un hardware libre, todos sus diseños están disponibles y pueden reutilizare, e incluso mejorarse.

Las placas de Arduino que se centran en el IoT son Arduino Yún [10] y Arduino Ethernet [11], además de varios módulos que pueden añadirse. La primera de ellas es la que más se adecúa a las necesidades de nuestro proyecto. Se basa en el microcontrolador ATmega32u4 y el chip Atheros AR9331, que soporta una distribución Linux basada en OpenWrt llamada Linino OS. Sin embargo, solo soporta conexión Ethernet y WiFi, por lo que no sería adecuada para las necesidades que se nos plantean. No obstante, es posible conectarle un módulo de comunicación inalámbrica que permita la conexión mediante protocolos XBee.

Ilustración 14: Logotipo de Arduino

Ilustración 15: Placa Arduino Yún

5.1.2 MangOH

La plataforma mangOH [9] desarrolla productos especialmente orientados al internet de las cosas. Es también de código abierto, por lo que cualquiera puede utilizar sus diseños, copiarlos o modificarlos. Sus productos están diseñados con componentes de grado industrial, para que los prototipos desarrollados con ellos puedan ser convertidos facilmente en productos comerciales. Además, son elementos lo suficientemente flexibles para adaptaros a cualquier caso de uso basado en tecnologías inalámbricas y sensores. Su placa mangOH Green nació en 2015 y es utilizada en numerosos proyectos relacionados con el medio ambiente. Posee conectores compatibles con accesorios de Arduino y permite el uso de múltiples tecnologías de comunicación mediante el uso de la placa de extension apropiada.

Ilustración 16: Logotipo de MangOH

Ilustración 17: Placa mangOH Green

5.1.3 Libelium

Libeliun [11] es una empresa española cuyo proyecto está orientado a crear redes inalámbricas de sensores, con requisitos muy específicos y destinados a ser desplegados en un entorno real, al contrario que otras placas, diseñadas para la experimentación.

Libelium diseña y desarrolla un router propio multiprotocolo, el único de su clase que reúne las tecnologías WiFi, ZigBee, GPRS, Bluettoh y GPS en una sola máquina, por lo que ofrece multitude de posibilidades de comunicación.

El entorno de desarrollo que utiliza es igual que el de Arduino, por lo que pueden reutilizarse los códigos desarrollados en esta otra plataforma con solo realizar algunas modificaciones.

Los módulos de comunicación que se utilizan con la plataforma Waspmote pertenecen al fabricante **Digi International Inc** [12], el cual produce varios dispositivos de interés para el diseño de redes de sensores. Se hablará de dos en concreto: el XBee ZB y el XBee-PRO ZB, ya que ambos implementan el protocolo Zigbee. Solo se diferencian en la potencia de transmisión y, por tanto, en el alcance. Poseen un chip Ember EM250, el cual include un microcontrolado dedicado y un transceptor 802.15.4. Debido a su firmware, solo pueden configurarse algunos de sus parámetros, por lo que para aplicaciones más complejas será necesario comunicarse con un microcontrolador de uso general a través de una interfaz serie (como es el caso en el que nos encontramos). Ambos módulos poseen certificación de la Zigbee Alliance, lo que garantiza que puedan comunicarse con cualquier plataforma con la misma certificación.

Ilustración 18: Logotipo de Libellium

5.1.4 TST Sistemas

TST Sistemas [13] es una empresa creada en 2007 por investigadores de la Universidad de Cantabria, cuyos dispositivos permiten el desarrollo rápido de aplicaciones gracias a libre´rias API de TST, la programación sencilla en ANSI C y el uso de un potente micro controlador ARM de 32 bit de bajo consume. Las placas más interesantes desde el punto de vista de crear una red de sensores son el TSgaTe [14] y el TSmoTe [15].

El TSmoTe es un sistema embebido diseñado para permitir desarrollos rápidos y sencillos de aplicaciones inalámbricas de monitorización, control remote y soluciones M2M. Contiene múltiples interfaces de interconexión para conectarle sensores y actuadores.

El concetrador TSgaTe es una plataforma más potente que el TsMote y posee otro tipo de requerimientos. Puede ejercer de pasarela entre dispositivos TSmoTe gracias a su Puerto Ethernet o sus módulos de expansión.

Ilustración 19: TSgaTe

Ilustración 20: TSmoTe

Ilustración 21: Logotipo TST Sistemas

5.2 Solución propuesta

Para cumplir con el objetivo de este proyecto, se han utilizado finalmente las placas de Waspmote, utilizando el módulo de comunicación Zigbee-PRO ZB de Digi. Los motivos que han propiciado esta decisión son los siguientes:

- Se trata de un dispositivo orientado al desarrollo real de redes de sensores, al contrario que otras de las soluciones propuestas, orientadas a la experimentación.
- La empresa desarrolladora, Libelium, es española y tiene su sede en Zaragoza. Ofrece un buen soporte y proporciona sus productos de manera rápida. Además, la marca ya se conocía, por haber trabajado con otros de sus productos con anterioridad.
- La idea de Libelium es vender el resto de componentes en función de las necesidades del cliente y los objetivos del proyecto. Es por ello que el dispositivo cuenta con diversos conectores de expansión que le permiten añadir diferentes elementos en función de las necesidades finales, ofreciendo una gran versatilidad.
- La mayoría de microcontroladores citados anteriormente solo trabaja con una tecnología de comunicación específica. En el caso de Waspmote, podemos utilizar el protocolo de comunicación que más se adecúe a las necesidades y características de nuestra red, con tan solo elegir y conectar el módulo deseado.
- Waspmote cuenta con una gran comunidad, a través de la cual resulta muy sencillo plantear cuestiones o hallar recursos. En su foro, el equipo de Libelium ayuda de forma rápida a solventar cualquier

situación problemática o duda que se plantee.

- La posibilidad de conectar al nodo una placa fotovoltaica resulta muy interesante en un desarrollo real de una red de sensores, ya que gracias a ella, los sensores pueden mantenerse de forma autónoma y conservar sus niveles de batería.

6 ARQUITECTURA HARDWARE

Para desarrollar los objetivos de este proyecto se ha diseñado una red de sensores compuesta por diferentes elementos.

En total se disponen de ocho nodos, cada uno de ellos formados por un sensor Waspmote con su correspondience módulo de comunicación Zigbee, una placa de prototipado Waspmote y un sensor de irradiancia conectado al convertidor analógico digital de la misma. Para dotarlos de autonomía energética, a pesar de que los nodos pueden mantenerse con batería bastante tiempo, se dispone de una placa solar para cada uno de ellos, que les aporta la suficiente energía como para que se vayan recargando mientras están en funcionamiento, a fin de guardar energía para los días nublados.

Además, los nodos se comunican con una pasarela Meshlium, en cuya base de datos se almacenarán los resultados obtenidos para ser procesados y empleados por otras partes del proyecto CODISOL. El conjunto de los nodos con la pasarela forma una red cuya topología se ha ido modificando a lo largo de la realización del proyecto, tras numerosas pruebas.

En los apartados que siguen, se muestran algunas características de los elementos citados y la topología final escogida.

6.1 El microcontrolador Wasmote v1.2 de Libelium

Waspmote [13] es un sensor especialmente orientado a desarrolladores. Trabaja con diferentes protocolos y frecuencias y, además, en el mercado hay muchos sensores ya preparados para conectarse a este nodo.

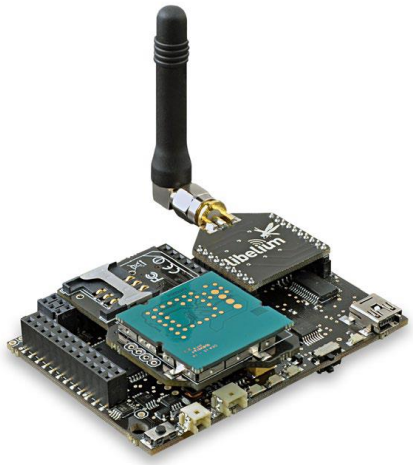

Ilustración 22: Microcontrolador Waspmote

6.1.1 Especificaciones generales de Waspmote

En la siguiente tabla se recogen los datos más relevantes sobre las especificaciones de Waspmote, indicadas en el manual de Libelium.

Microcontrolador	ATmega1281
Frecuencia de operación	14.7456 MHz
Memoria SRAM	8KB
Memoria EEPROM	4KB
Memoria FLASH	128KB
Admite tarjeta SD	2GB
Peso	20 gramos
Dimensiones	73.5 x 51 x 13mm
Rango de temperaturas	-10ºC, + 65ºC
Reloj	RTC (32KHz)

Tabla 2: Especificaciones generales de Waspmote

6.1.2 Consumo energético de Waspmote

La siguiente tabla muestra el consumo teórico del dispositivo Waspmote, indicado en su manual técnico.

Encendido	15mA
Modo sleep	55µA
Modo deep sleep	55µA
Modo hibernado	0.07µA

Tabla 3: Datos de consumo del dispositivo Wasmote

6.1.3 Entradas y salidas de Waspmote

7 pines analógicos de entrada
8 pines digitales de entrada/salida
1 PWM
2 UART (transmisor-receptor asíncrono universal)
2 buses serie de datos I2C
1 puerto USB
1 bus de interfaz serie SPI

Tabla 4: Entradas y salidas del dispositivo Waspmote

En las siguientes imágenes se detallan los elementos que componen la placa del sensor Waspmote:

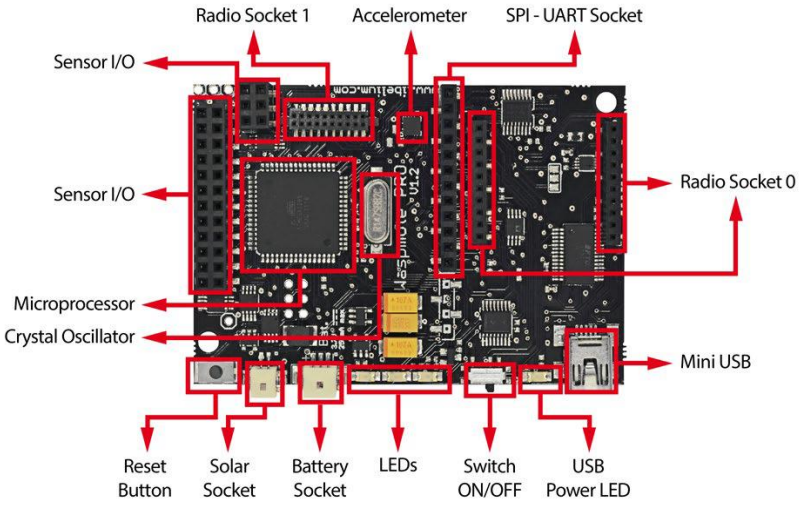

Ilustración 23: Parte superior de la placa Waspmote

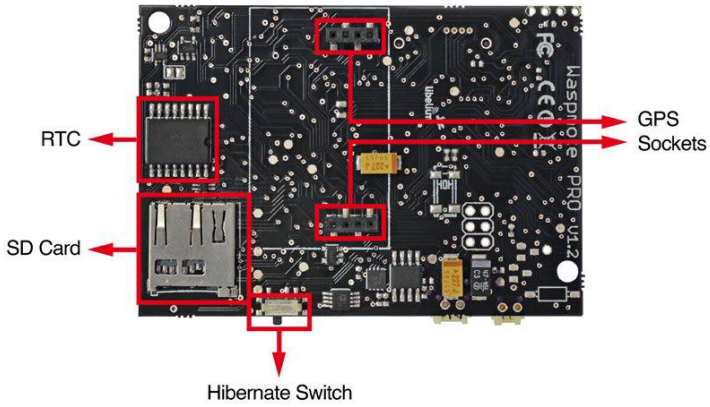

Ilustración 24: Parte inferior de la placa Waspmote

6.1.4 Datos eléctricos de Waspmote

Voltaje de la batería	3.3V – 4.2V
Carga USB	5V – 280 mA
Carga mediante panel solar	6 – 12V – 280mA

Tabla 5: Datos eléctricos de Waspmote

6.1.5 Sensores en la placa Waspmote

Los sensores que se encuentran incorporados en Waspmote se muestran en la tabla que sigue. Al tratarse de un dispositivo modular, es posiblem como ya se ha comentado, conectar otros elementos y sensores al mismo.

Sensor de temperatura	-40ºC, + 85ºC (exactitud de ±0.25ºC)
Acelerómetro en los 3 ejes	±2g/±4g/±8g
Nivel de batería	

Tabla 6: Sensores incorporados en el dispositivo Waspmote

6.2 El módulo Zigbee

El módulo Zigbee empleado en el desarrollo de este proyecto es el XBee-ZB-PRO [13] desarrollado por Digi International Inc [12]. Este dispositivo cumple con el estándar ZigBee Pro v2007. Además de cumplir las especificaciones necesarias para ello, añade ciertas funcionalidades, como son la detección de mensajes duplicados o el descubrimiento de nodos.

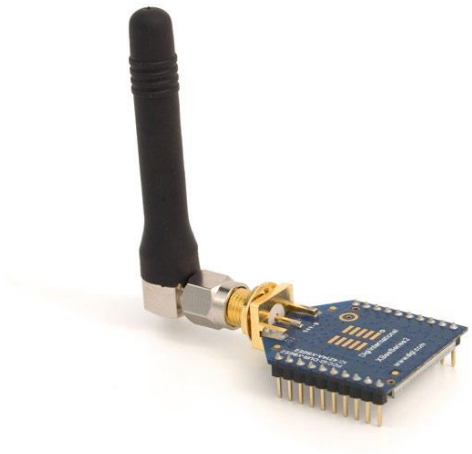

Ilustración 25: Módulo XBee-ZB-PRO de Waspmote

En la siguiente tabla se muestran algunas de sus características:

Antena	2.4GHz, 2dBi/5dBi (se usa la de 5dBi) 868/900MHz: 0dBi/4.5dBi
Conector RPSMA	
Comunicación encriptada	Protocolo AES 128b
Señal de control	RSSI
Topologías	Estrella, árbol, malla

Tabla 7: Características del módulo XBee-PRO-ZB

En la siguiente tabla comparativa, pueden observarse las diferencias con otros módulos XBee de Digi.

Modelo	Protocolo	Frecuencia	Potencia de transmisión	Sensibilidad	Rango de distancia
XBee-802.15.4-Pro	802.15.4	2.4GHz	100mW	-100dBm	7000m
XBee-ZB-Pro	ZigBee-Pro	2.4GHz	50mW	-102dBm	7000m
XBee-868	RF	868MHz	315mW	-112dBm	12km
XBee-900	RF	900MHz	50mW	-100dBm	10km

Tabla 8: Comparativa de módulos XBee de Digi

6.3 Placa de prototipado 2.0 de Waspmote

Para obtener unas medidas más exactas de las mediciones, se ha optado por usar un sensor de irradiancia más preciso que el que desarrolla Libelium. Es por ello que se emplea la placa de prototipado de Waspmote [14], diseñada para permitir la integración del microcontrolador con cualquier tipo de sensor.

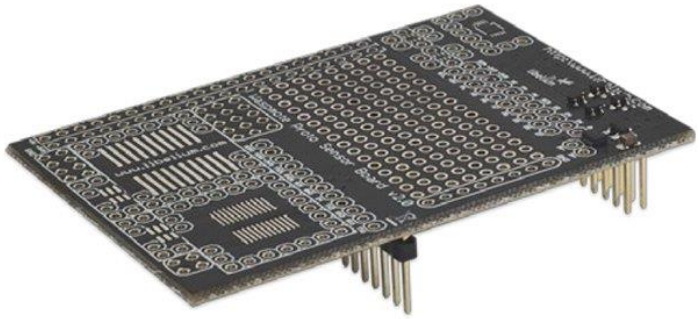

Ilustración 26: Placa de prototipado del sensor Waspmote

En los siguientes apartados se detallan algunas de sus características y componentes.

6.3.1 Especificaciones de la placa de prototipado

Peso	20 gramos
Dimensiones	73.5 x 51x 1.3mm
Rangos de temperatura	-20ºC, 65ºC

Tabla 9: Especificaciones de la placa de prototipado

6.3.2 Características eléctricas de la placa de prototipado

Características operacionales	
Voltaje de alimentación de la placa	3.3V y 5.5V
Voltaje de alimentación del convertidor ADC	5V
Máxima corriente admitida (continua)	-0.3 – 3.8V
Máxima corriente admitida (de pico)	400mA
Máximos absolutos	
Voltaje de los pines	-0.5V – 3.8V
Voltaje de entrada del convertidor analógico digital	-0.3V – 5.3V
Corriente de los pines	40mA

Tabla 10: Características eléctricas de la placa de prototipado

6.3.3 Área de pines

La placa dispone de una matriz de 16x8 pines que permite conectar diferentes elementos.

También dispone de una serie de pines que proporcionan salida a todas las señales del microprocesador. La correspondencia de los mismos es la que se indica en la siguiente tabla:

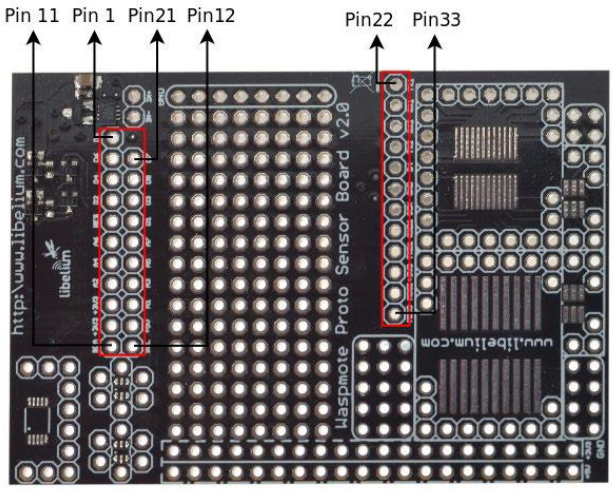

Ilustración 27: Pines de entrada y salida de la placa de prototipado

Finalmente, en la placa pueden encontrarse una serie de pines conectados a alimentación de 3.3V, de 5V y conexiones a tierra.

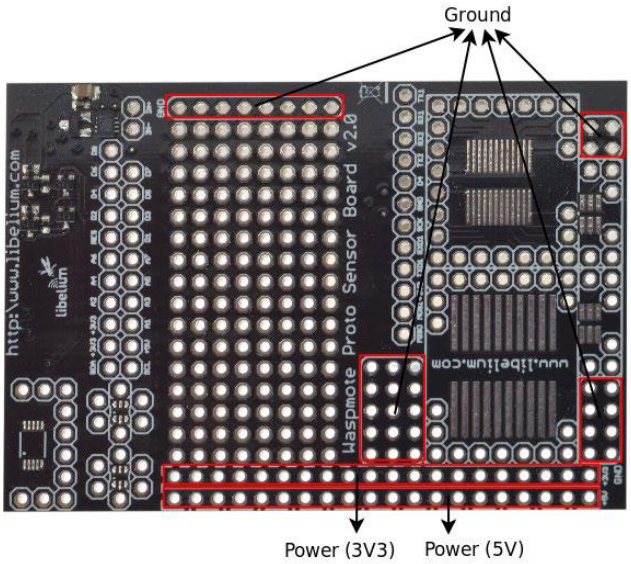

Ilustración 28:Pines de alimentación de la placa de prototipado

6.3.4 Área de circuitos integrados

La placa dispone de 7 circuitos SMD integrados, de diferentes tamaños: uno de 20 puertos de tipo SO, una de 20 puertos TSSOP, uno de 10 puertos micro-SOIC, dos de 6 puertos SOT-23 y dos de 60 puertos SC-70.

En las siguientes figuras se muestran cada uno de ellos.

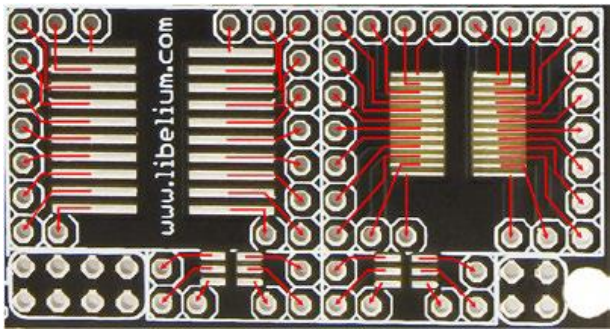

Ilustración 29:Área de ciuritos SO, TSSOP y SOT-23 de la placa de prototipado

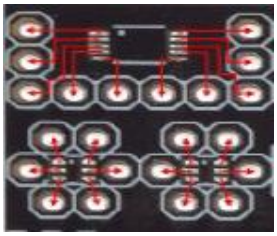

Ilustración 30:Área de circuitos micro-SOIC y SC-70 de la placa de prototipado

6.3.5 El convertidor analógico digital de la placa de prototipado

El microprocesador de Waspmote incorpora un convertidor analógico digital de 10 bits. En la placa de prototipado se dispone de un convertidor de 16 bits sigma-delta, que proporciona una mayor resolución. Su tiempo máximo de conversion es de 23 ms.

En este proyecto se hará uso del convertidor analógico digital de la placa para conectar al Waspmote un sensor de irradiancia de mayor precisión.

Las comunicaciones con este dispositivo se realizan a través de un bus I2C que permite una lectura diferencial mediante dos pines de entrada. Dichos pines permiten una medida diferencial entre -4.5V y 4.5V

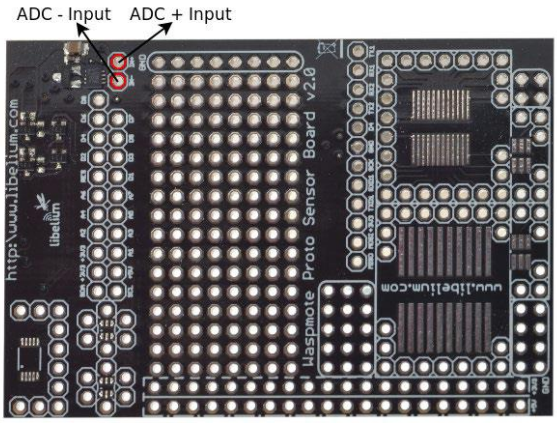

Ilustración 31: Pines de entrada para el convertidor analógico digital de la placa de prototipado

El consumo del convertidor analogico digital, según las especificaciones, es el que se muestra en la siguiente tabla:

Mínimo (constante)	0µA
Convertidor analógico digital (en reposo)	200µA
Lectura del convertidor (20ms aprox.)	900µA

Tabla 11: Datos de consumo del convertidor analógico digital de la placa de prototipado

6.4 Meshlium

Meshlium [15] es un router cuya función es hacer de puerta de enlace de la red diseñada con Waspmote. A través de esta pasarela, con la configuración adecuada, los datos recogidos serán remitidos al punto final de almacenamiento y tratamiento de la información.

Este dispositivo puede ser configurado hasta con seis tipos de radio diferentes a la vez: Wifi 2,4Gh, Wifi 5Gh, 3G/GPRS, Bluetooh y XBee y LoRa.

Además, incluye la posibilidad de montar un modulo GPS para aplicaciones móviles, así como ser alimentado por baterías o paneles solares.

Para la realización de este proyecto, se ha escogido el modelo Meshlium ZigBee-PRO-AP, el cual puede tomar los datos provenientes de la red de sensores, formada por los dispositivos Waspmote, y enviarlos a internet

usando una interfaz 3G/GPRS. A su vez, los sensores podrían ser capaces de mandar datos a la pasarela a través de GPRS, 3G, Wifi o a través de un punto de acceso vía HTTP (esta funcionalidad no se usa en el proyecto, y solo se utiliza el protocolo Zigbee). Además, los usuarios pueden conectarse directamente a Meshlium a través de su interfaz de control y consultar los datos almacenados.

