Departamento de Ingeniería Telemática

PROYECTO FIN DE CARRERA

INTERCONEXIÓN DE REDES DE SENSORES INALÁMBRICAS Y APLICACIONES EXTERNAS USANDO EL LENGUAJE SENCOMLNG

Autora: Iria Dopico Fernández

Tutor: Ángel Cuevas Rumín

Leganés, marzo de 2011

Agradecimientos

A todos aquellos que durante estos años han sabido estar ahí,

y especialmente, a mi madre.

Resumen

En este proyecto se pretende plantear e implementar un modelo para intercomunicar una red de sensores inalámbrica con una aplicación externa. Para ello se desarrollará una aplicación extremo a extremo de un sistema de comunicaciones para la solicitud de información de dispositivos inalámbricos.

Las redes inalámbricas de sensores no están diseñadas para soportar grandes cargas de información, lo que supone un problema a la hora de desarrollar aplicaciones para este tipo de redes.

Mediante la solución planteada en *eXtensible Binary Enconding (XBE32)* [1] y LWESP: *Light-Weight Exterior Sensornet Protocol* [2]*, se* desarrollará una aplicación que solvente estos problemas, pudiendo utilizar el potencial de XML para realizar peticiones sin que esto afecte a la hora de realizar peticiones en la red de sensores final.

Esta implementación se realizará mediante una aplicación Java para realizar peticiones en la red externa y la cual llevará un registro de las respuestas recibidas. Utilizará Internet para comunicarse con un equipo directamente conectado al punto de enlace con la red inalámbrica IEEE 802.15.4 sobre la que se realizarán las peticiones.
El punto de enlace será el dispositivo que crea la red y estará formado por una placa con un microprocesador, un transceptor y una serie de sensores, entre otros elementos. Mediante la programación de este, junto con el resto de dispositivos inalámbricos, la comunicación extremo a extremo será posible.

Palabras clave: Protocolos de comunicación, XBE32, SENCOMLNG, LWESP, IEEE 802.15.4, redes inalámbricas de sensores.

Abstract

The goal of this project is the implementation of a mechanism to intercommunicate a Wireless Sensor Network (WSN) and an external application. With this objective, an end-to-end application for a communication system will be developed. This application will request information to the wireless devices.

WSNs are not conceived to carry large amounts of data. This entails a problem when designing applications for this kind of networks.

Through the proposals described in *eXtensible Binary Enconding (XBE32)* [1] and LWESP: *Light-Weight Exterior Sensornet Protocol* [2], an application that solves this problem will be developed. In this way, the power of XML could be used to make requests to a wireless sensor network.

A Java application will make the request in the external network and will carry a register of the received replies. Internet will be used to link the external application with a computer directly connected to the access point to the final IEEE 802.15.4.
The access point is the wireless network coordinator and it is a device with a microcontroller, a transceptor and a series of embedded sensors. Programming this device, along with the other wireless devices, will make the end-to-end communication possible.

Palabras clave: Communication protocolos, SENCOMLNG, LWESP, IEEE 802.15.4, wireless sensor networks.

Contenido

Índice de tablas y figuras

Tablas

Figuras

1 Introducción

En este apartado se detallan la motivación para la realización de este proyecto y los objetivos que se pretenden alcanzar con su desarrollo.
También se resume el contenido de cada una de las secciones en las que está dividida esta memoria.

1.1 Motivación

Las redes inalámbricas de sensores serán fundamentales en un futuro más bien próximo dado que tienen multitud de utilidades: pueden utilizarse en entornos industriales para monitorizar procesos, en entornos agrícolas para controlar riegos (ver capítulo 3), en entornos hospitalarios [31] para monitorizar pacientes (temperatura, administración de alimentos por vía intravenosa, etc.) o incluso en entornos domésticos[4] para centralizar el control de los electrodomésticos y ahorrar energía. Todo esto es posible utilizando además tecnologías sencillas y de bajo coste basadas en IEEE 802.15.4 [3].
Si estas redes pudiesen administrarse a distancia a través de aplicaciones externas, el potencial de las tecnologías inalámbricas de sensores aumentaría aún más.
Por ejemplo, para grandes plantaciones agrícolas se podría optimizar el control de riego consultando parámetros como la humedad del suelo, la luz ambiental y la temperatura, ahorrando así agua, disminuyendo el coste por producto y optimizando el cultivo.

El objetivo de este proyecto es, por tanto, comunicar redes inalámbricas de sensores con aplicaciones externas y comprobar que es posible interaccionar con dichas redes de una forma telemática.