Ilustración 32: Dispositivo Meshlium

Algunas de sus características principales son las que se indican en la siguiente tabla:

Procesador	500MHz (x86)	
Memoria RAM	356MB (DDR)	
Disco duro	8GB *El sistema operativo y el sistema de administración ocupan aproximadamente 2,5GB, por lo que el espacio libre disponible para almacenar datos es 5,5GB	
Potencia	5W (18V)	
Fuente de alimentación	POE (Power Over Ethernet)	
Consumo normal de corriente	270mA	
Consumo alto de corriente	450mA	
Máximo suministro de corriente	1,5A	
Cubierta	Material	Aluminio

	Dimensiones	210 x 175 x 50mm
	Peso	1,2Kg
	Protección externa	IP65
Tiempo de respuesta a ping Ethernet	60s	
Tiempo para tener todos los servicios funcionando	90s	
Tipos de fuente de alimentación para el POE	Corriente AC 220V Batería o panel solar (DC-12V) Cargador de coche (DC-12V)	
Sistema	Linux Debian. Protocolo de comunicación OLRS Mesh Drivers MadWiFi	
Software de administración	Meshlium Manager System (open source)	
Seguridad	Autenticación WEP, WPA-PSK, HTTPS y acceso SSH	

Tabla 12: Características de Meshlium

Al disponer de un Sistema operativo completo como Linux, Meshlium puede ofrecer diferentes servicios, entornos de programación y sistemas de almacenamiento. Los servicios activados que podemos encontrar en la pasarela son HTTP/HTTPS y SSH. Aunque los entornos de programación instalados por defecto son las librerías de C y PHP, también pueden instalarse otros como C++, Java, Python, Perl o Ruby. En cuanto a sistemas de almacenamiento, Meshlium cuenta con dos sistemas de base de datos diferentes: MySQL instalado por defecto, y la posibilidad de instalar Postgre.

Estas características hacen que Meshlium sea una opción interesante, que permite integrar en un solo dispositivo la coordinación de los distintos nodos Waspmote, el almacenaje de las lecturas realizadas y el envío de esta información mediante Wifi a otro nodo para su posterior tratamiento.

Al disponer de una base de datos local dentro del propio dispositivo, disponemos de la opción de almacenar los datos directamente en él, o bien sincronizarlos con otra base de datos externa mediante Wifi o conexión a internet. En este último caso, la base de datos interna proporcionaría una redundancia útil que evitaría la pérdida de datos en caso de una pérdida de conexión entre la pasarela y el punto final de destino de los datos.

El siguiente esquema muestra las posibilidades que ofrece Meshlium en cuanto a almacenamiento, ilustrando esta filosofía.

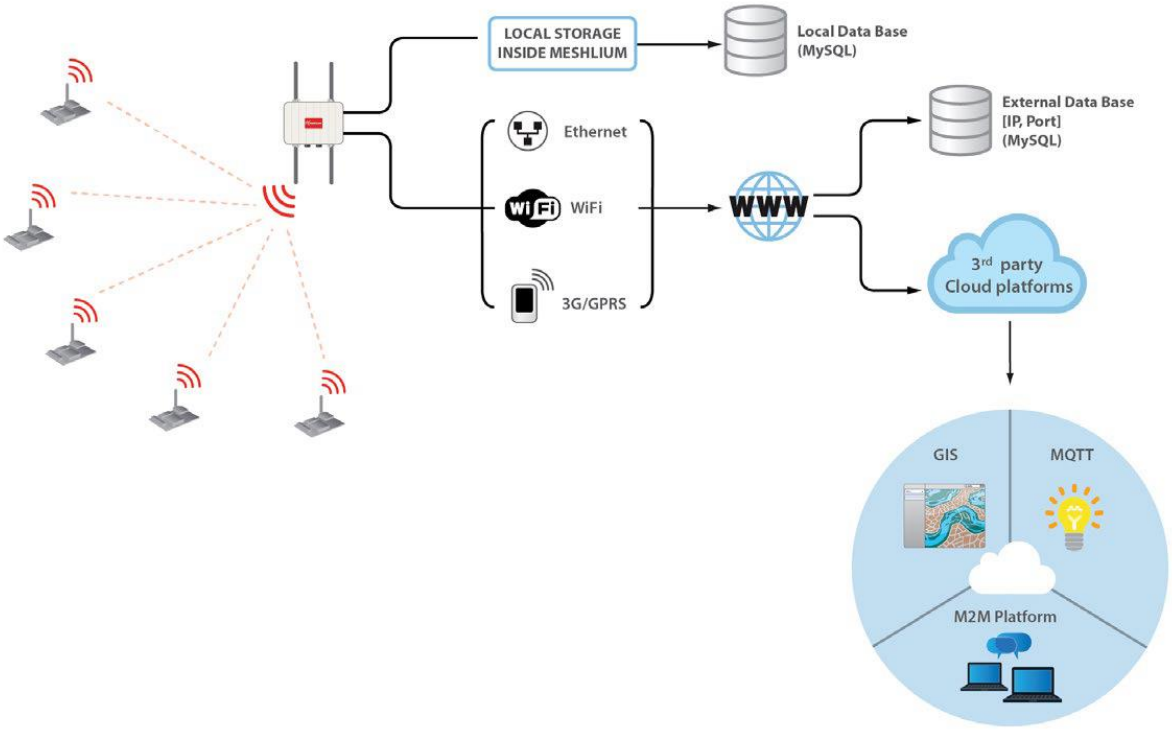

Ilustración 33: Opciones de almacenamiento de Meshlium

Como puede observarse, también pueden sincronizarse los datos almacenados con diversas plataformas en la nube que colaboran con Libelium.

6.5 Waspmote Gateway

El Waspmote Gateway [13] es un dispositivo que permite recolectar los datos que circulan por la red de sensores, conectándose a un ordenador a través de USB o a cualquier otro dispositivo con entrada USB estándar. Actúa como un puente de datos o punto de acceso entre la red de sensores y el equipo receptor, y es capaz de decidir qué datos almacenar en función de las necesidades. Su uso está orientado a realizar pruebas de laboratorio, y para desarrollos reales se debe utilizar la pasarela Meshlium. Sin embargo, como se verá mas adelante, Meshlium no permite realizar programación remota cuando se utilizan módulos de comunicación Zigbee. Para realizar este tipo de tareas se ha tenido que utilizar este dispositivo.

Además, contiene botones programables y LEDs que indicant si la placa está encendida, si un mensaje es transmitido o recibido, y otro configurable.

Ilustración 34: Waspmote Gateway

6.6 Sensor de irradiancia

Para medir la irradiancia global horizontal se ha elegido el piranómetro fotodiodo modelo SP1110 de Campbell Scientific [16]. Se trata de un sensor de irradiancia altamente compacto y estable, de alto rendimiento y absado en una fotocélula de silicio. Está calibrado usando referencias en condiciones de luz natural. Algunas de sus características son las siguientes:

- Ofrece una sensibilidad a la luz entre 350nm y 1100nm.
- Su detector de fotocélula de silicio ofrece características de baja fatiga.
- Su temperatura de operación va desde los -35ºC hasta los +75ºC
- Ofrece una gran precisión de ±5%, aunque normalmente es menor que ±3%

Ilustración 35: Sensor de irradiancia SP1110

6.7 Placa fotovoltaica

Para dotar de alimentación adicional a los nodos, a fin de que la batería se vaya recargando con la luz solar, se ha equipado a cada nodo con una placa fotovoltaica de 7,2V que les aporta la energía suficiente para que no se apaguen, aunque haya largos periodos de tiempo con el cielo nublado.

Ilustración 36: Placa fotovoiltaica de 7,2V

6.8 Montaje de elementos

En la siguiente imagen puede apreciarse el montaje de un nodo, compuesto por el dispositivo Waspmote, la placa de prototipado, el módulo de comunicación Zigbee y el sensor de irradiancia. Todo ello se ha montado dentro de una caja que protege a los dispositivos de las condiciones meteorológicas. Además, para dotar al Waspmote de una fuente de alimentación, a fin de alargar la vida del dispositivo, se ha conectado a cada nodo una pequeña placa solar que aporta la energía suficiente como para que la batería se vaya recargando, sin llegar a descargarse por completo en los periodos en los que no hay luz solar.

Ilustración 37: Montaje de elementos de un nodo

7 SOFTWARE E IMPLEMENTACIÓN

Para que toda la red funcionara de manera correcta, ha sido necesario realizar una configuración adecuada a cada elemento de la misma. En este capítulo se explica el detalle de funcionamiento de cada uno, el procedimiento que se ha seguido para realizar ciertas tareas y los programas implementados en cada uno de los nodos, así como la formación de la red de sensores del proyecto y su topología final.

7.1 El IDE de Waspmote

El IDE o *Integrated Development Environment* es un entorno de programación compuesto por diferentes herramientas que ayudan a desarrollar aplicaciones. Se compone usualmente de un editor de código, un compilador, un depurador y una interfaz gráfica.

El software desarrollado para cargar en plataformas Waspmote, debe ser compilado y cargado a través del IDE de Waspmote [17],basada en la IDE de Arduino. El desarrollo de la misma no habría sido posible sin el carácter *open source* de Arduino. El entorno de desarrollo de Waspmote incorpora barra de herramientas, editor de código, terminal de mensajes del compilador y monitor serie, entre otras funcionalidades.

Cualquier otro entorno distinto al de Waspmote podría volver inestable al dispositivo, por lo que será éste el que se utilice en el proceso de desarrollo y carga de programas. Libelium proporciona versiones del mismo para los sistemas operativos más comunes, así como las APIs para lectura de sensores y utilización de los módulos de comunicación de las diferentes placas. Concretamente, se ha utilizado en su versión 04, instalada en un sistema Linux de 64 bits.

La programación de los Waspmote es idéntica a la de las placas Arduino, por lo que hereda el lenguaje y todas las características asociadas. El software escrito para Arduino se denomina *sketch,* está basado en C y C++ y librerías estándar de este lenguaje. Los *sketches* son convertidos a lenguje máquina a través de un compilador tipo *gcc* propio de los microprocesadores Atmel que incorporan las placas. Este código ya convertido, combinado con las librerías básicas de Arduino, contiene el fichero binario que será cargado en la memoria de programa del chip de la placa.

La estructura principal de un *sketch* en Arduino y, por tanto, en Waspmote, se compone de dos funciones principales: *setup()* y *loop(). Setup()* es la función llamada cuando comienza la ejecución del *sketch* y contiene todas las inicializaciones de variables, inicio de módulos de la placa mediante librería y seteo de pines como entrada/salida o encendido/apagado. Esta función solo se ejecuta una vez, después de cada reseteo o encendido de la placa.

Loop() es, como su propio nombre indica, el bucle que se mantiene en ejecución y provee un control active de la placa, provocando que ésta responda ante cambios externos según se programe.

Las estructuras de control, operadores lógicos, sintaxis, formatos de variables y funciones básicas son idénticas a las de C/C++.

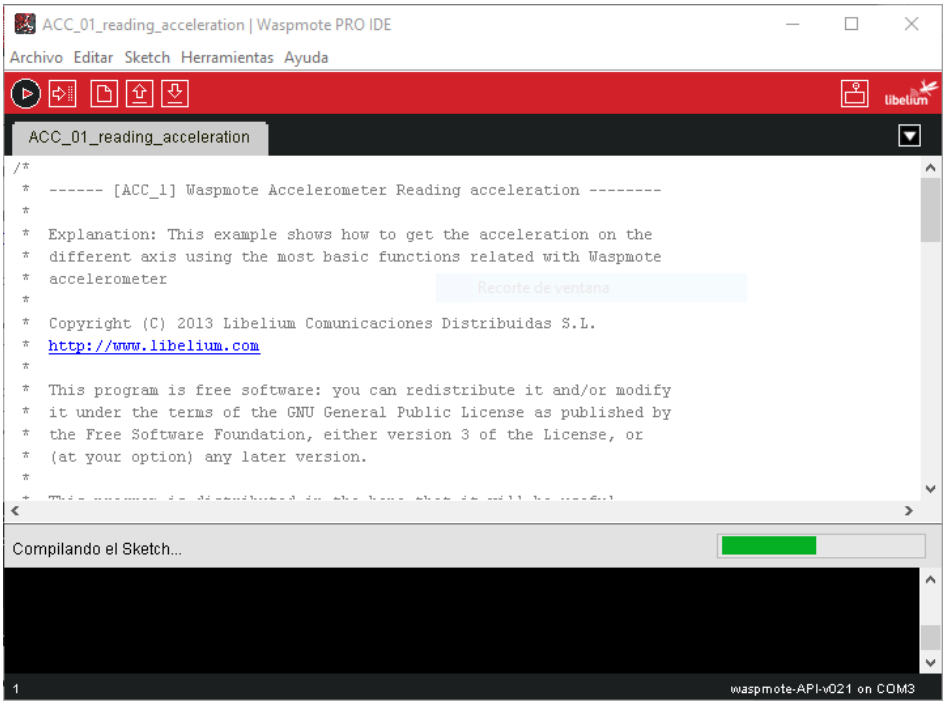

Ilustración 38: Pantalla principal del IDE de Waspmote

7.2 Funcionamiento y configuración de la pasarela Meshlium

La pasarela Meshlium puede controlarse y configurarse mediante su interfaz web, o bien accediendo a ella vía SSH (esta opción permite una configuración mucho más avanzada). En esta sección se detallan las posibilidades que ofrece la interfaz web, y la configuración que se le ha aplicado a la pasarela, teniendo en cuenta que solo se han explotado algunas de las muchas posibilidades que ofrece, de forma que no se explicarán todas sus características en este documento.

7.2.1 Acceso a Meshlium mediante HTTP/HTTPS

Una vez puesta en funcionamiento la pasarela, ésta crea una red Wifi a la que podemos conectarnos. Dicha red, como veremos más adelante, puede protegerse. Tras esto, podemos acceder a la interfaz de web de Meshlium introduciendo en un navegador su dirección IP.

A partir de este punto podremos acceder a la interfaz web de Meshlium, introduciendo en un navegador su dirección IP y los credenciales correctos.

En la imagen que sigue se muestra la pantalla que aparece una vez completado el acceso. En ella pueden apreciarse los distintos menús de la interfaz: configuración de interfaces, redes de sensores, configuración de conexiones a plataformas en la nube, herramientas, ajustes del sistema, administración de actualizaciones y una sección de ayuda a través de la cual puede accederse a la documentación de Libelium y otro tipo de recursos.

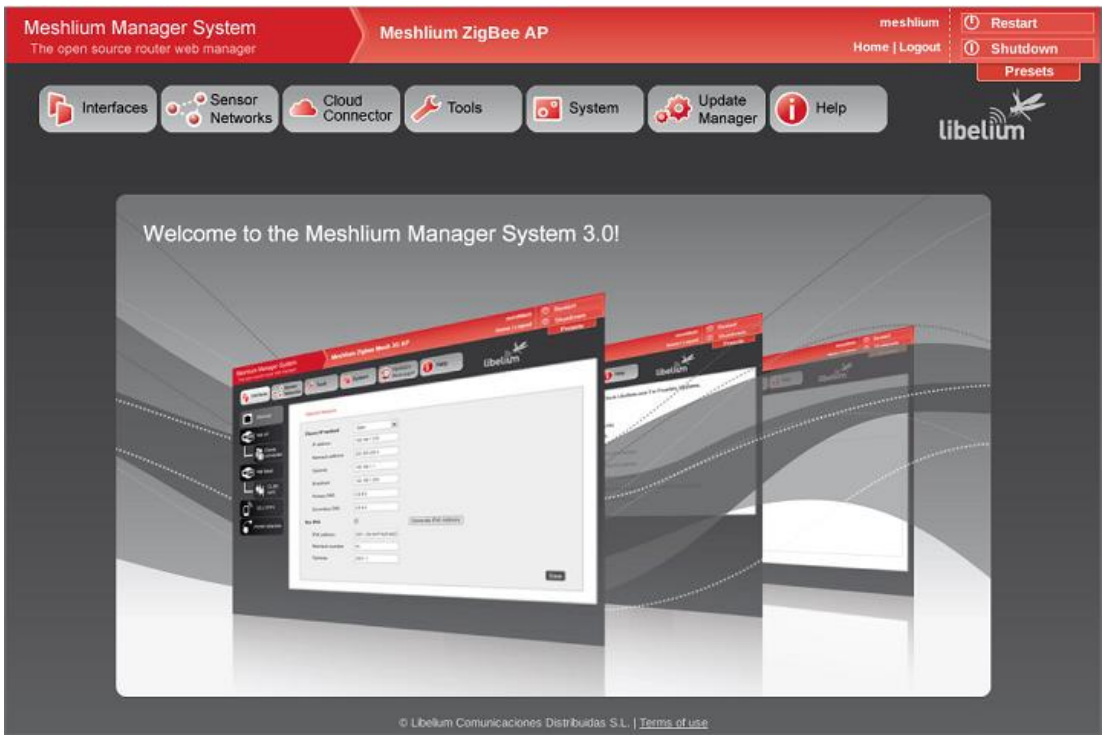

Ilustración 39: Interfaz web de gestión de Meshlium

7.2.2 Configuración de interfaces

En nuestro caso, la pasarela se ha conectado a una boca que le otorga una dirección IP fija del rango de direcciones ofrecida por el centro de cálculo de la escuela.

En la siguiente imagen se muestra la configuración Ethernet citada:

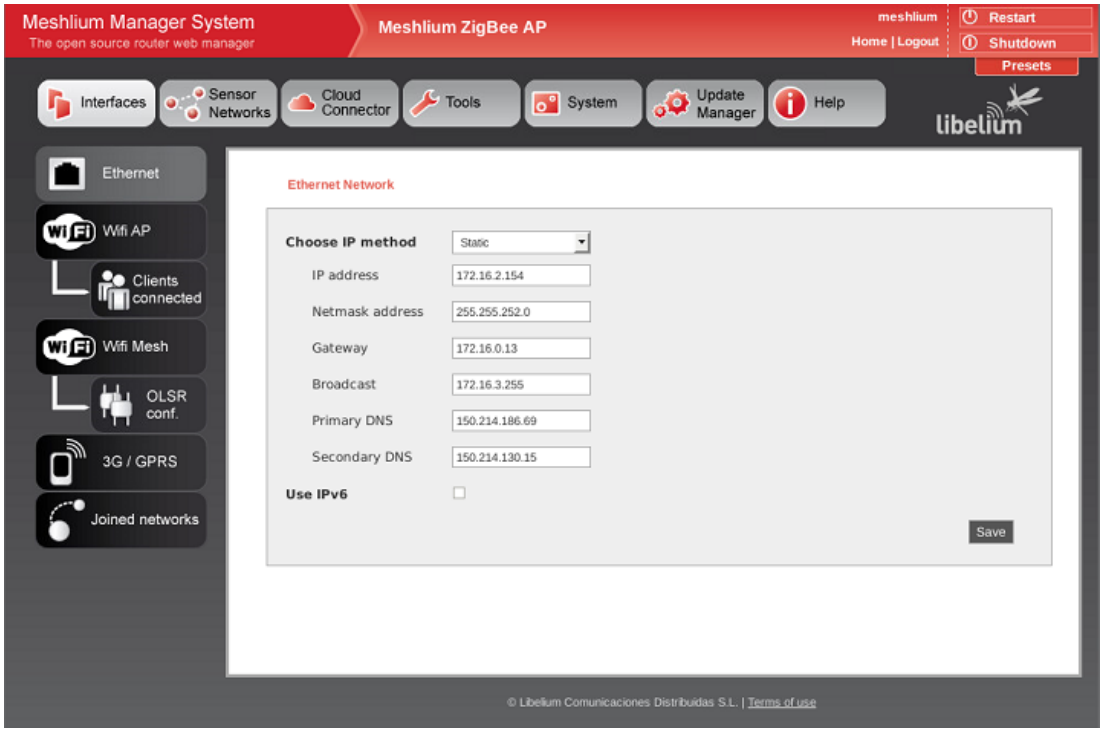

Ilustración 40: Configuración de interfaces de Meshlium

Con un ordenador instalado en el departamento de telemática, que pueda conectarse a dicha IP, es posible acceder a Meshlium de forma remota (desde un lugar distinto a la escuela), facilitando así muchas de las tareas del presente proyecto. Para esta finalidad se ha usado la aplicación TeamViewer.

Por otro lado, también es possible modificar la configuración del punto de acceso Wifi. En este caso, la única modificación realizada ha sido asignarle seguridad WPA a la red, tal y como puede verse a continuación:

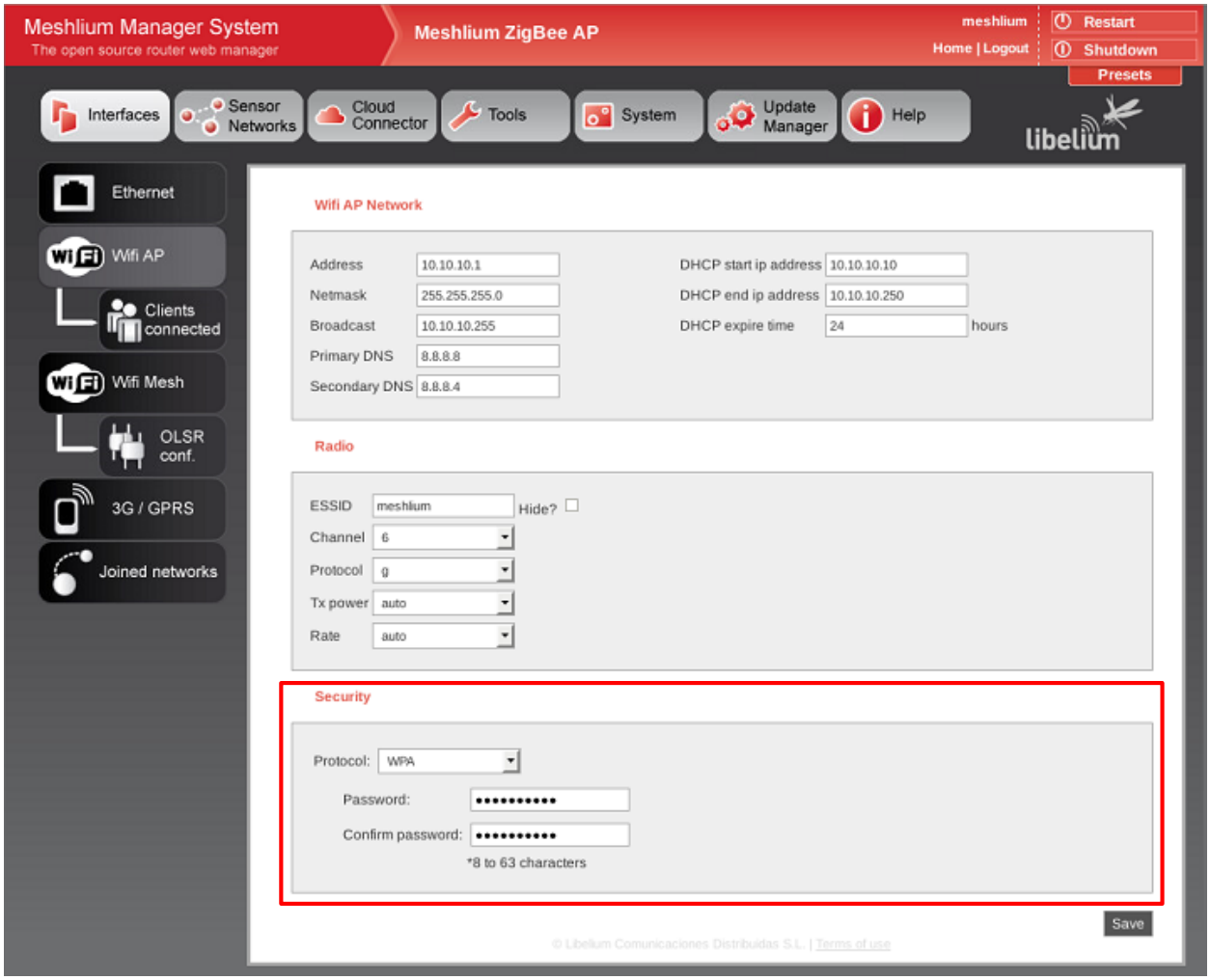

Ilustración 41: Configuración de seguridad de Meshlium

También podrían configurarse las opciones de red y de la interfaz radio, pero no ha sido necesario para la realización del proyecto.

7.2.3 Redes de sensores

En este menú podemos acceder a la información sobre las redes de sensores de las que forma parte Meshlium. Como se ha comentado, la pasarela puede trabajar con diferentes protocolos de comunicación, pero en nuestro caso solo se encuentra implementada la interfaz Zigbee. A través de esta sección pueden configurarse algunos de sus parámetros, como el nombre de la pasarela o un modo de encriptación, que en nuestro caso se ha preferido evitar, ya que los datos que circulan por nuestra red de sensores no son especialmente sensibles. Además, puede

verse la dirección MAC Zigbee de la pasarela, necesaria para configurar el resto de nodos y que la comunicación funcione.

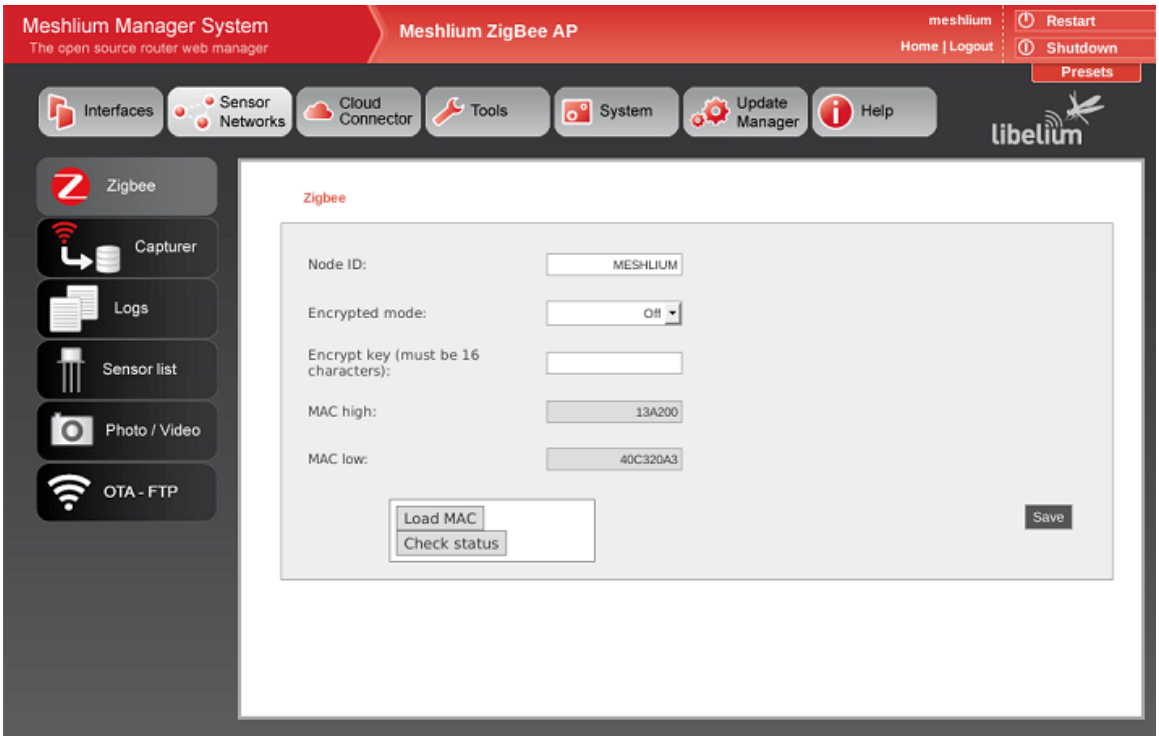

Ilustración 42: Parámetros de configuración Zigbee de Meshlium

En la sección “Capturer” puede visualizarse el tráfico de la red, así como la configuración de las bases de datos interna y externa. Los parámetros de la base de datos tampoco han sido modificados para este proyecto.

La pasarela es capaz de almacenar los datos que recibe gracias al *SensorParser*. Se trata de un software integrado en Meshlium capaz de recibir las tramas Zigbee, procesarlas y almacenarlas en la base de datos, así como sincronizar la base de datos interna con una externa, en caso de ser necesario. Este software puede desactivarse, y será necesario hacerlo para realizar tareas como la programación remota, tal y como se muestra en apartados posteriores. Durante el tiempo que permanezca inactivo, no se almacenarán en la base de datos las tramas que reciba la pasarela.

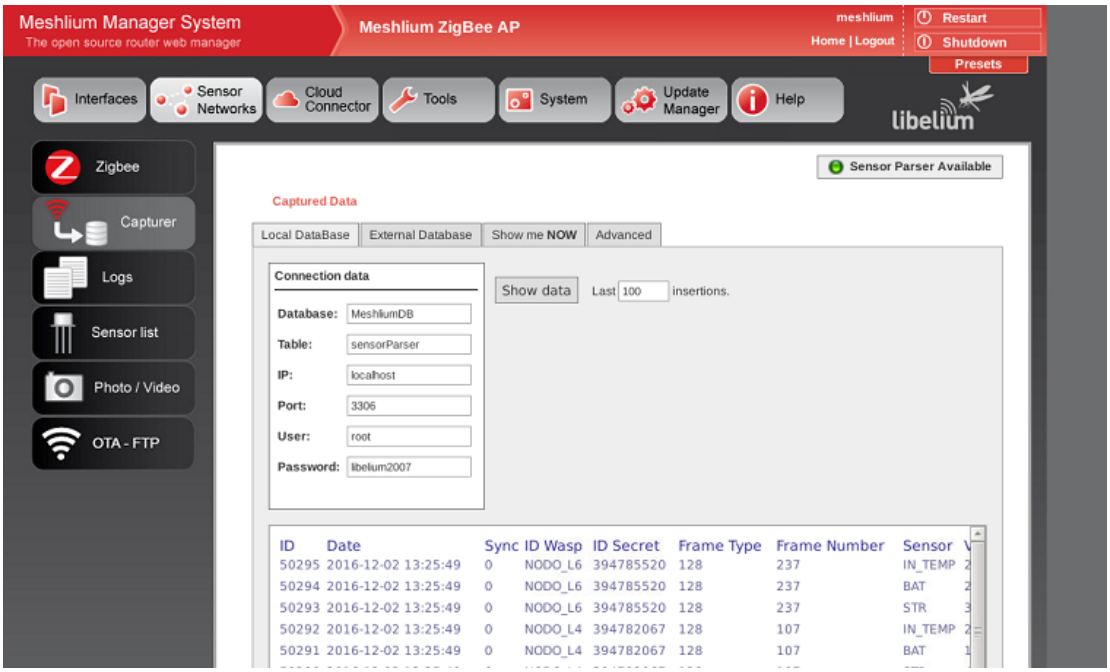

Ilustración 43: Vista del capturador de datos de Meshlium

Por último, cabe destacar en esta sección una funcionalidad destinada a la programación OTA (que se verá en apartados posteriores). A través de ella, es posible subir ficheros a la pasarela con la finalidad de usarlos posteriormente para actualizar los nodos remotamente. Esto podría hacerse también copiando los ficheros deseados mediante SSH, pero la interfaz ofrece un método más seguro para desarrolladores inexpertos.

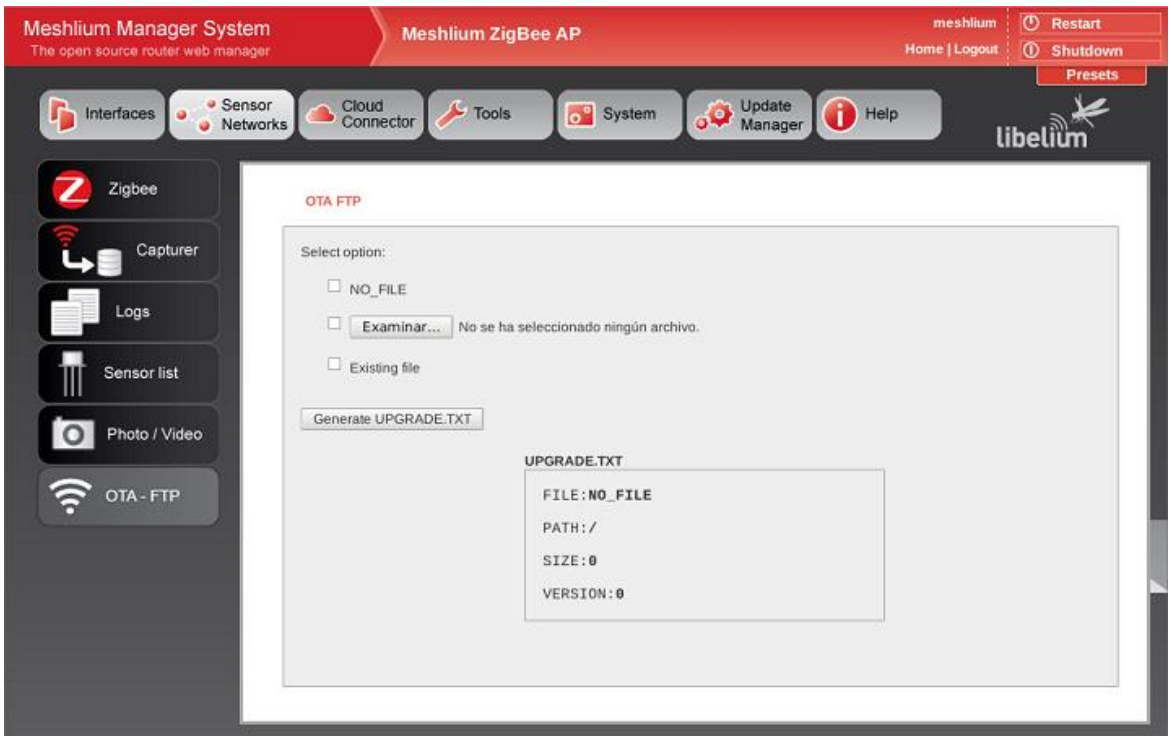

Ilustración 44: Pantalla para el traspaso de ficheros a Meshlium mediante FTP

7.2.4 Herramientas

En esta sección podemos encontrar numerosas funcionalidades. En este documento solo se hace referencia a la sección PhpMyAdmin, que nos permite acceder a la base de datos de la pasarela y consultar los datos almacenados en ella, eliminarlos, o exportarlos.

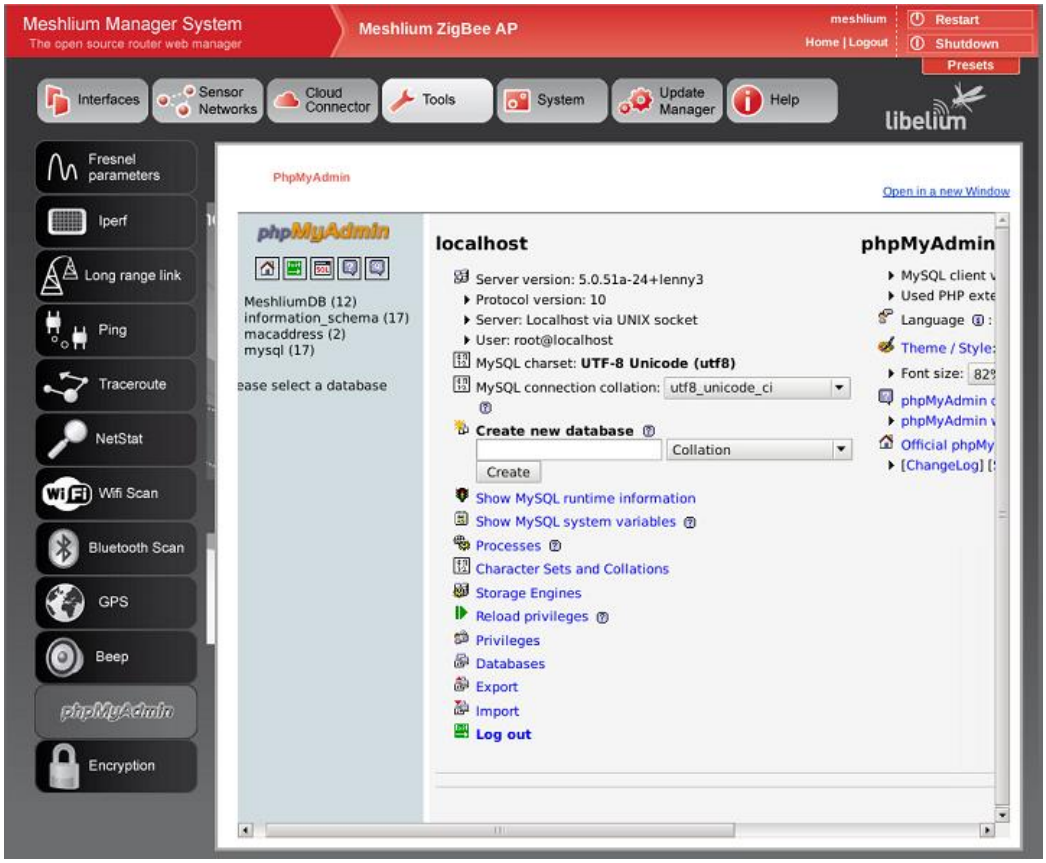

Ilustración 45: Gestor PhpMyAdmin de Meshlium

El resto de funcionalidades que aporta Meshlium no son especialmente relevantes para detallarlos en este documento.

7.3 Topología de la red

Desde un principio se ha tratado de formar una topología en estrella, en la que todos los nodos se comunicaran directamente con la pasarela, sin necesidad de existir nodos enrutadores de paquetes. Las razones por las que se ha intentado mantener esta topología son evidentes:

- Las distancias entre los nodos no es tan grande como para encontrar problemas de comunicación entre ellos y la pasarela.
- Se minimiza el número de dispositivos que forman la red, abaratando por tanto los costes.
- Se evita la necesidad de que algunos nodos sean enrutadores. Este tipo de nodos tienen que mantenerse siempre despiertos a la espera de recibir mensajes de otros nodos. Al no haber nodos routers en la topología en estrella, se ahorra energía.

- La configuración de los dispositivos y el funcionamiento de la red son más sencillos.

Sin embargo, y tras realizar diversas pruebas, se ha comprobado que no todos los nodos conseguían comunicarse correctamente. A pesar de que las distancias no son grandes, en el escenario en el que nos encontramos hay multitud de obstáculos que impiden la visibilidad entre los nodos y la pasarela. Es por ello que, finalmente, se ha optado por formar una red con topología en árbol, en el que dos de los nodos se comportan como routers. Además, se ha introducido un nodo adicional que no toma medidas de irradiancia y tiene como única finalidad la de enrutar paquetes de otros nodos. Podrían haberse introducido más, pero tras muchas pruebas se ha comprobado que, con esta topología y la ubicación dada a los nodos, todos eran capaces de comunicarse y enviar sus datos a la pasarela.

En la siguiente imagen puede apreciarse la ubicación de cada uno de los nodos en los laboratorios del departamento de Ingeniería de Sistemas y Automática de la Escuela Técnica Superior de Ingeniería de Sevilla. En total se han implementado ocho nodos, que aportan la suficiente información para los objetivos que persigue este proyecto. Entre todos ellos cubren una superficie total de 100x240 metros aproximadamente. En la esquina inferior izquierda de la imagen, se muestra una leyenda que indica si los nodos son routers, dispositivos finales, o la pasarela.

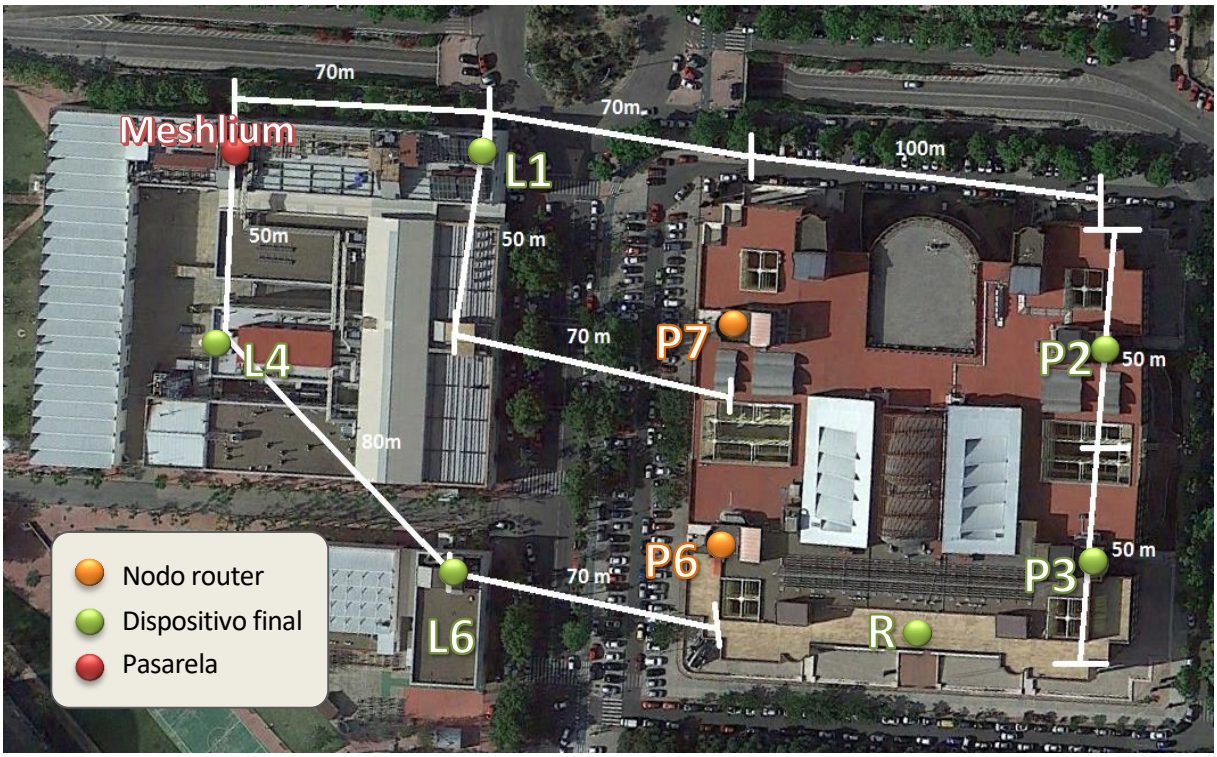

Ilustración 46: Topología y distribución de nodos de la red implementada

Como puede apreciarse en la imagen, cada uno de los nodos posee un identificador, utilizado para distinguir cada nodo dentro de la red y en los datos que lleguen a la pasarela. Además, cada uno de ellos dispone de una dirección MAC Zigbee. Esta dirección es la que debe indicarse como destino en los paquetes de datos que mandan los sensores Waspmote. En la siguiente tabla se adjuntan las direcciones de todos los nodos, ya que su conocimiento resulta indispensable si se desea realizar programación remota, como se verá más adelante.

Nodo de la red	Dirección MAC Zigbee
Pasarela Meshlium	0013A200 – 40C320A3
L1	0013A200 - 40E6063F
L4	0013A200 - 40E6063C
L6	0013A200 - 40E60645
P6	0013A200 - 40E60641
P7	0013A200 - 40E60656
P3	0013A200 - 40E6064B
P2	0013A200 - 40E60655
R	0013A200 - 40E6063E

Tabla 13: Direccionamiento Zigbee de los nodos

A continuación, se adjuntan fotos de la pasarela y de cada uno de los nodos, emplazados en sus ubicaciones correspondientes.

Ilustración 47: Pasarela Meshlium

Ilustración 48: Nodo L4

Ilustración 49: Nodo L1

Ilustración 50: Nodo L6

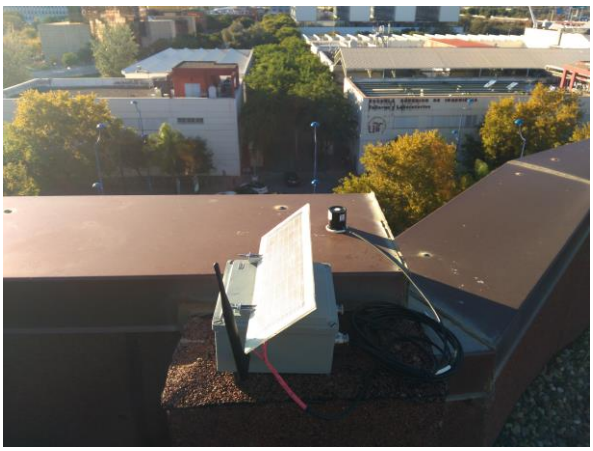

Ilustración 51: Nodo P6

Ilustración 52: Nodo P7

Ilustración 53: Nodo P2

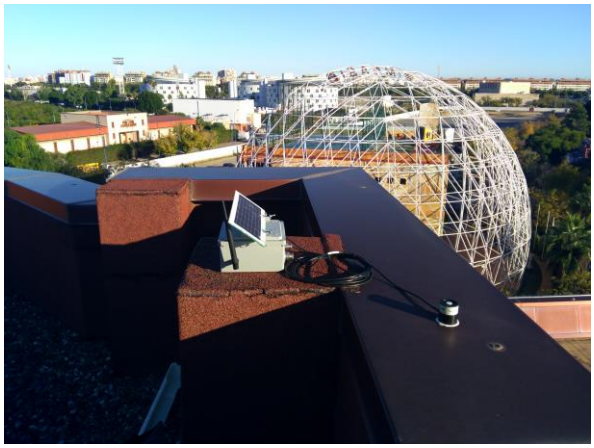

Ilustración 54: Nodo P3

7.4 Programas implementados

La realización de este proyecto ha abarcado muchos meses de pruebas de diferentes topologías y distintos programas implementados en los sensores, a fin de llegar a la mejor solución que optimizara los recursos y ofreciera la mayor cantidad de información posible. Ante todo, se ha buscado que una vez puestos en marcha tras el periodo de prueba, no sea necesario volver a configurarlos (exceptuando condiciones anormales en los que se produzca algún tipo de fallo).

Finalmente, se han implementado dos programas distintos en los sensores, uno para los nodos que actúan como router y otro para los que actúan como dispositivo final. En este apartado se detalla el funcionamiento de cada uno y se muestra su correspondiente diagrama de flujo. El código [18] de los mismos puede encontrarse en los anexos de este documento.

7.4.1 Dispositivos finales

La primera vez que se enciende cada nodo y ejecuta su software, se realiza la inicialización de las variables del programa y de todos los módulos necesarios para ejecutar las tareas: el puerto USB (para depuración), la placa de prototipado para utilizar el convertidor analógico digital, el módulo de comunicación Zigbee y el reloj del microcontrolador. Además, el RTC se inicializa con una fecha y hora que se introduce a mano y es provisional: más tarde, el nodo intentará sincronizar su reloj con el de la pasarela Meshlium

Después, se ejecuta una función que comprueba si el nodo se ha unido correctamente a la red creada por la pasarela, así como los parámetros de la misma. Por útimo, se calcula el número del día del año en el que se encuentra a partir de la fecha del RTC. Con el valor obtenido se buscará en una matriz los valores horarios del orto y el ocaso correspondientes al día actual. Esta matriz contiene dichos valores para todos los días del año, sin contemplar años bisiestos, en horas GSM, por lo que se tiene en cuenta el horario español de invierno o verano, según corresponda, a través de una función que lo calcula. La finalidad de esta operación es calcular la hora a la cual se pone el sol cada día, para dormir al nodo y ahorrar batería durante las horas en las que no incide la luz del sol. Al calcular la hora del amanecer, es posible realizar una diferencia de tiempos y obtener el tiempo que el nodo debe permanecer dormido, para que despierte cuando salga el sol.