Para ello, se utilizará XML dado que es mayoritario en el desarrollo de aplicaciones puesto que permite codificar cualquier tipo de información de una forma muy flexible.

Al ser XML demasiado genérico, se crea SENCOMLNG [2] que es un lenguaje XML orientado a redes de sensores y define campos como identificador de mensaje, identificador de sensor y otros parámetros presentes en este tipo de comunicaciones.

El problema de XML es que es una codificación demasiado pesada para trabajar con ella dentro de redes de sensores puesto que en estas redes el espacio para datos es reducido. Además, estas redes pueden generar gran cantidad de trafico cuando se estén realizando operaciones de monitorización y si todo este tráfico fuese XML se podría causar saturación en la red.
Para evitar estos problemas de tamaño de codificación SENCOMLNG se puede codificar mediante XBE32, lo que permite reducir el ancho de banda hasta en un 45% en peticiones y en un 35% en respuestas [1]. Esta codificación consiste en otorgar una representación hexadecimal a las etiquetas definidas por SENCOMLNG.

XBE32 es la codificación que se utiliza en la transferencia de datos entre la aplicación que generará las peticiones y el Gateway de la red de sensores.
Una vez transformado el XML a XBE32, se envía la petición por la red hasta dicho Gateway que a su vez se comunicará con la red inalámbrica de sensores para realizar peticiones.
Para conseguir comunicación dentro de la red inalámbrica será necesario conocer el funcionamiento de los dispositivos inalámbricos e implementar un protocolo que reconozca el formato de las peticiones enviadas.

Para ello, se genera una aplicación externa que realice peticiones a la red de sensores. Estas peticiones inicialmente se codifican en SENCOMLNG, para después pasarlas a XBE32 y mandarlas por la red hacia la red de sensores.
En la parte de la red de sensores, el Gateway recibe la petición, extrae los datos codificados en XBE32 y los transforma a un protocolo interno, iLWESP, establecido para comunicar el Gateway y el nodo coordinador de la red.
El nodo coordinador de la red, una vez reciba la petición, la tramita y la envía al nodo final al que vaya dirigida que realizará la acción solicitada y enviará una respuesta.

Una vez realizada la petición y recibida la respuesta, la información recorre el camino inverso llegando con formato XBE32 desde el Gateway hasta la aplicación externa. En este punto es posible realizar el proceso de codificación inversa, de XBE32 a SENCOMLNG, para finalmente poder consultar la respuesta a la petición enviada.

1.2 Objetivos

El objetivo de este proyecto es la implementación práctica extremo a extremo de un sistema de comunicaciones para facilitar la solicitud de información de dispositivos inalámbricos ubicados en cualquier parte de la red.

Se basa en los trabajos *eXtensible Binary Enconding (XBE32)* [1] y LWESP: *Light-Weight Exterior Sensornet Protocol* [2] para desarrollar una aplicación extremo a extremo que implemente lo que este documento recoge.

Se busca poder realizar solicitudes de cualquier fenómeno medido por un sensor (humedad, temperatura, etc.) y recibir la respuesta desde una aplicación transparente al usuario. Para ello existe una lógica por detrás que será la encargada de traducir al lenguaje XML las peticiones y más adelante a una codificación ya definida, XBE32.

Los datos ya codificados se enviarán por la red hasta otro equipo que hará las veces de Gateway para la red 802.15.4 que se quiere monitorizar y que será quien se comunique directamente con el coordinador de la red. El coordinador será el encargado de mandar las peticiones que le lleguen a los distintos nodos inalámbricos de la red (*endpoints*).
Para terminar, este proceso de recolección de información recorrerá este camino en dirección inversa hasta llegar de nuevo a la aplicación cliente y así poder presentar la información al usuario final.

Como los dispositivos del extremo soportan poca carga computacional, se utiliza la codificación XBE32 que transforma las solicitudes XML a un lenguaje ligero a nivel de bit.

Otro de los objetivos es el estudio del conjunto de protocolos utilizados:

- Será importante el estudio de XBE32
- De la librerías de Java para la comunicación por red
- Las librerías para la lectura de XML
- Las bibliotecas y funciones de Jennic, el microprocesador que utiliza los sensores.

Una vez definido el protocolo será necesario especificar el diseño que se ha seguido a la hora de realizar la implementación y el funcionamiento final de la aplicación. Para ello, se hará uso de distintas herramientas como diagramas de flujo que nos permitirán comprender la ejecución del protocolo, capturas de pantalla de la aplicación en el lado cliente, diagramas de comunicación, y ejemplos de las distintas codificaciones de una misma petición a lo largo del proceso de petición respuesta.