Tras la primera inicialización comienza el bucle que se repetirá de manera infinita. En su ejecución, el nodo realiza una lectura de los voltios proporcionados por el convertidor analógico digital de la placa, conectado al sensor de irradiancia. Dicha medida se convierte a Watios y se introduce en una trama junto a la temperatura actual (proporcionada por el RTC) y la batería restante del dispositivo; se realiza el envío de la trama, con un máximo de tres reintentos en caso de fallo. Tras el envío se comprueba si ya es la hora del ocaso para dormir al nodo hasta el día siguiente; en caso contrario, solo duerme 30 segundos y comienza de nuevo el bucle. Cada vez que el nodo despierta tras haber dormido toda la noche, vuelve a calcular los nuevos horarios de orto y ocaso correspondientes al día actual.

El código de este programa puede encontrarse en el Anexo A.

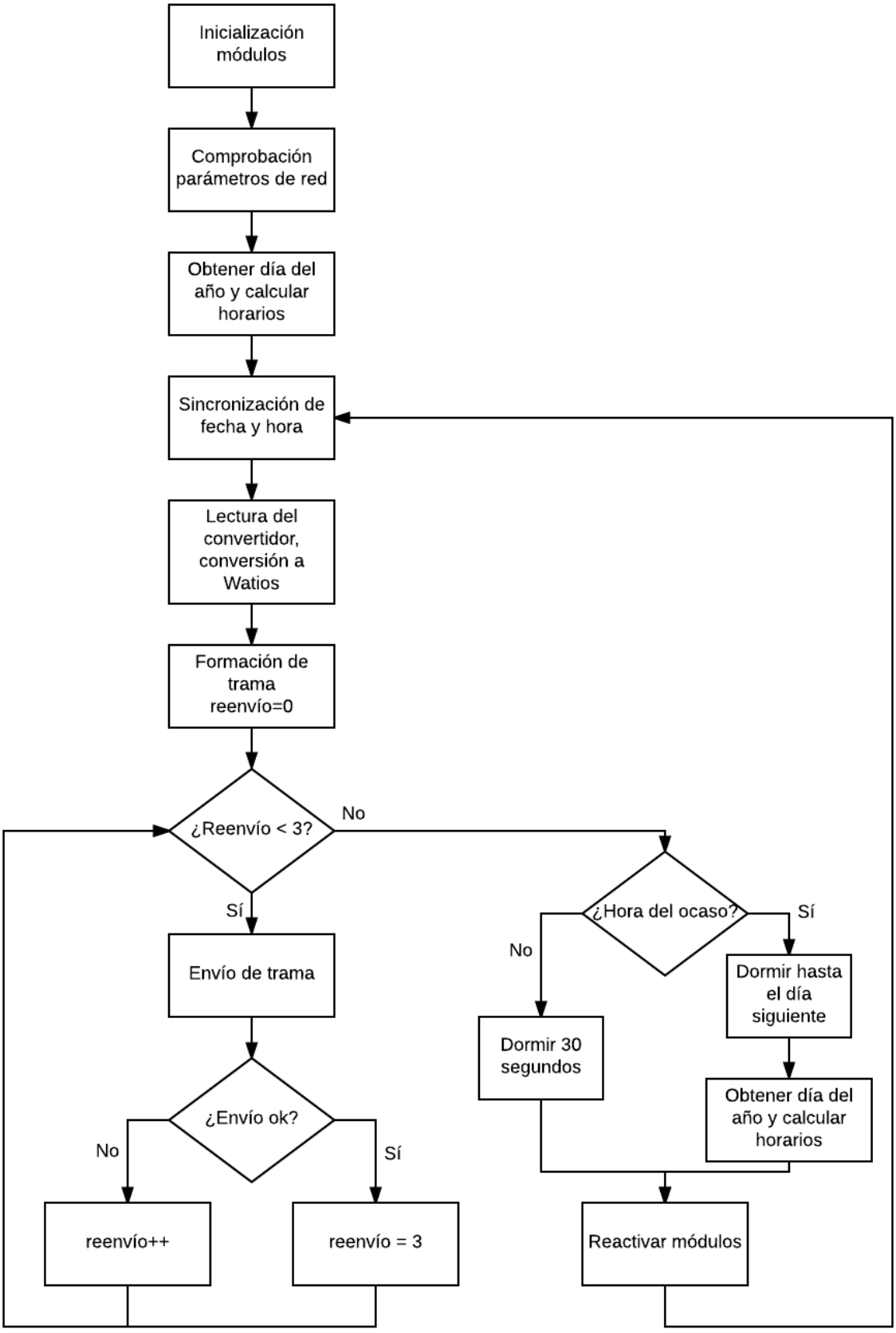

Ilustración 55: Diagrama de flujo del comportamiento de un dispositivo final

7.4.2 Nodos enrutadores

El funcionamiento de los nodos enrutadores es muy similar al del resto de nodos, con una única diferencia: en lugar de hibernar al nodo durante 30 segundos tras el envío de la trama, éste permanecerá a la espera durante 25 segundos, escuchando tramas de otros nodos para reenviarlas a la pasarela. Los nodos enrutadores solo serán hibernados a la hora del ocaso para guardar la batería que sea posible, ya que no tiene sentido permanecer escuchando durante ese periodo. El resto del funcionamiento se mantiene. En el siguiente diagrama de flujo solo se muestra el detalle del bucle, ya que la inicialización es igual que en el caso anterior.

El código de este programa puede encontrarse en el Anexo B

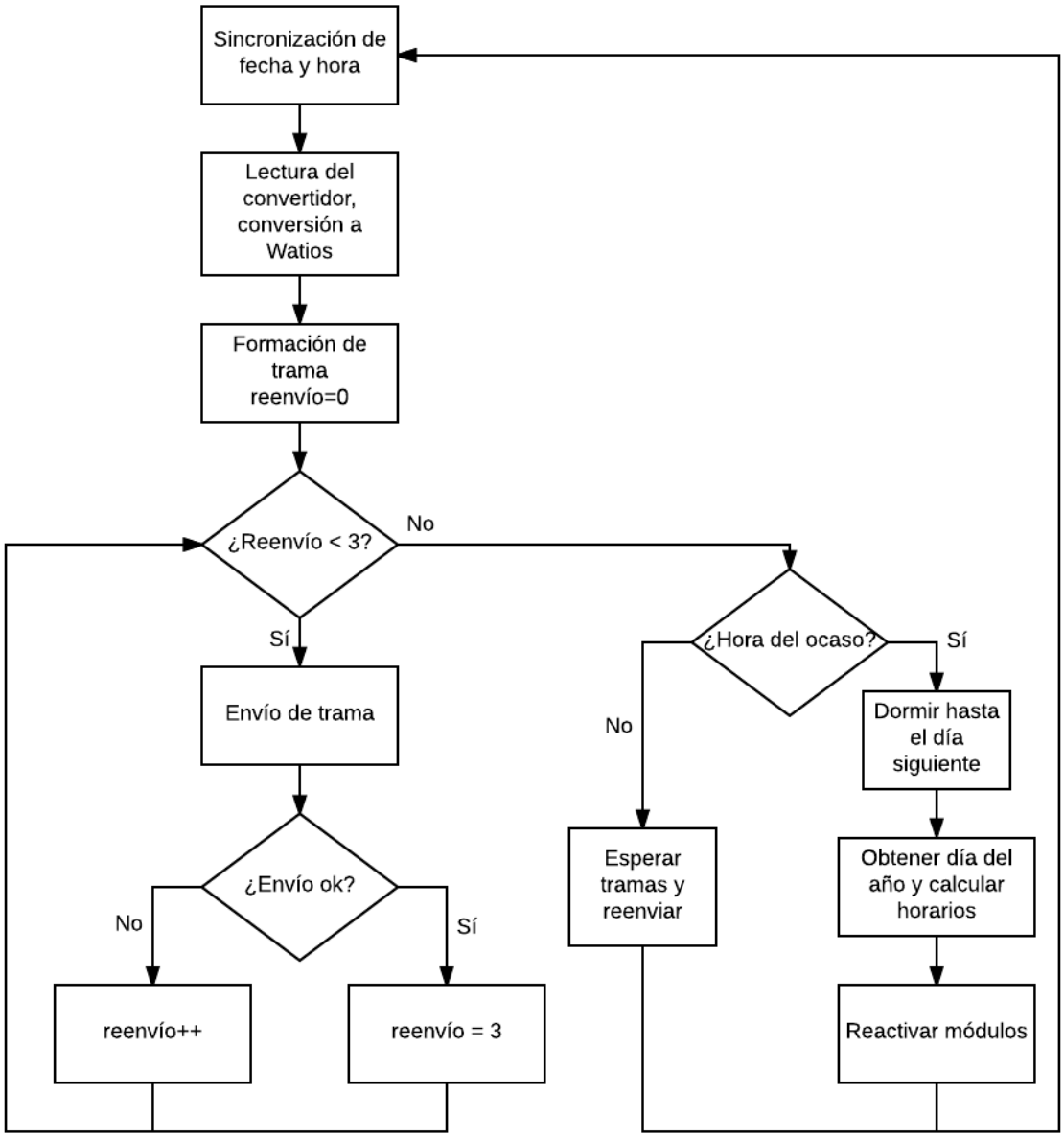

Ilustración 56: Diagrama de flujo del comportamiento de un nodo enrutador

7.4.3 Recogida de datos

A fin de facilitar el trabajo del resto de grupos implicados en el proyecto CODISOL, se ha desarrollado un pequeño programa bastante sencillo, que se encuentra ejecutándose constantemente y cuya función es meramente volcar el contenido de la base de datos de Meshlium en un fichero. Los parámetros irrelevantes se ignoran, manteniendo solo los datos que resultan interesantes para el análisis.

El programa se ha codificado en C, usando la API MySQL C para comunicaciones con el servidor MySQL de la base de datos de Meshlium. A través de esta API pueden utilizarse librerías que permitan este tipo de operaciones. En este caso se ha hecho uso de la libreria libmysqlclient, la versión cliente.

La instalación de la misma se ha realizado a través de su código fuente y se ha construído la API manualmente, ya que de intentándolo de otra manera se han producido fallos.

Para compilar el programa utilizando esta librería:

```
gcc -o conexion conexion.c -I/usr/local/mysql/include -L/sur/local/mysql/lib -
lmysqlclient
```

El código de este programa puede encontrarse en el Anexo C.

7.5 La OTAP de Libelium

La OTAP [3] de Libelium ha sido diseñada para trabajar exclusivamente con la plataforma Waspmote. Los módulos radio de los sensores, además de facilitar la comunicación entre ellos, permiten realizar programación OTA desde cualquier punto de la red. Asimismo, es posible modificar la programación de un nodo ejecutando una nueva versión de un programa, sin necesidad de acceder físicamente al dispositivo. En el presente proyecto resulta de gran utilidad utilizar este método, ya que los nodos se encuentran ubicados en tejados y terrazas, protegidos dentro de cajas estanca. De esta manera, no es necesario desmontarlas cada vez que se desee realizar alguna modificación del código.

Libelium incluye en su API una implementación de OTAP, mediante la cual pueden ejecutarse órdenes por linea de commandos. Soporta módulos XBee que operan con diferentes protocolos: 802.15.4, Zigbee, Digimesh, RF 900 y RF 868, además de funcionar también con GPRS y Bluetooh.

Es muy importante indicar que no es posible realizar programación remota a través de la pasarela Meshlium en caso de utilizar módulos de comunicación Zigbee. La razón de este pequeño inconveniente se debe a diferencias de firmware: Meshlium usa el modo de coordinación AT, y la shell de OTA que se emplea para realizar estas tareas necesita un modo API. Por ello, para llevarlas a cabo, es necesario utilizar el dispositivo *Waspmote Gateway* citado en el capítulo anterior.

7.5.1 Ventajas de la OTAP implementada por Libelium

Como ya se ha comentado, trabajar mediante programación remota facilita muchos aspectos de la puesta en marcha y mantenimiento de una red de sensores. Concretamente, la OTAP que implementa Libelium permite llevar a cabo diversas actividades, como son:

- La actualización o cambio de firmware de manera inalámbrica.
- La recuperación de algún sensor waspmote en caso de que no responda.
- Realizar un descubrimiento de nodos a través de un mensaje de difusión.
- Dependiendo del tipo de módulo XBee utilizado, cabe la posibilidad de seleccionar un canal de radio concreto para desarrollar OTAP con el waspmote destino, que no interfiera con el resto de nodos que componen la red.
- Es posible almacenar en una memoria externa miles de versiones diferentes de programas y nuevo

firmware dentro de cada nodo, y con una simple orden elegir cuál de ellos debe ejecutar.

7.5.2 Formas de comunicación para realizar OTAP

Existen varias topologías a la hora de realizar la reprogramación de los nodos, dependiendo del número de saltos desde cada uno de ellos hasta la pasarela, del protocolo de comunicación utilizado y de cómo se crea la subred en caso de querer reprogramar varios nodos (cuando se usan commandos a direcciones multicast y difusión).

Cuando los nodos son accedidos en un solo salto desde la pasarela, estaremos hablando de acceso directo. Este es el caso de, por ejemplo, el protocolo IEEE 802.15.4. Cuando los nodos son accedidos en dos o más saltos, hablaremos de acceso indirecto. En este caso, algunos de los nodos deberán reenviar los paquetes enviados por la pasarela para que éstos alcancen su destino. Se trata de la forma de comunicación que utiliza el protocolo Zigbee empleado en este proyecto, y será este el caso que se explique en el presente documento.

Comunicación unicast. Antes de comenzar con el proceso OTA, la pasarela manda un paquete especial al nodo que se desea reprogramar, especificando una clave de autenticación. Ésta será utilizada por el nodo para validar si los paquetes llegan desde una fuente fiable. En una red multisalto no pueden modificarse ni el canal ni la clave de encriptación, de lo contrario, el paquete nunca llegaría a su destino. En esta topología, por tanto, la información mandada por la pasarela hasta el nodo es la clave de autenticación y el nuevo firmware, y no necesita indicar ningún otro parámetro.

Ilustración 57: Comunicación unicast en una red multisalto

Comunicación multicast. Para enviar nuevo firmware en modo multicast, lo que realmente se hace es crear una subred formada por los nodos que se desean reprogramar y, entonces, mandar el nuevo programa a difusión. Mediante esta forma de proceder, los dispositivos no envían paquetes de confirmación, por lo que será necesario esperar al final del proceso para comporbar si todos los nodos deseados han recibido el nuevo firmware correctamente.

En una topología multisalto, no pueden modificarse ni el canal ni la clave de encriptación, por lo que para crear la nueva subred se modifica la clave de autenticación, para distinguir entre nodos implicados o no en la reprogramación. Los nodos elegidos cambiarán su clave, y volverán a reestablecer la antigua cuando el proceso acabe. Por lo tanto, los datos que manda la pasarela a los nodos son la nueva clave de autenticación, la clave antigua y el nuevo firmware.

Ilustración 58: Comunicación multicast en una red multisalto

Comunicación mediante difusión. Al igual que en el caso anterior, no se reciben ACK por parte de los nodos, por lo que es necesario esperar a la finalización del proceso para comprobar si se ha realizado correctamente. En este modo, todos los nodos están implicados, por lo que no es necesario modificar la clave de autenticación ni usar dos diferentes. La pasarela, por tanto, solo necesita enviar la clave de autenticación y el nuevo firmware.

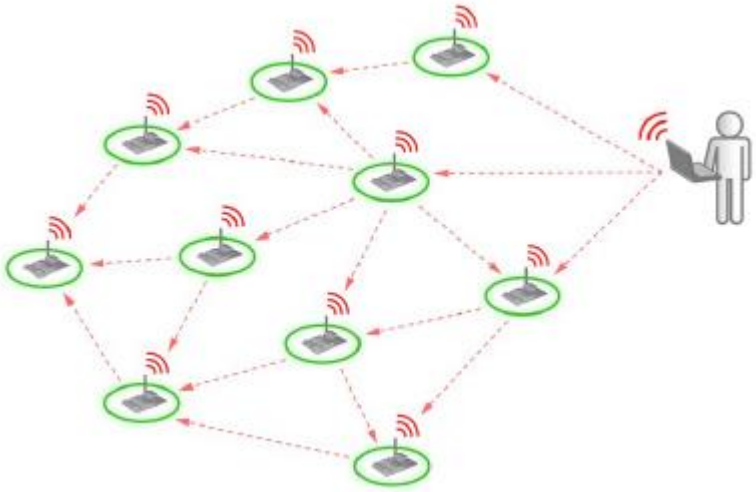

Ilustración 59: Comunicación mediante difusión en una red multisalto

7.5.3 Configuración del entorno

Para que el proceso de la programación remota funcione correctamente, es necesario llevar a cabo una serie de configuraciones, tanto en los dispositivos Waspmote, como en el dispositivo Waspmote Gateway mediante el cual se realizarán estas tareas. Además, es necesario configurar la shell de OTA que permite ejecutar los comandos necesarios para llevar a cabo el proceso.

7.5.3.1 Configuración de la OTA-shell

Para poder realizar la programación remota, es necesario configurar la aplicación *OTA-shell* [3], la cual permite controlar todas las tareas disponibles en la OTAP. Cuando la programación remota se realiza mediante la pasarela Meshlium, el entorno ya se encuentra previamente instalado, configurado y listo para ser usado. Este

método es el más recomendable cuando se trata de desarrollar escenarios reales, pero en el caso de este proyecto no ha sido posible proceder de esta manera, por utilizar módulos de comunicación Zigbee. Por ello, debe seguirse el procedimiento adecuado para trabajar con el Waspmote Gateway.

La versión adecuada que debe utilizare de la shell de OTAP es la vb111 [19].

Para la realización del proyecto en un entorno Linux, es necesario tener Java instalado. Una vez conectado el Waspmote Gateway en un puerto USB, es necesario conocer la ruta que le asigna el sistema. Para obtenerlo, se ejecuta el siguiente comando en un terminal:

```
meshlium:/# ls -l /dev/ttyUSB*
crw-rw---- 1 root dialout 188, 0 may 4 12:17 /dev/ttyUSB0
```

Tras descargar el software de la shell de OTA desde la web de Libelium y descomprimirlo, es necesario modificar el fichero de configuración *xbee.conf,* indicando la ruta obtenida en el paso anterior:

```
# port where the xbee moduel is connected
port = /dev/ttyUSB0
# auth key of network
auth_key = LIBELIUM
# pan ID of network
panID = 0x1234
# xbee model
xbeeModel = DM
# channel number
channel = 0x0F
# encryption of network
encryption = off
# encryption key of network
encryptionKey = 1234567890123456
# name of the file where discarded data goes
#discardedDataFile = data.txt
# Waspmote version
WaspmoteVersion = 12
```

Como puede verse, en el fichero de configuración también pueden modificarse parámetros como la clave de autenticación, el protocolo de comunicación empleado, si se desea activar o no la comunicación encriptada, etc.

Una vez modificado el fichero ya puede ejecutarse el programa, introcudiendo en el terminal:

```
meshlium:/mnt/user/ota/otap# ./otap
You have to specify one command
usage: otap
 -delete_program          deletes a specified program
 -get_boot_list           get the boot list of nodes
 -info_program            shows info about hex file
 -scan_nodes              scan for active nodes
 -send                    send firmware to nodes
 -start_new_program       start specified program
 -version                 show version of the program
```

7.5.3.2 Configuración de los Waspmote

Por otra parte, la preparación de cada uno de los dispositivos Waspmote es sencilla. En su version v11 no estaban preparados para realizar operaciones OTA y es necesario descargar una nueva version API. Sin embargo, en la version v12, que se usa en este proyecto, ya se encuentran configurados para ello. Lo único necesario es dotar a cada dispositivo de una tarjeta de memoria extraíble.

Los dispositivos Waspmote disponen de cuatro sistemas de memoria diferentes: la memoria RAM guarda las variables e instrucciones del programa que actualmente se ejecuta; le memoria EEPROM se usa para almacenar variables y ciertas banderas usadas en los programas y que necesitan conservarse tras un reinicio del dispositivo; la memoria flash es usada para almacenar el programa binario que se ejecuta actualmente; por útlimo, es posible insertar una memoria extraíble (tarjeta micro SD) que permite gestionar ficheros. Los programas nuevos que se manden a los nodos deberán estar almacenados en ella.

Por tanto, para poder llevar a cabo la programación remota, se ha equipado a cada uno de los nodos con una

tarjeta micro SD de 2GB. Cuando se envía el nuevo software a los nodos, éstos lo almacenan en la memoria extraíble, y tras la recepción del comando adecuado se les ordena posteriormente que inicialicen el programa. Entonces, los nodos lo copian desde la tarjeta SD a la memoria flash.

Debe asignarse a cada Waspmote (mediante código) la clave de autenticación que soporte la reprogramación. Por defecto ya contienen una almacenada en la memoria EEPROM, configurada como "LIBELIUM". Además, es necesario que cada nodo tenga asignado un identificador.

Es necesario saber que la programación OTA usa tramas especiales, cuya utilización debe evitarse en los programas que se usen en el sensor, y que para poder realizar esta programación no se deben utilizar las direcciones de memoria EEPROM destinadas a almacenar banderas y direcciones especiales, o de lo contrario el proceso no funcionaría correctamente.

7.5.3.3 Configuración del Waspmote Gateway

Por último, solo resta configurar el Waspmote Gateway, que actuará de pasarela para enviar el nuevo firmware a cada nodo. Para modificar los parámetros necesarios debe usarse el software *X-CTU* [20] [21] de Digi, los fabricantes del módulo XBee.

Una vez descargado e instalado el software, solo es necesario conectar el Waspmote Gateway al ordenador y tener en cuenta el puerto serial al que se ha conectado. Debe conectarse un módulo de comunicación XBee en el Waspmote Gateway y comenzar la búsqueda de dispositivos radio mediante la funcionalidad que ofrece XCTU. Tras la búsqueda se nos muestran los resultados obtenidos y podrá iniciarse el modo de configuración con el dispositivo deseado. En él podremos modificar los parámetros del módulo y deberá configurarse su tasa a 38400 baudios.

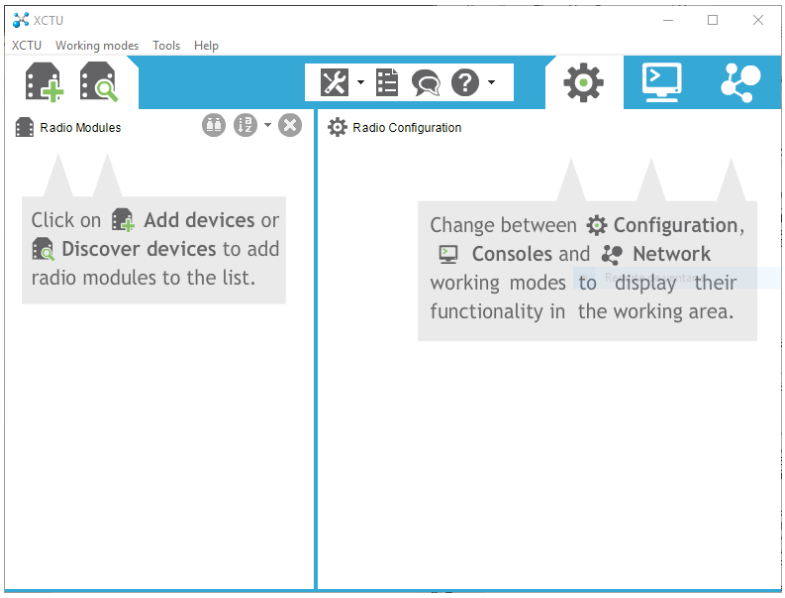

Ilustración 60: Software XCTU

7.5.4 Realizando OTAP

Los pasos para llevar a cabo la programación remota son los siguientes:

- Localizar el nodo o nodos a actualizar. Para ello se usa la funcion *scan_nodes*, a través de la cual podemos buscar un nodo específico o mandar una petición global buscando nodos disponibles para la reprogramación. Los nodos que estén preparados en ese momento contestarán indicándolo.
- Mandar el nuevo programa. Es posible utilizar el comando *send* con una dirección unicast, multicast, o difusión, dependiendo del número de nodos que se desee reprogramar en un mismo instante. Cada nodo

que reciba el programa informará a la pasarela del éxito del proceso.

- Reiniciar e iniciar el nuevo programa. Para que los nodos ejecuten el nuevo software, la pasarela debe mandarles el comando *start_new_program*. Entonces, cada nodo que lo reciba copiará el programa indicado en la memoria flash y comenzará a ejecutar el nuevo código.

Una consideración debe tenerse en cuenta a la hora de realizar estas operaciones: los programas que deben mandarse a los nodos durante la realización de la programación remota deben ser ficheros hexadecimales. El IDE de Waspmote genera estos ficheros de forma automática cuando un programa se compila o se introduce en el dispositivo. De esta forma, se generan dos ficheros; dentro del directorio /sketchbook/OTA-FILES se almacena el fichero hexadecimal, y en el directorio /scketshbook se guarda el fichero binario normal. Es muy importante no modificar el nombre de los ficheros hexadecimales generados, para que no se produzcan errores. Éstos tienen como tamaño máximo 7 caracteres, por lo que si el nombre original del programa es más largo, se truncará a esta cifra.

A continuación se muestra la utilización comandos y la realización de los pasos anteriormente citados, y otros pasos de interés.

En primer lugar se realiza la búsqueda de nodos. Cuando se ejecuta el comando, se muestran las opciones disponibles: se puede realizar una petición unicast, multicast, o a toda la red mediante difusión.

```
meshlium:/mnt/user/ota/otap# ./otap -scan_nodes
 --mac <000000000000000>         mac address, 16 hex characters, separated with commas
 --mode <UNICAST|BROADCAST>      scanning mode
 --time <seconds>                timeout seconds (optional, by default 10)
```

n el siguiente ejemplo se muestra la búsqueda de nodos durante tres segundos. Como respuesta se obtiene una lista de nodos disponibles mostrando su dirección mac, su ID y el programa que tiene implementado.

```
meshlium:/mnt/user/ota/otap# ./otap -scan_nodes --mode BROADCAST --time 3
   Total Nodes: 2 - Time elapsed 6s
   0 - Node 0013a2004061097c - P2 - prog001 - READY
   1 - Node 0013a20041615623 - P3 - prog001 - READY
```

El siguiente paso es obtener el *bootlist.* Se trata de un fichero especial que contiene la información de todos los programas disponibles en el Waspmote. Cada vez que se manda un nuevo programa al dispositivo, se añade una línea de referencia al mismo en este fichero. Para obtener dicha información, se usa el comando *get_boot_list,* el cual devolverá el PID y la fecha de todos los programas disponibles en la memoria externa.

```
meshlium:/mnt/user/ota/otap# ./otap -get_boot_list
 The command needs required options
 usage: otap -scan_nodes
 --mac <000000000000000>         mac address, 16 hex characters
 --mode <UNICAST>                target mode
```

Para preparar el envío del programa, primero se debe obtener cierta información para saber cuántos paquetes y cuánto tiempo estimado tomará la subida del código. Para ello puede utilizarse el comando *info_program.*

```
meshlium:/mnt/user/ota/otap# ./otap -info_program --file PROG001.hex
 Name:                   PROG001.hex
 Date:                   Wed Nov 27 13:24:18 CEST 2016
 Size:                   2100 bytes
 Packets to send:   17
 Estimated time:    3.77 seconds
```

Una vez seleccionado el nodo o nodos a actualizar y obtenida la información necesaria, se utiliza el comando *send* para realizar el envío.

```
meshlium:/mnt/user/ota/otap# ./otap -send
 The command needs required options
 usage: otap -send
 --deliveries <times>                    number of times each packet is sent
 --file <firmware_file.hex>              file containing the firmware
 --mac <0000000000000000>                mac address
 --mode <UNICAST|MULTICAST|BROADCAST>    transmission mode
 --new_authkey <LIBELIUM>                8 ascii characters
 --new_channel <00|0x00>                 channel in decimal or hexadecimal format
 --new_enckey <1234567890123456>         1 ascii characters
 --pid <PROGRAM_1>                       program identifier, 7 characters in length
```

Cuando los nodos deseados hayan recibido el nuevo programa, se utiliza el siguiente comando para dar inicio al mismo:

```
meshlium:/mnt/user/ota/otap# ./otap -start_new_program
 usage: otap -start_new_program
 --mac <0000000000000000>                mac address, 16 hex characters
 --macs_file <list.txt>                  file containing list of macs
 --mode <UNICAST|MULTICAST|BROADCAST>    star mode
 --pid <program_name>                    the program identifier
```

A veces, resulta interesante la opción de poder borrar uno de los programas almacenados en el nodo. Para ello puede utilizarse el comando *delete_program*:

```
meshlium:/mnt/user/ota/otap# ./otap -delete_program
 The command needs required options
 usage: otap -delete_program
 --mac <0000000000000000>                mac address, 16 hex characters
 --macs_file <list.txt>                  file containing list of macs
 --mode <UNICAST|MULTICAST|BROADCAST>    start mode
 --pid <program_name>                    the program identifier
```

Los nodos pueden informar del resultado de la ejecución de cada uno de los comandos vistos mediante respuestas de éxito o de fallo, indicadas en la siguiente tabla:

Respuestas de éxito	
READY	Respuesta tras la búsqueda de nodos, cuando se encuentra preparado para ser reprogramado
PROGRAM RECEIVED OK	Respuesta cuando un programa se recibe en el nodo correctamente
START WITH FIRMWARE OK	Respuesta del nodo antes de reiniciarse cuando se le ordena iniciar un nuevo programa
NEW PROGRAM RUNNING	Respuesta del nodo cuando el programa nuevo se ha iniciado
RESTARTING	Respuesta del nodo antes de reiniciarse cuando no se realiza OTA

Tabla 14: Respuestas de éxito durante la realización de OTAP

Respuestas de error	
PROGRAM RECEIVED ERROR	Respuesta del nodo cuando uno o más paquetes se pierden en la recepción del nuevo programa
PREVIOUS PROGRAM RUNNING	Respuesta del nodo cuando intenta iniciar el nuevo programa y falla. Vuelve a iniciarse el programa estable anterior y el nuevo vuelve a almacenarse
START WITH FIRMWARE ERROR	Respuesta del nodo cuando se ordena que se inicie u nuevo programa indicando erróneamente su PID

Tabla 15: Respuestas de error durante la realización de OTAP

8 ANÁLISIS Y RESULTADOS

Los datos que han sido recogidos por la red de sensores, facilitan mucha información al resto de grupos que trabajan en el proyecto CODISOL. El tratamiento de estos datos para producir modelos y algoritmos no es una tarea a realizar en el presente documento. Sin embargo, a partir de los datos pueden sacarse ciertas conclusiones que serán detalladas en este apartado. Observando las medidas de irradiancia y las horas en las que fueron tomadas, ha sido posible detectar problemas de configuración en la red. Con las gráficas generadas a partir de estos datos es posible visualizar cuándo se han producido cúmulos nubosos. Además, gracias a los valores proporcionados del nivel de batería en cada instante, ha sido fácil detectar problemas en la carga de los sensores.

En los siguientes subapartados se muestran las conclusiones a las que se ha llegado en cada uno de los casos citados. Es necesario indicar que, en algunos apartados, los datos ofrecidos se recogieron durante pruebas de otros programas que no han sido la versión definitiva, pero este aspecto no interfiere en las muestras que se desean enseñar; tan solo varían los tiempos en los que los nodos permanecían despiertos y las medidas de irradiancia son totalmente válidas. Con esto pretende mostrarse también la diferencia entre las distintas versiones de programa que se han ido implementando a lo largo de la realización del proyecto.

8.1 Problemas de conectividad

En la gráfica que se adjunta y se estudia en este apartado, puede verse que los datos recibidos por parte del nodo P2 durante toda una semana son escasos. Esto podría deberse, o bien a que el nodo no tiene visibilidad con la pasarela, o bien a que el dispositivo se ha quedado sin batería y permanece apagado. Sin embargo, la segunda opción queda descartada, porque vuelven a recibirse datos tras periodos de inactividad.

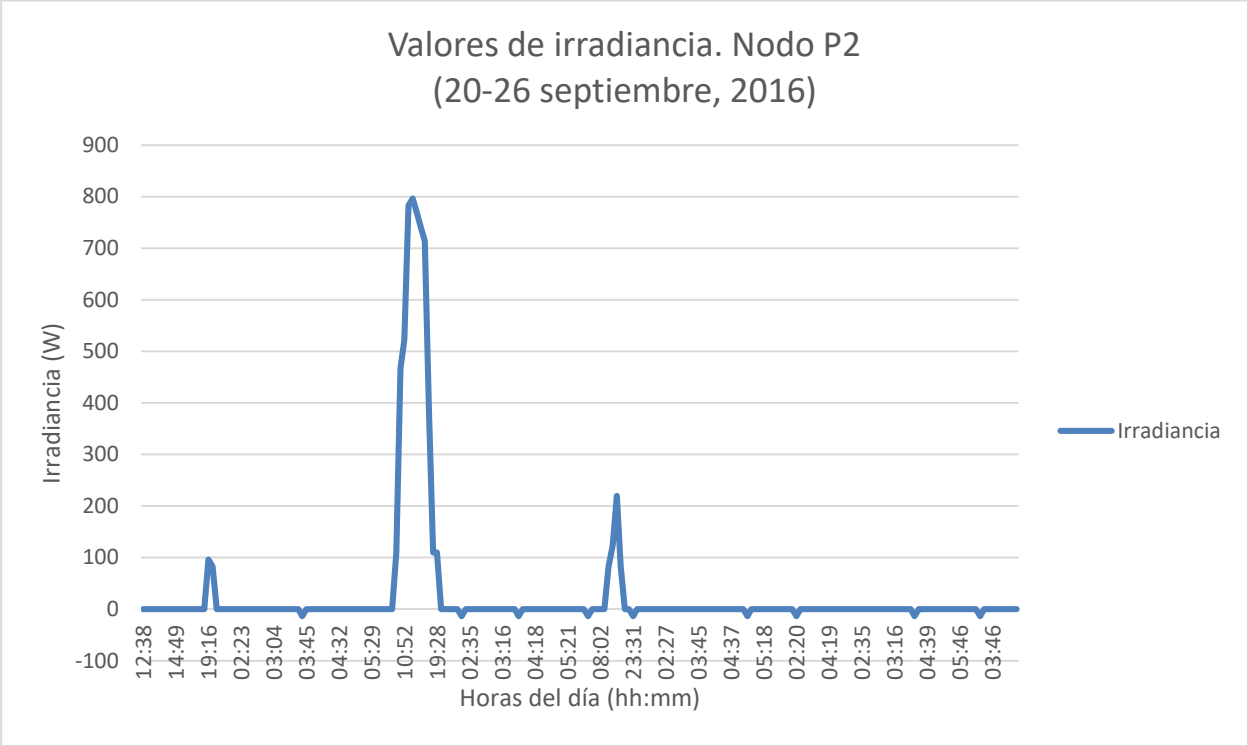

Ilustración 61: Gráfica con ausencia de valores

Esta situación se daba cuando en la red, todos los nodos sensores actuaban como dispositivos finales. Algunos de ellos encontraban obstáculos que les impedían la comunicación directa con la pasarela, por lo que muchas de sus tramas se perdían y nunca se almacenaban en la base de datos de Meshlium.

En la siguiente imagen se muestra el esquema de la red inicial, en la que ningún nodo se comportaba como router. Aparecen marcados los nodos que no podían comunicarse correctamente con la pasarela.

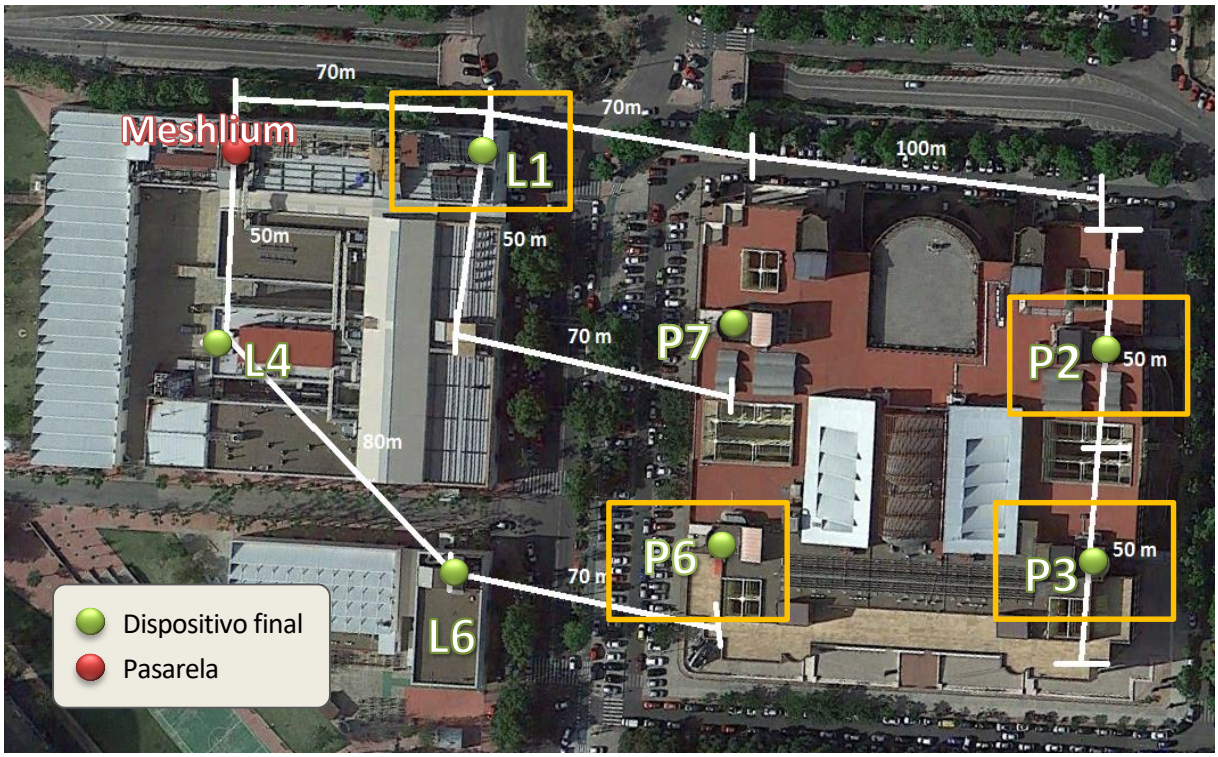

Ilustración 62: Topología inicial de la red de sensores

Para solucionar este problema, se ha hecho un estudio de la evolución de la batería de un nodo enrutador, a fin de comprobar si los niveles podían mantenerse con la ayuda de la placa fotovoltaica, a pesar de no poder dormir al nodo durante todo el día. Los resultados de estas comprobaciones pueden verse en la siguiente gráfica, que muestra los datos obtenidos durante el día nueve de diciembre de 2016. En ella se aprecia que, aunque el consumo del dispositivo es mayor en esta situación, los niveles de batería ascienden en las horas en las que más incide el sol.

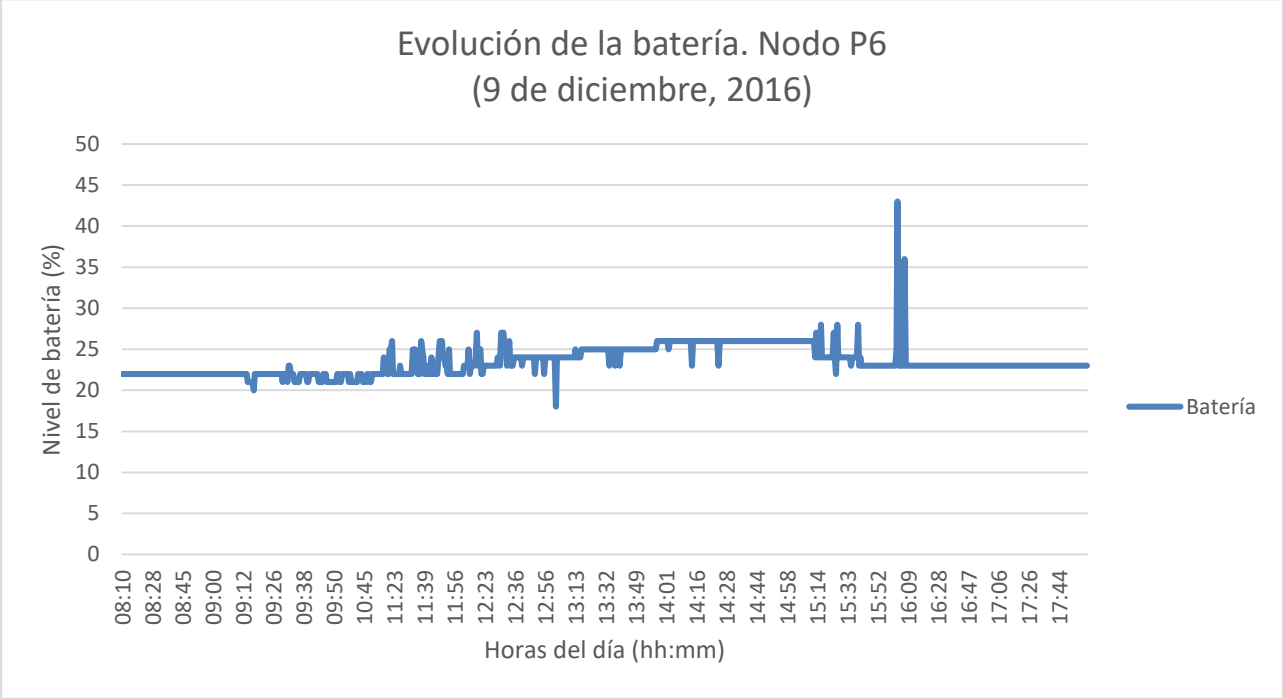

Ilustración 63: Gráfica sobre la evolución de la batería de un nodo router

Tras conocer este resultado positivo, se procedió a implementar la red final, explicada en el capítulo anterior, en la que dos de los routers que toman medidas se comportan también como nodos enrutadores, y se coloca un nodo adicional cuya función es únicamente reenviar tramas sin tomar medidas.

8.2 Problemas de nodos bajo sombra

La siguiente gráfica muestra la evolución de la irradiancia a lo largo del día de las muestras tomadas por el nodo L1, durante los días 23 y 24 de septiembre de 2016. Los valores de irradiancia pasan a ser nulos o negativos e torno a las cuatro de la tarde. Sin embargo, sabemos que en esa franja horaria todavía hay luz. Además, la situación se prolonga durante periodos demasiado largos como para deberse al paso de una nube, y siempre se produce sobre la misma hora. Esta situación se repetía también en otros nodos.

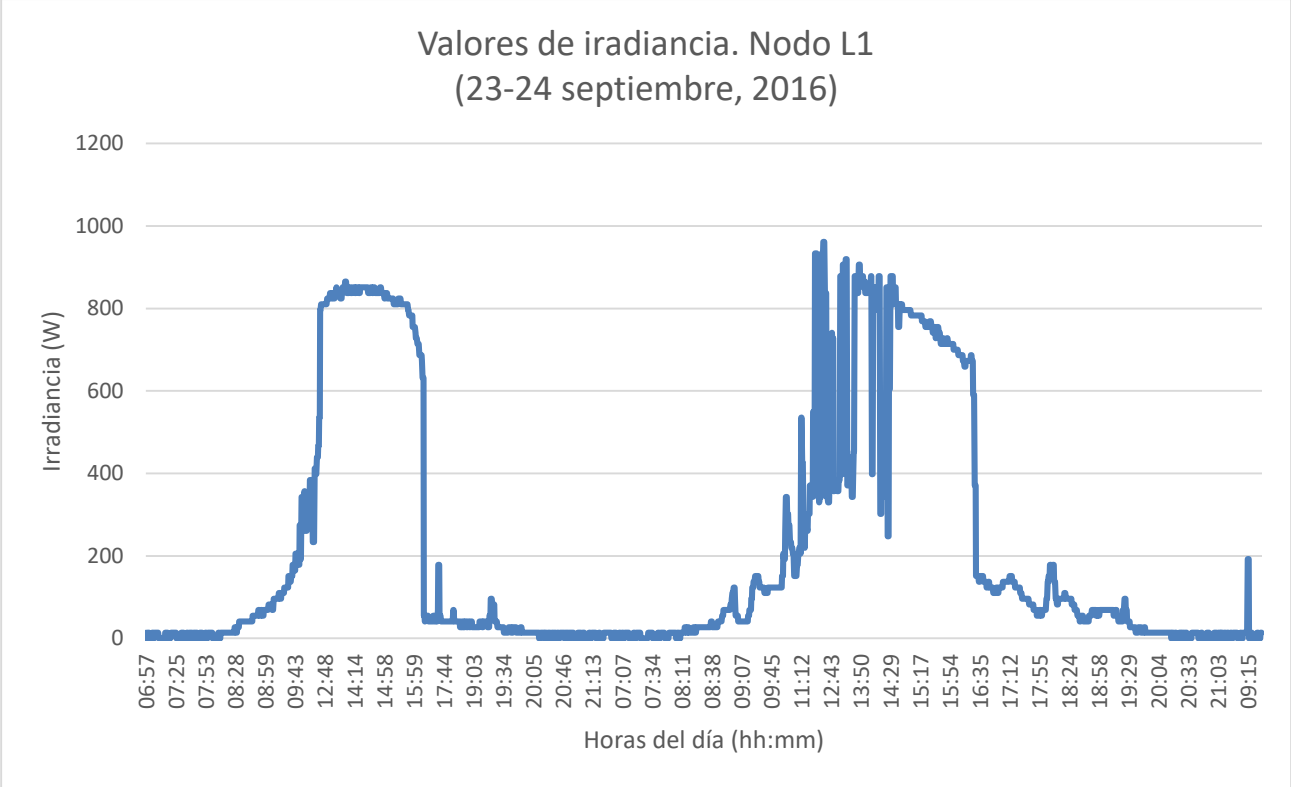

Ilustración 64: Gráfica que muestra la incidencia de sombra sobre un nodo

Estos datos han ayudado a descubrir que, a una cierta hora del día, algunos de los sensores eran cubiertos por una sombra producida por algún obstáculo. Esto resultaba problemático porque bajo estas circunstancias se perdían muestras de irradiancia y, además, la placa fotovoltaica no podía realizar su función. Para solucionarlo se han modificado ligeramente las ubicaciones de algunos de los nodos o, en su defecto, del sensor de irradiancia conectado a ellos.

8.3 Problemas en la conexión de las placas fotovoltaicas

En las gráficas que se adjuntan a continuación, puede observarse que la batería del nodo L6 se descarga durante cuatro días progresivamente hasta llegar casi a agotarse, a pesar de que las medidas de irradiancia indican que el sol incide sobre el sensor. En ningún momento se refleja en los datos que la placa fotovoltaica esté alimentando las baterías del sensor, lo que nos lleva a pensar que está mal conectada o tiene algún tipo de fallo. Debe indicarse que niveles mostrados en la gráfica decaen demasiado porque durante la realización de esas pruebas, el nodo no hibernaba.

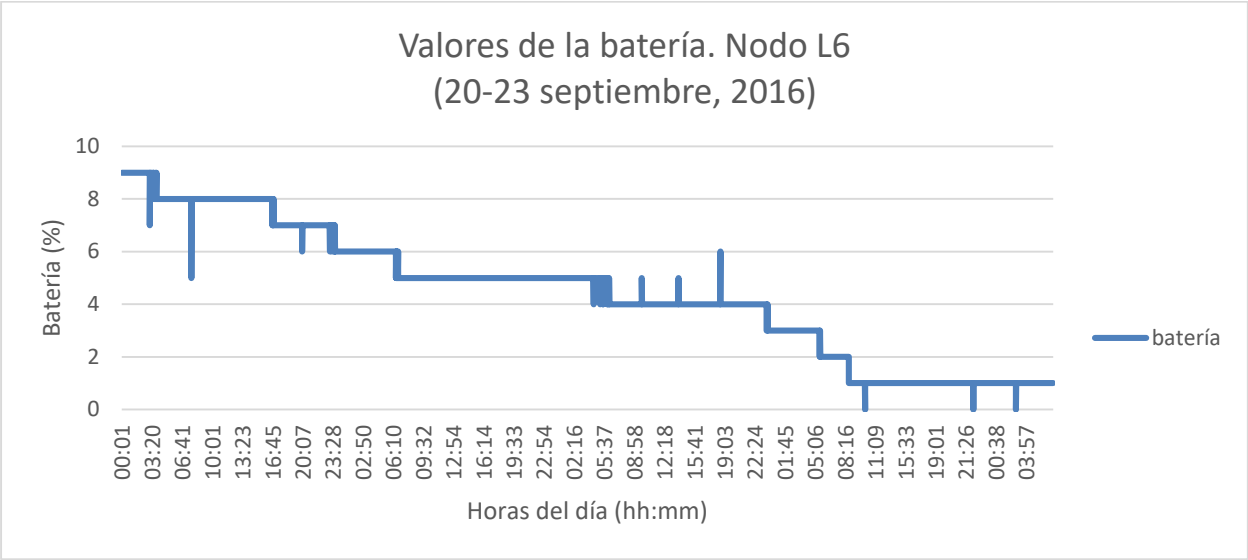

Ilustración 65: Gráfica que muestra la evolución errónea de la batería de un nodo

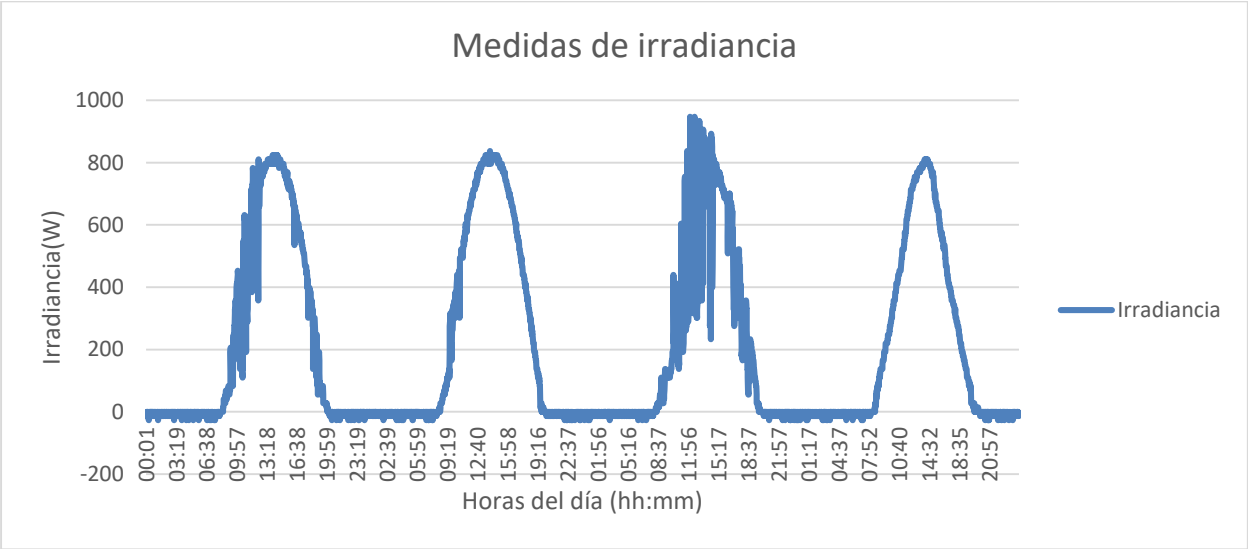

Ilustración 66: Gráfica con valores correctos de irradiancia

Este es un problema que se ha dado en la mayoría de los nodos y se ha solucionado revisandolos uno por uno hasta conseguir el funcionamiento de todas las placas.

En la siguiente gráfica se muestra el funcionamiento correcto de la placa fotovoltaica y se comprueba como, durante dos días, la batería del nodo L1 se recarga.

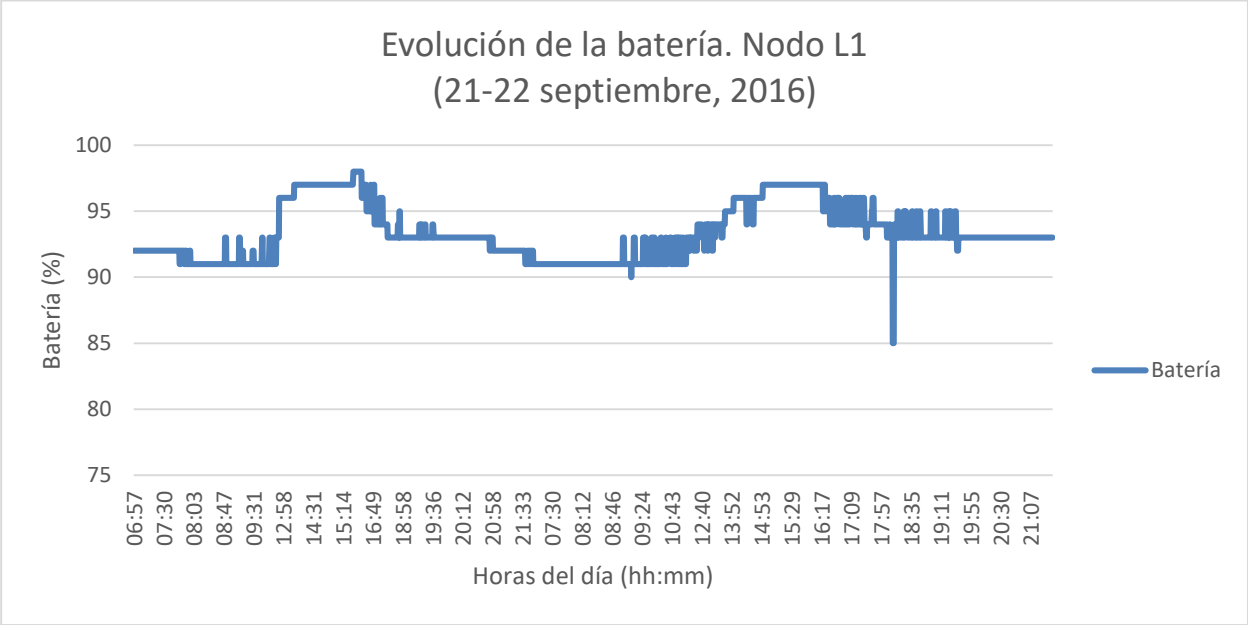

Ilustración 67: Gráfica que muestra la evolución correcta de la batería de un nodo

8.4 Paso de nubes por el cielo

En este apartado se muestran los datos recogidos por el nodo L4, en este caso funcionando todo con normalidad y correctamente. En la segunda curva, correspondiente al día 22 de septiembre, apreciarse cómo varían los valores de la irradiancia, produciéndose fluctuaciones en determinados momentos. Se trata de un indicativo del paso de nubes por encima del nodo, dejándolo durante periodos en sombra. Gracias a este tipo de datos, el resto de grupos del proyecto CODISOL pueden llevar a cabo sus tareas.

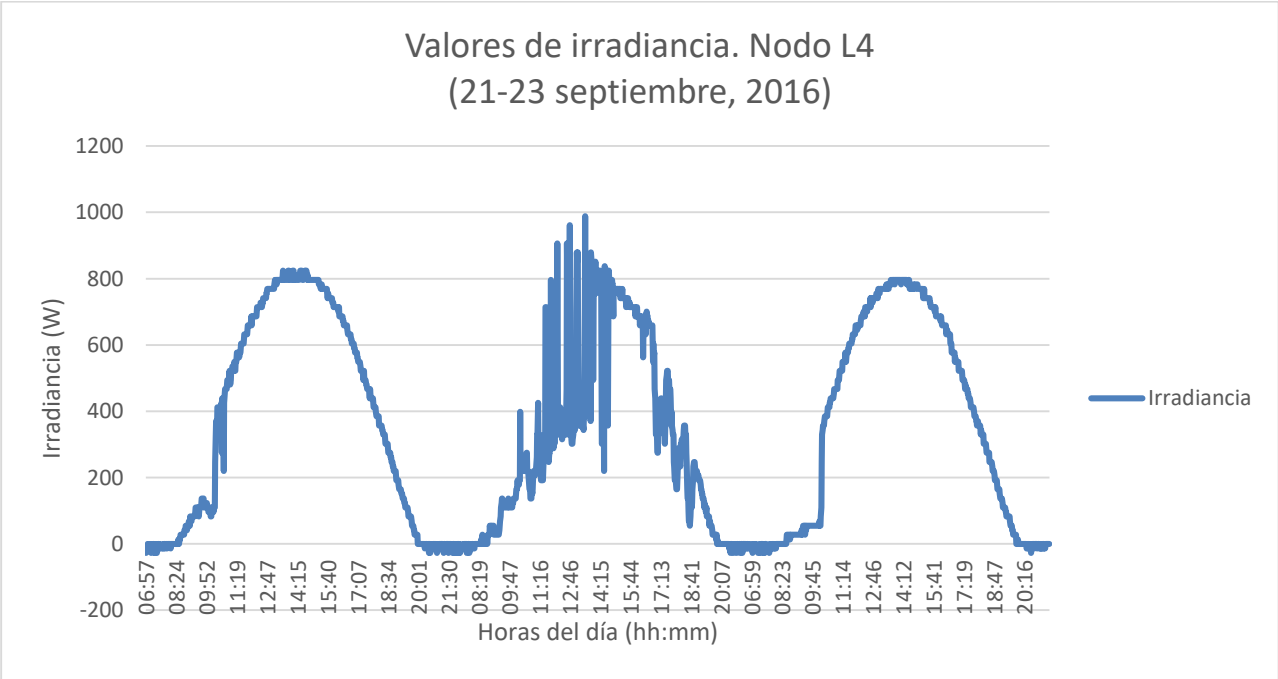

Ilustración 68: Gráfica que muestra el paso de nubes por el cielo

8.5 Comprobación de los datos de irradiancia

Los datos anteriores han sido contrastados y validados gracias a otros datos, aportados por el grupo de energéticos que trabaja en el proyecto CODISOL. Este grupo ya puso en marcha con anterioridad una estación con otro sensor de irradiancia del mismo modelo que los usados en la red de sensores, el cual ha ido recogiendo datos que ya están validados y son fiables.

Ilustración 69: Estación instalada por el grupo de energéticos

Las siguientes gráficas sirven para testificar la comprobación y el contraste de estos datos. La primera muestra los valores recogidos por el sensor en cuestión, el 22 de septiembre de 2016. La segunda muestra los datos recogidos por el nodo L6 de la red de sensores de este proyecto, el mismo día. Como puede apreciarse, los datos en ambas gráficas coinciden.

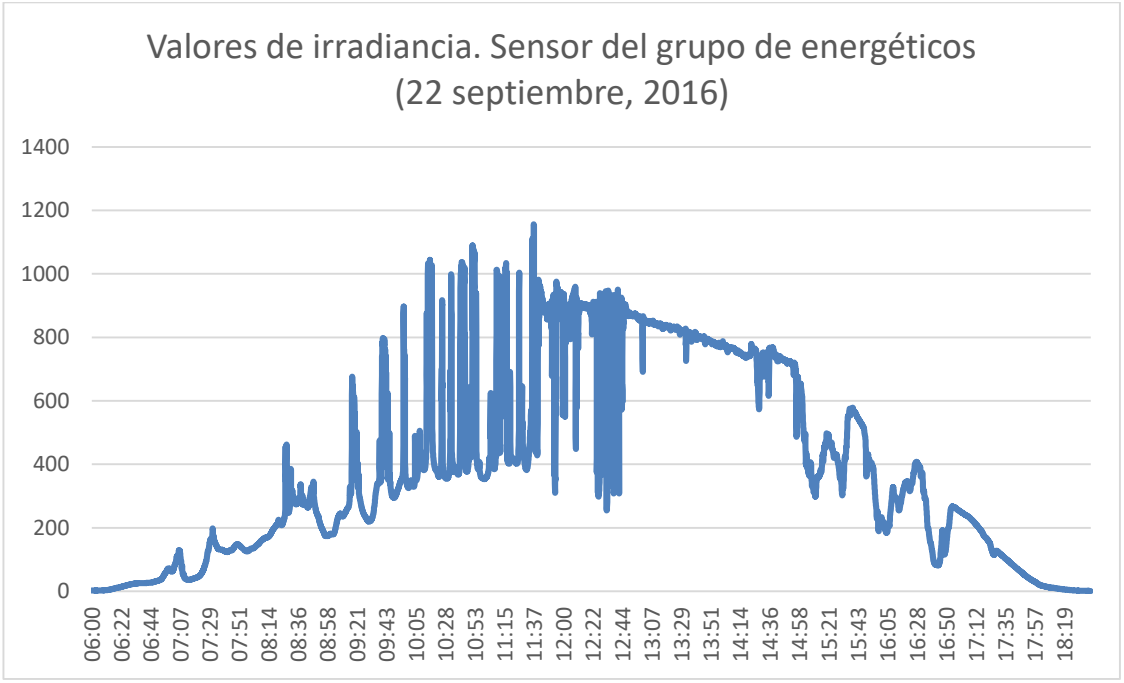

Ilustración 70: Gráfica comparativa, sensor del grupo de energéticos

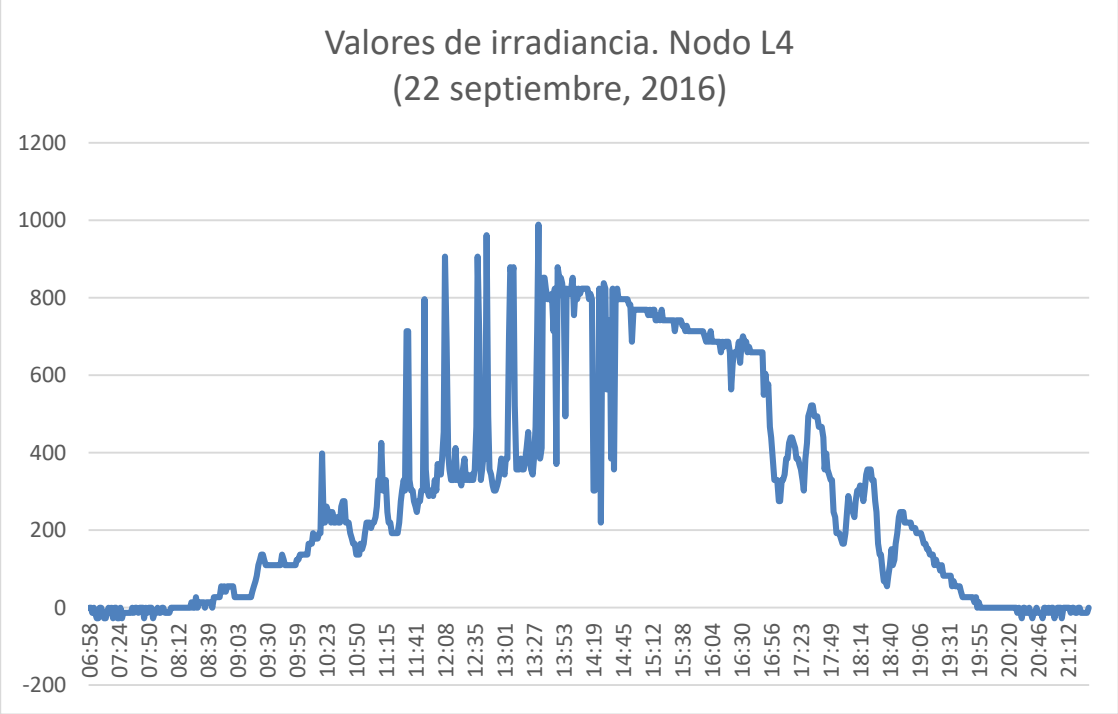

Ilustración 71: Gráfica comparativa, nodo L4

8.6 Presupuesto del proyecto

La siguiente tabla muestra los gastos realizados en la compra de todos los dispositivos y elementos necesarios para llevar a cabo la ejecución de este proyecto.

Producto	Cantidad	Precio unidad (€)	IVA (21%)	Total (€)
Waspmote ZB PRO SMA 5 DBI	11	155,00	358,05	**2.063,05**
Batería recargable 6600mA/h	11	30,00	69,30	**399,30**
Panel solar flexible 7.2V	11	34,00	78,54	**452,54**
Placa de prototipado de Waspmote	11	40,00	92,40	**532,40**
Meshlium Zigbee-PRO-AP	1	690,00	144,90	**834,90**
Waspmote Gateway ZB PRO SMA 5 DBI	1	155,00	32,55	**187,55**

Piranómetro SP1110 Skye Instruments	8	264,00	443,52	**2.555,52**
Transporte y envío de productos	1	50,70	10,47	**61,17**
TOTAL (€)				**7.086,43**

Tabla 16: Presupuesto del proyecto

REFERENCIAS

[1] S. C. Mukhopadhyay, Internet Of Things, 2014.

[2] S.-H. Yang, Wireless Sensor Networks, 2014.

[3] Libelium, Over the air programming guide, 2013.

[4] D. Gislason, Zigbee wireless networking, Newnes/Elsevier, 2008.

[5] T. J. Q. Z. Chonggang Wang, ZigBee® Network Protocols and Applications, Auerbach Publications, 2014.

[6] www.zigbee.org.

[7] Libelium, Zigbee networking guide, 2016.

[8] Libelium, WSN with Waspmote and Meshlium, 2014.

[9] Arduino, www.arduino.cc.

[10] A. Yún, www.arduino.cc/en/Main/ArduinoBoardYun.

[11] A. Ethernet, www.arduino.cc/en/Main/ArduinoBoardEthernet.

[12] MangOH, http://mangoh.io/.

[13] www.libelium.com.

[14] www.digi.com.

[15] T. Sistemas, www.tst-sistemas.es/.

[16] TSgaTe, http://www.tst-sistemas.es/Docs/TSgaTe_v2-EN.pdf.

[17] TSmoTe, http://www.tst-sistemas.es/Docs/TSmoTe-EN.pdf.

[18] Libelium, Waspmote technical guide, 2016.

[19] Libelium, Prototyping sensor board technical guide, 2014.

[20] Libelium, Meshlium technical guide, 2014.

[21] www.campbellsci.es.

[22] Libelium, IDE User guide, 2014.

[23] Libelium, Programming guide, 2015.

[24] http://downloads.libelium.com/otap_0.1-b111.zip.

[25] X-CTU, https://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu.

[26] Digi, XCTU User guide, http://www.digi.com/resources/documentation/digidocs/90001458-13/default.htm.

[27] www.contiki-os.org.

[28] www.tinyos.net.

ANEXO A

En este anexo puede encontrarse el código implementado en los nodos que actúan de dispositivo final.

```
/* Implementacion del comportamiento de un dispositivo final */
/* Gloria Martinez Muñoz */

#include <WaspXBeeZB.h>
#include <WaspFrame.h>
#include <WaspSensorPrototyping_v20.h>
#include <WaspUtils.h>


// Direccion MAC Zigbee de Meshlium
char TX_ADDRESS[] = "0013A20040C320A3";

// ID del nodo
char WASPMOTE_ID[] = "NODO_L6";

// Definicion de variables
uint8_t errorTX;
uint8_t errorRTC;
float medida_ADC;
char valor_string[20];
int numeroDia;
float tiempo;
char conversion[11];
int horaDormir;
int minDormir;
bool horario;


void setup()
{
  // Inicializacion del puerto USB para depuracion
  USB.ON();
  USB.println( WASPMOTE_ID );

  // Almacenamiento del ID del nodo en la memoria EEPROM
  frame.setID( WASPMOTE_ID );

  // Inicializacion del modulo Zigbee
  xbeeZB.ON();

  // Inicializacion de la placa de prototipado
  SensorProtov20.ON();

  // Inicializacion del RTC
  RTC.ON();

  // Ajuste de fecha y hora
  // año:mes:dia:dia_semana:hora:min:seg (domingo = 1)
  RTC.setTime("16:11:19:02:13:54:00");

  // Comprobacion de los parametros de red
  comprobarParametrosRed();
  //Llamada a la funcion obtenerDiaAnio que calcula el dia del año
```

```
  // para obtener el tiempo y las horas a las que se debe dormir el nodo
  obtenerDiaAnio();
  delay(5000);
}



void loop()
{

  // sincronizacion del reloj con la hora de la pasarela
  errorRTC = xbeeZB.setRTCfromMeshlium(TX_ADDRESS);

  // // Comprobacion de bandera. Si la sincronizacion falla,
  // se mantiene la hora introducida a mano durante la carga del programa
  if( errorRTC == 0 )
  {
    USB.print(F("Sincronizacion OK "));
  }
  else
  {
    USB.print(F("Error en la sincronizacion de relojes. "));
  }

  USB.print(F("RTC Time:"));
  USB.println(RTC.getTime());
  delay(1000);


  // Lectura del convertidor para tomar una muestra de irradiancia
  medida_ADC = SensorProtov20.readADC();


  // Muestra de resultados por pantalla para depuracion
  USB.print(F("Valor: "));
  USB.print(medida_ADC);
  USB.println(F("V"));

  // Conversion de la muestra de Voltios a Watios
  medida_ADC = medida_ADC * 100000;
  USB.print(F("Valor: "));
  USB.print(medida_ADC);
  USB.println(F("W"));
  USB.println();

  USB.print(F("Hora de la muestra: "));
  USB.println(RTC.getTime());
  USB.print(F("Nivel de bateria: "));
  USB.print(PWR.getBatteryLevel(),DEC);
  USB.println();


  // Conversion del valor a cadena de caracteres para introducirla en la
trama a enviar
  Utils.float2String(medida_ADC, valor_string, 10);

  // Creacion de una nueva trama
  frame.createFrame(ASCII);

  // Adicion de campos de la trama: valor de la medida, nivel de bateria,
temperatura
  frame.addSensor(SENSOR_STR, valor_string);
```

```
  frame.addSensor(SENSOR_BAT, PWR.getBatteryLevel());
  frame.addSensor(SENSOR_IN_TEMP, RTC.getTemperature());

  // Mostrar por pantalla el formato de la trama
  frame.showFrame();

  //Envio del paquete con un maximo de tres reintentos en caso de fallo
  int i = 0;
  while (i < 3)
  {
    errorTX = xbeeZB.send( TX_ADDRESS, frame.buffer, frame.length );
    // Comprobar el envio correcto
    if( errorTX == 0 )
    {
      USB.println(F("Envio correcto"));
      // Parpadea el led verde
      Utils.blinkGreenLED();
      i = 3;
    }
    else
    {
      USB.println(F("Fallo de envio"));
      //Parpadea led rojo
      Utils.blinkRedLED();
      i = i + 1;
      delay(1000);
    }
  }

  //Comprobacion de si es la hora exacta del ocaso anteriormente
  // calculada para dormir al nodo
  RTC.getTime();
  if( RTC.hour == horaDormir )
  {
    if( RTC.minute >= minDormir )
    {
      // Dormir al nodo desconectando todos los switches y modulos hasta el
dia siguiente
      USB.println(F("Deep sleep hasta manana"));
      PWR.deepSleep(conversion,RTC_OFFSET,RTC_ALM1_MODE1,ALL_OFF);
      //Cuando se despierta, vuelven a calcularse las horas de orto
      //y ocaso y el tiempo dormido para el nuevo dia
      obtenerDiaAnio();
    }
    //Si no coinciden los minutos, se duerme al nodo 30 segundos
    else
    {
      // Dormir al nodo desconectando todos los switches y modulos durante 60
segundos
      USB.println(F("Deep sleep 30seg"));
      PWR.deepSleep("00:00:00:30",RTC_OFFSET,RTC_ALM1_MODE1,ALL_OFF);
    }
  }

  //Si la hora no coincide, se duerme al nodo durante 30 segundos
  else
  {
    // Dormir al nodo desconectando todos los switches y modulos durante 60
segundos
    USB.println(F("Deep sleep 30seg"));
    PWR.deepSleep("00:00:00:30",RTC_OFFSET,RTC_ALM1_MODE1,ALL_OFF);
  }
```

```
  USB.ON();
  USB.println(F("\nDespierto!"));
  // Captura de bandera e interrupcion
  if( intFlag & RTC_INT )
  {
    USB.println(F("Interrupcion capturada"));
    // Parpadean todos los leds
    Utils.blinkLEDs(300);
    Utils.blinkLEDs(300);
    Utils.blinkLEDs(300);
    intFlag &= ~(RTC_INT);
  }
  // Inicializar de nuevo los modulos
  xbeeZB.ON();
  SensorProtov20.ON();
  RTC.ON();

  delay(3000);

}

//Funcion que hace las comprobaciones necesarias para unir al nodo a una red
// y obtener los parametros de la misma
void comprobarParametrosRed()
{
  // Obtener operating 64-b PAN ID
  xbeeZB.getOperating64PAN();

  // Esperar indicacion de asociacion
  xbeeZB.getAssociationIndication();

  while( xbeeZB.associationIndication != 0 )
  {
    delay(2000);

    // Obtener operating 64-b PAN ID
    xbeeZB.getOperating64PAN();

    USB.print(F("Operating 64-b PAN ID: "));
    USB.printHex(xbeeZB.operating64PAN[0]);
    USB.printHex(xbeeZB.operating64PAN[1]);
    USB.printHex(xbeeZB.operating64PAN[2]);
    USB.printHex(xbeeZB.operating64PAN[3]);
    USB.printHex(xbeeZB.operating64PAN[4]);
    USB.printHex(xbeeZB.operating64PAN[5]);
    USB.printHex(xbeeZB.operating64PAN[6]);
    USB.printHex(xbeeZB.operating64PAN[7]);
    USB.println();

    xbeeZB.getAssociationIndication();
  }

  USB.println(F("\nUnido a la red!"));

  // Obtener los parametros de la red
  xbeeZB.getOperating16PAN();
  xbeeZB.getOperating64PAN();
  xbeeZB.getChannel();

  USB.print(F("operating 16-b PAN ID: "));
  USB.printHex(xbeeZB.operating16PAN[0]);
```

```
  USB.printHex(xbeeZB.operating16PAN[1]);
  USB.println();

  USB.print(F("operating 64-b PAN ID: "));
  USB.printHex(xbeeZB.operating64PAN[0]);
  USB.printHex(xbeeZB.operating64PAN[1]);
  USB.printHex(xbeeZB.operating64PAN[2]);
  USB.printHex(xbeeZB.operating64PAN[3]);
  USB.printHex(xbeeZB.operating64PAN[4]);
  USB.printHex(xbeeZB.operating64PAN[5]);
  USB.printHex(xbeeZB.operating64PAN[6]);
  USB.printHex(xbeeZB.operating64PAN[7]);
  USB.println();

  USB.print(F("Canal: "));
  USB.printHex(xbeeZB.channel);
  USB.println();

}

//Esta funcion calcula el numero del dia del año
//a partir de la fecha del RTC. No tiene en cuenta si el año
//es bisiesto puesto que la matriz de valores no lo contempla
//Tambien calcula si nos encontramos en horario de verano o invierno
void obtenerDiaAnio()
{
  int diasEnMeses[] = {
    31,28,31,30,31,30,31,31,30,31,30,31  };
  int dia;      //dia del mes
  int mes;      //mes
  int anio;     //anio
  int diasemana;        //dia de la semana
  int numeroDia=0;
  int restante; //dias restantes para que acabe el mes

  RTC.getTime();
  dia = RTC.date;
  mes = RTC.month;
  anio = RTC.year;
  diasemana = RTC.dow(anio,mes,dia);

  //Calculo del dia del anio
  for (int i = 0; i < mes - 1; i++) {
    numeroDia = numeroDia + diasEnMeses[i];
  }

  numeroDia = numeroDia + dia;

  USB.print("Numero dia: ");
  USB.println(numeroDia);


  //Calculo de horario de invierno o verano. Para ello se comprueba el
periodo en el que
  // se encuentra la fecha actual. El ultimo domingo de marzo se cambia a
horario de verano
  // mientras que el ultimo domingo de octubre, a horario de invierno
  if(mes>3 && mes<10)
  {
    //En los meses entre marzo y octubre, es horario de verano
    horario=true;
  }
```

```
  if(mes>10 || mes<3)
  {
    //En los meses entre octubre y marzo, es horario de invierno
    horario=false;
  }
  if (mes==3)
  {
    //En el mes de marzo, caculamos si se trata del ultimo domingo del mes
    restante=diasEnMeses[3] - dia;
    //Comprobamos si se trata de la ultima semana del mes
    if (dia+7 > diasEnMeses[3])
    {
      //Comprobamos si dentro de la ultima semana, ya es domingo algun dia
posterior a este
      if (7-diasemana >= restante)
      {
        horario=true; //Horario de verano
      }
      else
        horario= false;
    }
    else
      horario=false;
  }


  if (mes==10)
  {
    //En el mes de octubre, caculamos si se trata del ultimo domingo del mes
    restante= diasEnMeses[10] - dia;
    //Comprobamos si se trata de la ultima semana del mes
    if (dia+7 > diasEnMeses[10])
    {
      //Comprobamos si dentro de la ultima semana, ya es domingo algun dia
posterior a este
      if (7-diasemana >= restante)
      {
        horario=false; //horario de invierno
      }
      else
        horario=true;
    }
    else
      horario=true;
  }

  //Llamada a la funcion obtenerHoras
  obtenerHoras(numeroDia);
}



// Esta funcion obtiene las horas GSM a las cuales se procuden
//el orto y el ocaso para un numero de dia del año determinado
// Para ello se hace una busqueda en una matriz que contiene estos
//datos para cada dia del año, sin contemplar años bisiestos
// Por ultimo se calcula el tiempo que el nodo debe permanecer dormido
//calculando la diferencia entre ambos horarios y teniendo en cuenta el
horario de invierno o verano
void obtenerHoras(int numeroDia)
{
  float horaOcaso;
```

```
  float horaOrto;
  double matriz[365][3] =


{{7.713,17.185} ,
{7.715,17.198} ,
{7.716,17.211} ,
{7.717,17.225} ,
{7.718,17.239} ,
{7.718,17.253} ,
{7.717,17.268} ,
{7.716,17.283} ,
{7.714,17.299} ,
{7.712,17.314} ,
{7.709,17.330} ,
{7.705,17.347} ,
{7.702,17.363} ,
{7.697,17.380} ,
{7.692,17.397} ,
{7.687,17.414} ,
{7.681,17.432} ,
{7.674,17.450} ,
{7.667,17.467} ,
{7.659,17.485} ,
{7.651,17.504} ,
{7.642,17.522} ,
{7.633,17.540} ,
{7.624,17.559} ,
{7.614,17.578} ,
{7.603,17.597} ,
{7.592,17.616} ,
{7.581,17.634} ,
{7.569,17.653} ,
{7.556,17.673} ,
{7.543,17.692} ,
{7.530,17.711} ,
{7.516,17.730} ,
{7.502,17.749} ,
{7.487,17.768} ,
{7.472,17.787} ,
{7.457,17.806} ,
{7.441,17.825} ,
{7.425,17.844} ,
{7.408,17.863} ,
{7.391,17.882} ,
{7.374,17.901} ,
{7.356,17.920} ,
{7.338,17.939} ,
{7.319,17.957} ,
{7.301,17.976} ,
{7.282,17.994} ,
{7.262,18.013} ,
{7.242,18.031} ,
{7.222,18.049} ,
{7.202,18.067} ,
{7.181,18.085} ,
{7.160,18.103} ,
{7.139,18.121} ,
{7.118,18.139} ,
{7.096,18.156} ,
{7.074,18.173} ,
```

```
{7.052,18.191} ,
{7.030,18.208} ,
{7.007,18.225} ,
{6.984,18.242} ,
{6.961,18.259} ,
{6.938,18.275} ,
{6.914,18.292} ,
{6.891,18.308} ,
{6.867,18.325} ,
{6.843,18.341} ,
{6.819,18.357} ,
{6.794,18.373} ,
{6.770,18.389} ,
{6.745,18.405} ,
{6.721,18.420} ,
{6.696,18.436} ,
{6.671,18.452} ,
{6.646,18.467} ,
{6.621,18.482} ,
{6.596,18.498} ,
{6.571,18.513} ,
{6.546,18.528} ,
{6.520,18.543} ,
{6.495,18.558} ,
{6.470,18.573} ,
{6.444,18.588} ,
{6.419,18.603} ,
{6.394,18.617} ,
{6.368,18.632} ,
{6.343,18.647} ,
{6.318,18.662} ,
{6.292,18.676} ,
{6.267,18.691} ,
{6.242,18.705} ,
{6.217,18.720} ,
{6.192,18.734} ,
{6.167,18.749} ,
{6.142,18.763} ,
{6.117,18.778} ,
{6.093,18.792} ,
{6.068,18.807} ,
{6.044,18.821} ,
{6.020,18.836} ,
{5.996,18.850} ,
{5.972,18.865} ,
{5.948,18.879} ,
{5.924,18.894} ,
{5.901,18.908} ,
{5.878,18.923} ,
{5.855,18.937} ,
{5.832,18.952} ,
{5.809,18.967} ,
{5.787,18.981} ,
{5.765,18.996} ,
{5.743,19.010} ,
{5.721,19.025} ,
{5.700,19.040} ,
{5.679,19.054} ,
{5.658,19.069} ,
{5.637,19.084} ,
{5.617,19.098} ,
{5.597,19.113} ,
```

```
{5.578,19.128} ,
{5.558,19.142} ,
{5.539,19.157} ,
{5.521,19.172} ,
{5.502,19.186} ,
{5.484,19.201} ,
{5.467,19.216} ,
{5.449,19.230} ,
{5.433,19.245} ,
{5.416,19.259} ,
{5.400,19.274} ,
{5.384,19.288} ,
{5.369,19.302} ,
{5.354,19.317} ,
{5.339,19.331} ,
{5.325,19.345} ,
{5.312,19.359} ,
{5.298,19.373} ,
{5.286,19.386} ,
{5.273,19.400} ,
{5.261,19.413} ,
{5.250,19.427} ,
{5.239,19.440} ,
{5.228,19.453} ,
{5.218,19.466} ,
{5.208,19.478} ,
{5.199,19.491} ,
{5.191,19.503} ,
{5.182,19.515} ,
{5.175,19.527} ,
{5.167,19.539} ,
{5.161,19.550} ,
{5.154,19.561} ,
{5.149,19.572} ,
{5.143,19.582} ,
{5.139,19.593} ,
{5.134,19.603} ,
{5.130,19.612} ,
{5.127,19.622} ,
{5.124,19.631} ,
{5.122,19.639} ,
{5.120,19.648} ,
{5.119,19.656} ,
{5.118,19.663} ,
{5.118,19.671} ,
{5.118,19.677} ,
{5.118,19.684} ,
{5.119,19.690} ,
{5.121,19.696} ,
{5.123,19.701} ,
{5.125,19.706} ,
{5.128,19.710} ,
{5.132,19.714} ,
{5.135,19.717} ,
{5.140,19.721} ,
{5.144,19.723} ,
{5.149,19.725} ,
{5.155,19.727} ,
{5.161,19.728} ,
{5.167,19.729} ,
{5.174,19.729} ,
{5.181,19.729} ,
```

```
{5.189,19.728} ,
{5.196,19.727} ,
{5.205,19.725} ,
{5.213,19.723} ,
{5.222,19.721} ,
{5.231,19.717} ,
{5.241,19.714} ,
{5.251,19.710} ,
{5.261,19.705} ,
{5.271,19.700} ,
{5.282,19.694} ,
{5.293,19.688} ,
{5.304,19.682} ,
{5.315,19.675} ,
{5.327,19.667} ,
{5.339,19.659} ,
{5.351,19.651} ,
{5.363,19.642} ,
{5.376,19.632} ,
{5.389,19.622} ,
{5.401,19.612} ,
{5.414,19.601} ,
{5.427,19.590} ,
{5.441,19.578} ,
{5.454,19.566} ,
{5.468,19.553} ,
{5.481,19.540} ,
{5.495,19.526} ,
{5.509,19.512} ,
{5.522,19.498} ,
{5.536,19.483} ,
{5.550,19.468} ,
{5.564,19.453} ,
{5.578,19.437} ,
{5.592,19.420} ,
{5.607,19.403} ,
{5.621,19.386} ,
{5.635,19.369} ,
{5.649,19.351} ,
{5.663,19.333} ,
{5.677,19.314} ,
{5.692,19.295} ,
{5.706,19.276} ,
{5.720,19.256} ,
{5.734,19.236} ,
{5.748,19.216} ,
{5.762,19.195} ,
{5.776,19.175} ,
{5.790,19.153} ,
{5.804,19.132} ,
{5.818,19.110} ,
{5.832,19.088} ,
{5.846,19.066} ,
{5.860,19.044} ,
{5.874,19.021} ,
{5.888,18.998} ,
{5.901,18.975} ,
{5.915,18.951} ,
{5.929,18.928} ,
{5.942,18.904} ,
{5.956,18.880} ,
{5.969,18.856} ,
```

```
{5.983,18.831} ,
{5.996,18.807} ,
{6.010,18.782} ,
{6.023,18.757} ,
{6.037,18.732} ,
{6.050,18.707} ,
{6.063,18.682} ,
{6.077,18.656} ,
{6.090,18.631} ,
{6.104,18.605} ,
{6.117,18.580} ,
{6.130,18.554} ,
{6.144,18.528} ,
{6.157,18.502} ,
{6.170,18.476} ,
{6.184,18.450} ,
{6.197,18.424} ,
{6.211,18.398} ,
{6.224,18.372} ,
{6.238,18.346} ,
{6.251,18.320} ,
{6.265,18.294} ,
{6.279,18.268} ,
{6.292,18.242} ,
{6.306,18.216} ,
{6.320,18.190} ,
{6.334,18.164} ,
{6.348,18.139} ,
{6.362,18.113} ,
{6.376,18.087} ,
{6.390,18.062} ,
{6.405,18.037} ,
{6.419,18.011} ,
{6.434,17.986} ,
{6.448,17.961} ,
{6.463,17.936} ,
{6.478,17.911} ,
{6.493,17.887} ,
{6.508,17.863} ,
{6.523,17.838} ,
{6.538,17.814} ,
{6.554,17.790} ,
{6.569,17.767} ,
{6.585,17.743} ,
{6.601,17.720} ,
{6.617,17.697} ,
{6.633,17.674} ,
{6.649,17.652} ,
{6.665,17.630} ,
{6.682,17.608} ,
{6.698,17.586} ,
{6.715,17.565} ,
{6.731,17.544} ,
{6.748,17.523} ,
{6.765,17.502} ,
{6.782,17.482} ,
{6.799,17.462} ,
{6.817,17.443} ,
{6.834,17.424} ,
{6.852,17.405} ,
{6.869,17.386} ,
{6.887,17.368} ,
```

```
{6.905,17.350} ,
{6.922,17.333} ,
{6.940,17.316} ,
{6.958,17.300} ,
{6.976,17.283} ,
{6.994,17.268} ,
{7.012,17.252} ,
{7.030,17.238} ,
{7.048,17.223} ,
{7.066,17.209} ,
{7.084,17.196} ,
{7.103,17.182} ,
{7.121,17.170} ,
{7.139,17.158} ,
{7.156,17.146} ,
{7.174,17.135} ,
{7.192,17.124} ,
{7.210,17.114} ,
{7.228,17.104} ,
{7.245,17.095} ,
{7.263,17.086} ,
{7.280,17.078} ,
{7.297,17.071} ,
{7.314,17.064} ,
{7.331,17.057} ,
{7.347,17.051} ,
{7.364,17.045} ,
{7.380,17.041} ,
{7.396,17.036} ,
{7.412,17.032} ,
{7.427,17.029} ,
{7.442,17.026} ,
{7.457,17.024} ,
{7.472,17.023} ,
{7.486,17.022} ,
{7.500,17.021} ,
{7.514,17.021} ,
{7.527,17.022} ,
{7.540,17.023} ,
{7.553,17.025} ,
{7.565,17.027} ,
{7.577,17.030} ,
{7.588,17.034} ,
{7.599,17.038} ,
{7.610,17.042} ,
{7.620,17.047} ,
{7.629,17.053} ,
{7.639,17.059} ,
{7.647,17.066} ,
{7.656,17.073} ,
{7.663,17.081} ,
{7.671,17.089} ,
{7.677,17.098} ,
{7.684,17.107} ,
{7.689,17.117} ,
{7.695,17.127} ,
{7.699,17.138} ,
{7.704,17.149} ,
{7.707,17.161} ,
{7.710,17.173}};
```

```
  //Se ajustan los horarios a la franja horaria española
  //+1 hora en invierno
  //+2 horas en verano
  horaOrto = matriz[numeroDia][0];
  horaOcaso = matriz[numeroDia][1];

  if(horario==true)
  {
    horaOrto = horaOrto + 2;
    horaOcaso = horaOcaso + 2;
  }
  else
  {
    horaOrto = horaOrto + 1;
    horaOcaso = horaOcaso + 1;
  }

  USB.print("Hora orto: ");
  USB.println(horaOrto);
  USB.print("Hora ocaso: ");
  USB.println(horaOcaso);

  //Calculo del tiempo que el nodo debe permanecer dormido cuando llega la
hora del ocaso
  tiempo = 24 - horaOcaso + horaOrto;
  USB.print("Tiempo dormido: ");
  USB.println(tiempo);

  //Llamada a la funcion conversorHoras
  conversorHoras(tiempo, true);
  conversorHoras(horaOcaso, false);

}

//Esta funcion convierte las horas anteriormente calculadas al formato
adecuado
//para usarlas en la funcion de Waspmote e indicar al nodo que debe dormirse
//Segun el valor de la bandera, se utiliza el tiempo que debe permanecer
dormido,
// o bien la hora a la que debe irse a dormir
void conversorHoras(float tiempo, bool bandera)
{

  float segundos = 0;
  float minutos = 0;
  float horas = 0;

  char horachar[20];
  char minchar[20];


  segundos = tiempo*3600;
  horas = floor(tiempo);
  segundos = segundos - (horas*3600);
  minutos= floor(segundos/60);
  segundos = segundos - (minutos*60);

  // si la bandera vale true se realiza la conversion del tiempo que debe
permanecer dormido
  if (bandera==true)
```

```
  {
    Utils.float2String(horas, horachar, 1);
    Utils.float2String(minutos, minchar, 1);

    if (horas<10)
    {
      if(minutos<10)
        sprintf(conversion,"00:0%c:0%c:00",horachar[0],minchar[0]);

      else

sprintf(conversion,"00:0%c:%c%c:00",horachar[0],minchar[0],minchar[1]);
    }
    else
    {
      if(minutos<10)
        sprintf(conversion,"00:%c%c:0%c:00",horachar[0],
horachar[1],minchar[0]);

      else
        sprintf(conversion,"00:%c%c:%c%c:00",horachar[0],
horachar[1],minchar[0],minchar[1]);
    }

    USB.print("Conversion tiempo dormido: ");
    USB.print(conversion[0]);
    USB.print(conversion[1]);
    USB.print(conversion[2]);
    USB.print(conversion[3]);
    USB.print(conversion[4]);
    USB.print(conversion[5]);
    USB.print(conversion[6]);
    USB.print(conversion[7]);
    USB.print(conversion[8]);
    USB.print(conversion[9]);
    USB.print(conversion[10]);
    USB.println(conversion[11]);
  }

  //Si la bandera vale false se realiza la conversion de la hora a la que
debe dormirse
  if (bandera == false)
  {
    horaDormir = horas,DEC;
    minDormir = minutos,DEC;
    USB.print("HoraDormir: ");
    USB.println(horaDormir);
    USB.print("MinDormir: ");
    USB.println(minDormir);
  }


}
```

ANEXO B

En este anexo puede encontrarse el código implementado en los nodos que actúan de routers.

```
/* Implemetacion del comportamiento de un dispositivo enrutador */
/* Gloria Martinez Muñoz */

#include <WaspXBeeZB.h>
#include <WaspFrame.h>
#include <WaspSensorPrototyping_v20.h>
#include <WaspUtils.h>


// Direccion MAC Zigbee de Meshlium
char TX_ADDRESS[] = "0013A20040C320A3";

// ID del nodo
char WASPMOTE_ID[] = "NODO_P6";

// Definicion de variables
uint8_t errorTX;
uint8_t errorRX;
uint8_t errorRTC;
float medida_ADC;
char valor_string[20];
int numeroDia;
float tiempo;
char conversion[11];
int horaDormir;
int minDormir;
bool horario;


void setup()
{
  // Inicializacion del puerto USB para depuracion
  USB.ON();
  USB.println( WASPMOTE_ID );

  // Almacenamiento del ID del nodo en la memoria EEPROM
  frame.setID( WASPMOTE_ID );

  // Inicializacion del modulo Zigbee
  xbeeZB.ON();

  // Inicializacion de la placa de prototipado
  SensorProtov20.ON();

  // Inicializacion del RTC
  RTC.ON();

  // Ajuste de fecha y hora
  // año:mes:dia:dia_semana:hora:min:seg (domingo = 1)
  RTC.setTime("16:11:29:02:12:49 :00");

  // Comprobacion de los parametros de red
  comprobarParametrosRed();
```

```
  //Llamada a la funcion obtenerDiaAnio que calcula el dia del año
  // para obtener el tiempo y las horas a las que se debe dormir el nodo
  obtenerDiaAnio();
  delay(5000);
}



void loop()
{

  // Sincronizacion del reloj con la hora de la pasarela
  errorRTC = xbeeZB.setRTCfromMeshlium(TX_ADDRESS);

  // Comprobacion de bandera. Si la sincronizacion falla,
  // se mantiene la hora introducida a mano durante la carga del programa
  if( errorRTC == 0 )
  {
    USB.print(F("Sincronizacion OK "));
  }
  else
  {
    USB.print(F("Error en la sincronizacion de relojes. "));
  }

  USB.print(F("RTC Time:"));
  USB.println(RTC.getTime());
  delay(1000);


  // Lectura del convertidor para tomar una muestra de irradiancia
  medida_ADC = SensorProtov20.readADC();


  // Muestra de resultados por pantalla para depuracion
  USB.print(F("Valor: "));
  USB.print(medida_ADC);
  USB.println(F("V"));

  // Conversion de la muestra de Voltios a Watios
  medida_ADC = medida_ADC * 100000;
  USB.print(F("Valor: "));
  USB.print(medida_ADC);
  USB.println(F("W"));
  USB.println();

  USB.print(F("Hora de la muestra: "));
  USB.println(RTC.getTime());
  USB.print(F("Nivel de bateria: "));
  USB.println(RTC.getBatteryLevel());
  USB.println();

  // Conversion del valor a cadena de caracteres para introducirla en la
trama a enviar
  Utils.float2String(medida_ADC, valor_string, 10);

  // Creacion de una nueva trama
  frame.createFrame(ASCII);

  // Adicion de campos de la trama: valor de la medida, nivel de bateria,
temperatura
```

```
  frame.addSensor(SENSOR_STR, valor_string);
  frame.addSensor(SENSOR_BAT, PWR.getBatteryLevel());
  frame.addSensor(SENSOR_IN_TEMP, RTC.getTemperature());

  // Mostrar por pantalla el formato de la trama
  frame.showFrame();

  //Envio del paquete con un maximo de tres reintentos en caso de fallo
  int i = 0;
  while (i < 3)
  {
    errorTX = xbeeZB.send( TX_ADDRESS, frame.buffer, frame.length );
    // Comprobar el envio correcto
    if( errorTX == 0 )
    {
      USB.println(F("Envio correcto"));
      // Parpadea el led verde
      Utils.blinkGreenLED();
      i = 3;
    }
    else
    {
      USB.println(F("Fallo de envio"));
      //Parpadea led rojo
      Utils.blinkRedLED();
      i = i + 1;
      delay(1000);
    }
  }

  // Esperar trama de otros nodos durante 25 segundos para reenviarlas
  errorRX = xbeeZB.receivePacketTimeout( 25000 );

  //Comprobacion de si es la hora exacta del ocaso anteriormente
  // calculada para dormir al nodo
  RTC.getTime();
  if( RTC.hour == horaDormir )
  {
    if( RTC.minute >= minDormir )
    {
      // Dormir al nodo desconectando todos los switches y modulos hasta el
dia siguiente
      USB.println(F("Deep sleep hasta manana"));
      PWR.deepSleep(conversion,RTC_OFFSET,RTC_ALM1_MODE1,ALL_OFF);
      // Captura de bandera e interrupcion
      if( intFlag & RTC_INT )
      {
        USB.println(F("Interrupcion capturada"));
        // Parpadean todos los leds
        Utils.blinkLEDs(300);
        Utils.blinkLEDs(300);
        Utils.blinkLEDs(300);
        intFlag &= ~(RTC_INT);
      }
      // Inicializar de nuevo los modulos
      xbeeZB.ON();
      SensorProtov20.ON();
      RTC.ON();
      //Cuando se despierta, vuelven a calcularse las horas de orto
      //y ocaso y el tiempo dormido para el nuevo dia
      obtenerDiaAnio();
    }
```

```
  }

  delay(3000);

}

//Funcion que hace las comprobaciones necesarias para unir al nodo a una red
// y obtener los parametros de la misma
void comprobarParametrosRed()
{
  // Obtener operating 64-b PAN ID
  xbeeZB.getOperating64PAN();

  // Esperar indicacion de asociacion
  xbeeZB.getAssociationIndication();

  while( xbeeZB.associationIndication != 0 )
  {
    delay(2000);

    // Obtener operating 64-b PAN ID
    xbeeZB.getOperating64PAN();

    USB.print(F("Operating 64-b PAN ID: "));
    USB.printHex(xbeeZB.operating64PAN[0]);
    USB.printHex(xbeeZB.operating64PAN[1]);
    USB.printHex(xbeeZB.operating64PAN[2]);
    USB.printHex(xbeeZB.operating64PAN[3]);
    USB.printHex(xbeeZB.operating64PAN[4]);
    USB.printHex(xbeeZB.operating64PAN[5]);
    USB.printHex(xbeeZB.operating64PAN[6]);
    USB.printHex(xbeeZB.operating64PAN[7]);
    USB.println();

    xbeeZB.getAssociationIndication();
  }

  USB.println(F("\nUnido a la red!"));

  // Obtener los parametros de la red
  xbeeZB.getOperating16PAN();
  xbeeZB.getOperating64PAN();
  xbeeZB.getChannel();

  USB.print(F("operating 16-b PAN ID: "));
  USB.printHex(xbeeZB.operating16PAN[0]);
  USB.printHex(xbeeZB.operating16PAN[1]);
  USB.println();

  USB.print(F("operating 64-b PAN ID: "));
  USB.printHex(xbeeZB.operating64PAN[0]);
  USB.printHex(xbeeZB.operating64PAN[1]);
  USB.printHex(xbeeZB.operating64PAN[2]);
  USB.printHex(xbeeZB.operating64PAN[3]);
  USB.printHex(xbeeZB.operating64PAN[4]);
  USB.printHex(xbeeZB.operating64PAN[5]);
  USB.printHex(xbeeZB.operating64PAN[6]);
  USB.printHex(xbeeZB.operating64PAN[7]);
  USB.println();

  USB.print(F("Canal: "));
  USB.printHex(xbeeZB.channel);
```

```
  USB.println();

}

//Esta funcion calcula el numero del dia del año
//a partir de la fecha del RTC. No tiene en cuenta si el año
//es bisiesto puesto que la matriz de valores no lo contempla
//Tambien calcula si nos encontramos en horario de verano o invierno
void obtenerDiaAnio()
{
  int diasEnMeses[] = {
    31,28,31,30,31,30,31,31,30,31,30,31  };
  int dia;      //dia del mes
  int mes;      //mes
  int anio;     //anio
  int diasemana;        //dia de la semana
  int numeroDia=0;
  int restante; //dias restantes para que acabe el mes

  RTC.getTime();
  dia = RTC.date;
  mes = RTC.month;
  anio = RTC.year;
  diasemana = RTC.dow(anio,mes,dia);

  //Calculo del dia del anio
  for (int i = 0; i < mes - 1; i++) {
    numeroDia = numeroDia + diasEnMeses[i];
  }

  numeroDia = numeroDia + dia;

  USB.print("Numero dia: ");
  USB.println(numeroDia);


  //Calculo de horario de invierno o verano. Para ello se comprueba el
periodo en el que
  // se encuentra la fecha actual. El ultimo domingo de marzo se cambia a
horario de verano
  // mientras que el ultimo domingo de octubre, a horario de invierno
  if(mes>3 && mes<10)
  {
    //En los meses entre marzo y octubre, es horario de verano
    horario=true;
  }
  if(mes>10 || mes<3)
  {
    //En los meses entre octubre y marzo, es horario de invierno
    horario=false;
  }
  if (mes==3)
  {
    //En el mes de marzo, caculamos si se trata del ultimo domingo del mes
    restante=diasEnMeses[3] - dia;
    //Comprobamos si se trata de la ultima semana del mes
    if (dia+7 > diasEnMeses[3])
    {
      //Comprobamos si dentro de la ultima semana, ya es domingo algun dia
posterior a este
      if (7-diasemana >= restante)
      {
```

```
        horario=true; //Horario de verano
      }
      else
        horario= false;
    }
    else
      horario=false;
  }


  if (mes==10)
  {
    //En el mes de octubre, caculamos si se trata del ultimo domingo del mes
    restante= diasEnMeses[10] - dia;
    //Comprobamos si se trata de la ultima semana del mes
    if (dia+7 > diasEnMeses[10])
    {
      //Comprobamos si dentro de la ultima semana, ya es domingo algun dia
posterior a este
      if (7-diasemana >= restante)
      {
        horario=false; //horario de invierno
      }
      else
        horario=true;
    }
    else
      horario=true;
  }

  //Llamada a la funcion obtenerHoras
  obtenerHoras(numeroDia);
}



// Esta funcion obtiene las horas GSM a las cuales se procuden
//el orto y el ocaso para un numero de dia del año determinado
// Para ello se hace una busqueda en una matriz que contiene estos
//datos para cada dia del año, sin contemplar años bisiestos
// Por ultimo se calcula el tiempo que el nodo debe permanecer dormido
//calculando la diferencia entre ambos horarios y teniendo en cuenta el
horario de invierno o verano
void obtenerHoras(int numeroDia)
{
  float horaOcaso;
  float horaOrto;
  double matriz[365][3] =

{{7.713,17.185} ,
{7.715,17.198} ,
{7.716,17.211} ,
{7.717,17.225} ,
{7.718,17.239} ,
{7.718,17.253} ,
{7.717,17.268} ,
{7.716,17.283} ,
{7.714,17.299} ,
{7.712,17.314} ,
{7.709,17.330} ,
{7.705,17.347} ,
{7.702,17.363} ,
```

```
{7.697,17.380} ,
{7.692,17.397} ,
{7.687,17.414} ,
{7.681,17.432} ,
{7.674,17.450} ,
{7.667,17.467} ,
{7.659,17.485} ,
{7.651,17.504} ,
{7.642,17.522} ,
{7.633,17.540} ,
{7.624,17.559} ,
{7.614,17.578} ,
{7.603,17.597} ,
{7.592,17.616} ,
{7.581,17.634} ,
{7.569,17.653} ,
{7.556,17.673} ,
{7.543,17.692} ,
{7.530,17.711} ,
{7.516,17.730} ,
{7.502,17.749} ,
{7.487,17.768} ,
{7.472,17.787} ,
{7.457,17.806} ,
{7.441,17.825} ,
{7.425,17.844} ,
{7.408,17.863} ,
{7.391,17.882} ,
{7.374,17.901} ,
{7.356,17.920} ,
{7.338,17.939} ,
{7.319,17.957} ,
{7.301,17.976} ,
{7.282,17.994} ,
{7.262,18.013} ,
{7.242,18.031} ,
{7.222,18.049} ,
{7.202,18.067} ,
{7.181,18.085} ,
{7.160,18.103} ,
{7.139,18.121} ,
{7.118,18.139} ,
{7.096,18.156} ,
{7.074,18.173} ,
{7.052,18.191} ,
{7.030,18.208} ,
{7.007,18.225} ,
{6.984,18.242} ,
{6.961,18.259} ,
{6.938,18.275} ,
{6.914,18.292} ,
{6.891,18.308} ,
{6.867,18.325} ,
{6.843,18.341} ,
{6.819,18.357} ,
{6.794,18.373} ,
{6.770,18.389} ,
{6.745,18.405} ,
{6.721,18.420} ,
{6.696,18.436} ,
{6.671,18.452} ,
{6.646,18.467} ,
```

```
{6.621,18.482} ,
{6.596,18.498} ,
{6.571,18.513} ,
{6.546,18.528} ,
{6.520,18.543} ,
{6.495,18.558} ,
{6.470,18.573} ,
{6.444,18.588} ,
{6.419,18.603} ,
{6.394,18.617} ,
{6.368,18.632} ,
{6.343,18.647} ,
{6.318,18.662} ,
{6.292,18.676} ,
{6.267,18.691} ,
{6.242,18.705} ,
{6.217,18.720} ,
{6.192,18.734} ,
{6.167,18.749} ,
{6.142,18.763} ,
{6.117,18.778} ,
{6.093,18.792} ,
{6.068,18.807} ,
{6.044,18.821} ,
{6.020,18.836} ,
{5.996,18.850} ,
{5.972,18.865} ,
{5.948,18.879} ,
{5.924,18.894} ,
{5.901,18.908} ,
{5.878,18.923} ,
{5.855,18.937} ,
{5.832,18.952} ,
{5.809,18.967} ,
{5.787,18.981} ,
{5.765,18.996} ,
{5.743,19.010} ,
{5.721,19.025} ,
{5.700,19.040} ,
{5.679,19.054} ,
{5.658,19.069} ,
{5.637,19.084} ,
{5.617,19.098} ,
{5.597,19.113} ,
{5.578,19.128} ,
{5.558,19.142} ,
{5.539,19.157} ,
{5.521,19.172} ,
{5.502,19.186} ,
{5.484,19.201} ,
{5.467,19.216} ,
{5.449,19.230} ,
{5.433,19.245} ,
{5.416,19.259} ,
{5.400,19.274} ,
{5.384,19.288} ,
{5.369,19.302} ,
{5.354,19.317} ,
{5.339,19.331} ,
{5.325,19.345} ,
{5.312,19.359} ,
{5.298,19.373} ,
```

```
{5.286,19.386} ,
{5.273,19.400} ,
{5.261,19.413} ,
{5.250,19.427} ,
{5.239,19.440} ,
{5.228,19.453} ,
{5.218,19.466} ,
{5.208,19.478} ,
{5.199,19.491} ,
{5.191,19.503} ,
{5.182,19.515} ,
{5.175,19.527} ,
{5.167,19.539} ,
{5.161,19.550} ,
{5.154,19.561} ,
{5.149,19.572} ,
{5.143,19.582} ,
{5.139,19.593} ,
{5.134,19.603} ,
{5.130,19.612} ,
{5.127,19.622} ,
{5.124,19.631} ,
{5.122,19.639} ,
{5.120,19.648} ,
{5.119,19.656} ,
{5.118,19.663} ,
{5.118,19.671} ,
{5.118,19.677} ,
{5.118,19.684} ,
{5.119,19.690} ,
{5.121,19.696} ,
{5.123,19.701} ,
{5.125,19.706} ,
{5.128,19.710} ,
{5.132,19.714} ,
{5.135,19.717} ,
{5.140,19.721} ,
{5.144,19.723} ,
{5.149,19.725} ,
{5.155,19.727} ,
{5.161,19.728} ,
{5.167,19.729} ,
{5.174,19.729} ,
{5.181,19.729} ,
{5.189,19.728} ,
{5.196,19.727} ,
{5.205,19.725} ,
{5.213,19.723} ,
{5.222,19.721} ,
{5.231,19.717} ,
{5.241,19.714} ,
{5.251,19.710} ,
{5.261,19.705} ,
{5.271,19.700} ,
{5.282,19.694} ,
{5.293,19.688} ,
{5.304,19.682} ,
{5.315,19.675} ,
{5.327,19.667} ,
{5.339,19.659} ,
{5.351,19.651} ,
{5.363,19.642} ,
```

```
{5.376,19.632} ,
{5.389,19.622} ,
{5.401,19.612} ,
{5.414,19.601} ,
{5.427,19.590} ,
{5.441,19.578} ,
{5.454,19.566} ,
{5.468,19.553} ,
{5.481,19.540} ,
{5.495,19.526} ,
{5.509,19.512} ,
{5.522,19.498} ,
{5.536,19.483} ,
{5.550,19.468} ,
{5.564,19.453} ,
{5.578,19.437} ,
{5.592,19.420} ,
{5.607,19.403} ,
{5.621,19.386} ,
{5.635,19.369} ,
{5.649,19.351} ,
{5.663,19.333} ,
{5.677,19.314} ,
{5.692,19.295} ,
{5.706,19.276} ,
{5.720,19.256} ,
{5.734,19.236} ,
{5.748,19.216} ,
{5.762,19.195} ,
{5.776,19.175} ,
{5.790,19.153} ,
{5.804,19.132} ,
{5.818,19.110} ,
{5.832,19.088} ,
{5.846,19.066} ,
{5.860,19.044} ,
{5.874,19.021} ,
{5.888,18.998} ,
{5.901,18.975} ,
{5.915,18.951} ,
{5.929,18.928} ,
{5.942,18.904} ,
{5.956,18.880} ,
{5.969,18.856} ,
{5.983,18.831} ,
{5.996,18.807} ,
{6.010,18.782} ,
{6.023,18.757} ,
{6.037,18.732} ,
{6.050,18.707} ,
{6.063,18.682} ,
{6.077,18.656} ,
{6.090,18.631} ,
{6.104,18.605} ,
{6.117,18.580} ,
{6.130,18.554} ,
{6.144,18.528} ,
{6.157,18.502} ,
{6.170,18.476} ,
{6.184,18.450} ,
{6.197,18.424} ,
{6.211,18.398} ,
```

```
{6.224,18.372} ,
{6.238,18.346} ,
{6.251,18.320} ,
{6.265,18.294} ,
{6.279,18.268} ,
{6.292,18.242} ,
{6.306,18.216} ,
{6.320,18.190} ,
{6.334,18.164} ,
{6.348,18.139} ,
{6.362,18.113} ,
{6.376,18.087} ,
{6.390,18.062} ,
{6.405,18.037} ,
{6.419,18.011} ,
{6.434,17.986} ,
{6.448,17.961} ,
{6.463,17.936} ,
{6.478,17.911} ,
{6.493,17.887} ,
{6.508,17.863} ,
{6.523,17.838} ,
{6.538,17.814} ,
{6.554,17.790} ,
{6.569,17.767} ,
{6.585,17.743} ,
{6.601,17.720} ,
{6.617,17.697} ,
{6.633,17.674} ,
{6.649,17.652} ,
{6.665,17.630} ,
{6.682,17.608} ,
{6.698,17.586} ,
{6.715,17.565} ,
{6.731,17.544} ,
{6.748,17.523} ,
{6.765,17.502} ,
{6.782,17.482} ,
{6.799,17.462} ,
{6.817,17.443} ,
{6.834,17.424} ,
{6.852,17.405} ,
{6.869,17.386} ,
{6.887,17.368} ,
{6.905,17.350} ,
{6.922,17.333} ,
{6.940,17.316} ,
{6.958,17.300} ,
{6.976,17.283} ,
{6.994,17.268} ,
{7.012,17.252} ,
{7.030,17.238} ,
{7.048,17.223} ,
{7.066,17.209} ,
{7.084,17.196} ,
{7.103,17.182} ,
{7.121,17.170} ,
{7.139,17.158} ,
{7.156,17.146} ,
{7.174,17.135} ,
{7.192,17.124} ,
{7.210,17.114} ,
```

```
{7.228,17.104} ,
{7.245,17.095} ,
{7.263,17.086} ,
{7.280,17.078} ,
{7.297,17.071} ,
{7.314,17.064} ,
{7.331,17.057} ,
{7.347,17.051} ,
{7.364,17.045} ,
{7.380,17.041} ,
{7.396,17.036} ,
{7.412,17.032} ,
{7.427,17.029} ,
{7.442,17.026} ,
{7.457,17.024} ,
{7.472,17.023} ,
{7.486,17.022} ,
{7.500,17.021} ,
{7.514,17.021} ,
{7.527,17.022} ,
{7.540,17.023} ,
{7.553,17.025} ,
{7.565,17.027} ,
{7.577,17.030} ,
{7.588,17.034} ,
{7.599,17.038} ,
{7.610,17.042} ,
{7.620,17.047} ,
{7.629,17.053} ,
{7.639,17.059} ,
{7.647,17.066} ,
{7.656,17.073} ,
{7.663,17.081} ,
{7.671,17.089} ,
{7.677,17.098} ,
{7.684,17.107} ,
{7.689,17.117} ,
{7.695,17.127} ,
{7.699,17.138} ,
{7.704,17.149} ,
{7.707,17.161} ,
{7.710,17.173}};


  //Se ajustan los horarios a la franja horaria española
  //+1 hora en invierno
  //+2 horas en verano
  horaOrto = matriz[numeroDia][0];
  horaOcaso = matriz[numeroDia][1];

  if(horario==true)
  {
    horaOrto = horaOrto + 2;
    horaOcaso = horaOcaso + 2;
  }
  else
  {
    horaOrto = horaOrto + 1;
    horaOcaso = horaOcaso + 1;
  }

  USB.print("Hora orto: ");
```

```
  USB.println(horaOrto);
  USB.print("Hora ocaso: ");
  USB.println(horaOcaso);

  //Calculo del tiempo que el nodo debe permanecer dormido cuando llega la
hora del ocaso
  tiempo = 24 - horaOcaso + horaOrto;
  USB.print("Tiempo dormido: ");
  USB.println(tiempo);

  //Llamada a la funcion conversorHoras
  conversorHoras(tiempo, true);
  conversorHoras(horaOcaso, false);

}

//Esta funcion convierte las horas anteriormente calculadas al formato
adecuado
//para usarlas en la funcion de Waspmote e indicar al nodo que debe dormirse
//Segun el valor de la bandera, se utiliza el tiempo que debe permanecer
dormido,
// o bien la hora a la que debe irse a dormir
void conversorHoras(float tiempo, bool bandera)
{

  float segundos = 0;
  float minutos = 0;
  float horas = 0;

  char horachar[20];
  char minchar[20];

  segundos = tiempo*3600;
  horas = floor(tiempo);
  segundos = segundos - (horas*3600);
  minutos= floor(segundos/60);
  segundos = segundos - (minutos*60);

  // si la bandera vale true se realiza la conversion del tiempo que debe
permanecer dormido
  if (bandera==true)
  {
    Utils.float2String(horas, horachar, 1);
    Utils.float2String(minutos, minchar, 1);

    if (horas<10)
    {
      if(minutos<10)
        sprintf(conversion,"00:0%c:0%c:00",horachar[0],minchar[0]);

      else

sprintf(conversion,"00:0%c:%c%c:00",horachar[0],minchar[0],minchar[1]);
    }
    else
    {
      if(minutos<10)
        sprintf(conversion,"00:%c%c:0%c:00",horachar[0],
horachar[1],minchar[0]);

      else
```

```
        sprintf(conversion,"00:%c%c:%c%c:00",horachar[0],
horachar[1],minchar[0],minchar[1]);
    }

    USB.print("Conversion tiempo dormido: ");
    USB.print(conversion[0]);
    USB.print(conversion[1]);
    USB.print(conversion[2]);
    USB.print(conversion[3]);
    USB.print(conversion[4]);
    USB.print(conversion[5]);
    USB.print(conversion[6]);
    USB.print(conversion[7]);
    USB.print(conversion[8]);
    USB.print(conversion[9]);
    USB.print(conversion[10]);
    USB.println(conversion[11]);
  }

  //Si la bandera vale false se realiza la conversion de la hora a la que
debe dormirse
  if (bandera == false)
  {
    horaDormir = horas,DEC;
    minDormir = minutos,DEC;
    USB.print("HoraDormir: ");
    USB.println(horaDormir);
    USB.print("MinDormir: ");
    USB.println(minDormir);
  }


}
```

ANEXO C

En este anexo se encuentra el código correspondiente a la aplicación que se encuentra ejecutándose constantemente y se encarga de volcar los datos almacenados en la base de datos de la pasarela a un fichero de texto.

```
#include <mysql/mysql.h>
#include <stdio.h>


int main(int argc, char **argv) {


  MYSQL       * conn;                              // Variable para realizar la
conexion mysql
  MYSQL_RES   * result;                          // Variable que contiene el
valor de cada peticion
  MYSQL_ROW     columns;                      // Variable que contiene todos los
campos de cada registro de la consulta

  char        * server   = "172.16.2.154";       // Servidor IP
  char        * user     = "root";                        // Nombre de usuario
para el acceso a la bbdd
  char        * password = "libelium2007";     // Contraseña
  char        * database = "MeshliumDB";      // Nombre de la base de datos

  FILE        * fich;                           // Manejador para el fichero de
entrada


  /* Inicializacion de la conexion */
  conn = mysql_init(NULL);

  /* Conexion a la base de datos */
  if (!mysql_real_connect(conn, server, user, password, database, 0, NULL,
0))
    {
      fprintf(stderr, "%s\n", mysql_error(conn));
      exit(1);
    }

  else
    {
      printf("\nBienvenido al programa de consultas de Libelium\n\n");

      /* Comprobacion de los parametros de entrada */
      if (argc != 2)
    {
      fprintf(stderr, "Debe indicar el nombre del fichero para escribir los
datos.\n");
    }

      else
    {

      /* Si la apertura del fichero falla, el programa termina */
```

```
      if (NULL == (fich = fopen(argv[1], "w")))
        {
          fprintf(stderr, "No se ha podido abrir %s para su escritura.\n",
argv[1]);
        }

      else
        {

          printf("\tEscribiendo datos en el fichero...\n");

          /* Bucle infinito que mantiene al programa ejectutandose
continuamente */
          while(1)
        {

          /* Consulta de todos los datos de la base de datos */
              mysql_query(conn, "SELECT * FROM sensorParser");

          /* Almacenamiento de datos */
          result = mysql_store_result(conn);

          /* Se pone la position del flujo al principio del fichero para
sobreescribir todos
          datos antiguos en cada iteraccion. No puede realizarse de otra
forma porque el fichero
          permanece abierto */
          fseek(fich,0,SEEK_SET);

          /* Impresion de la primera linea del fichero generado */
          fprintf(fich, "ID\tID Wasp\t\tFrame
number\tSensor\t\tValue\t\t\tFecha\n\n");

          /* Impresion de los datos relevantes almacenados en la base de
datos */
          while((columns = mysql_fetch_row(result)))
            {
              fprintf(fich, "%s\t%s\t\t%s\t\t%s\t\t%s\t\t\t%s\n",
                  columns[0], columns[1], columns[4], columns[5], columns[6],
columns[7]);
            }
        }
        }

      /* Finalizacion de la conexion */
      mysql_close(conn);
      mysql_library_end();
      mysql_free_result(result);
      fclose(fich);
    }
    }

  return 0;
}
```