Dentro de todo desarrollo es necesaria la realización de unas pruebas que aseguren su correcto funcionamiento, y nos proporcionen una cierta seguridad en cuanto a usabilidad y robustez.

1.3 Organización de la Memoria

La memoria está dividida en varios apartados. A continuación realizaremos un breve comentario sobre cada uno de ellos:

El primer apartado es este, en el que nos encontramos.

El segundo apartado se titula Estado del Arte y trata sobre las tecnologías utilizadas en el proyecto desde el punto de vista de las redes inalámbricas de sensores. Además, nombra otras tecnologías similares actuales.

El tercer apartado muestra un par de ejemplos reales de aplicaciones sobre IEEE 802.15.4. Con esto se pretende demostrar que son tecnologías que han pasado el nivel teórico y que se están utilizando en la actualidad.

El cuarto apartado, Tecnologías Utilizadas, describe las tecnologías utilizadas en este proyecto, que son SENCOMLNG y XBE32, y se dan ejemplos de cómo se implementan. Además, se dan detalles sobre los dispositivos utilizados para el desarrollo del proyecto.

El quinto apartado se refiere a la implementación de la aplicación. Detalla las distintas partes en las que está compuesta, cómo están realizadas y algunas decisiones tomadas en la realización del mismo.

El sexto apartado, Funcionamiento de la aplicación, ilustra el funcionamiento detallado de la implementación para dos consultas consecutivas.

El séptimo apartado se refiere a aspectos del proyecto fuera del desarrollo: se incluye un diagrama temporal aproximado de la realización del proyecto además de detallar el coste asociado al mismo.

Para terminar, en el octavo apartado se exponen las conclusiones alcanzadas en la realización del proyecto y qué trabajos futuros se podrían hacer.

2 Estado del Arte

En los últimos años el potencial de las redes inalámbricas de ámbito doméstico o personal se ha hecho patente.

Bien para monitorizar una vivienda o para interconectar todos los aparatos electrónicos de uso personal (móvil, reproductor mp3, ordenador portátil, impresora, etc.), estas redes de bajo alcance tendrán cada día mayor uso en nuestras vidas [3].

En el desarrollo industrial, aquellas tecnologías basadas en el estándar IEEE 802.15.4 se posicionan más fuertemente frente a otras basadas en IEEE 802.11 o Bluetooth.

El apoyo de grandes compañías y estudios comparativos demuestran que para el tipo de redes de interconexión de aparatos domésticos, aquellas basadas en IEEE 802.15.4 se consideran mejor apuesta que WiFi [4].

En esta sección se explicarán las principales tecnologías que hoy en día conforman las comunicaciones inalámbricas en redes de sensores.

2.1 Wireless Sensor Networks

Una red inalámbrica de sensores (*Wireless Sensor Network*, WSN) es una red compuesta por sensores que pueden interactuar entre ellos y monitorizar parámetros físicos. Suelen ser redes de área personal utilizadas, por ejemplo, para la comunicación entre dispositivos cercanos al entorno de trabajo de una persona.

Su alcance es de decenas de metros y pueden utilizarse para la comunicación entre dispositivos en redes aisladas o para llegar a una red superior e incluso Internet.

Para el caso de una red de área personal, estos dispositivos pueden estar unidos por buses, como USB o Firewire o, para el caso particular de las redes inalámbricas de sensores, mediante tecnologías como IrDA, Bluetooth, UWB, Z-Wave o ZigBee.

Un concepto primordial en este tipo de redes es el plug-in: cuando dos dispositivos se encuentren en la misma área de alcance o cerca de un servidor central se puedan conectar como si estuvieran cableados.

Otra característica importante es la habilidad de bloquear el acceso a ellos de un modo selectivo, evitando problemas de seguridad.

Estas redes pueden estar formadas por un gran número de nodos, que utilizarán tecnología radio para comunicarse. Esto implica que ciertos problemas como la interferencia entre señales o la propagación multicamino estarán presentes. Habrá que buscar el compromiso entre estos factores y los objetivos de bajo coste e implementación sencilla que buscan las tecnologías que conforman las redes WSNs.

2.2 802.15.4

El IEEE 802.15.4 [5] [6] es un estándar que especifica las capas física y de acceso al medio para cierto tipo de redes inalámbricas de área personal y con requisitos bajos de velocidad.

Este estándar es la base de tecnologías como ZigBee, WirelessHART y MiWi, de las que hablaremos más adelante.

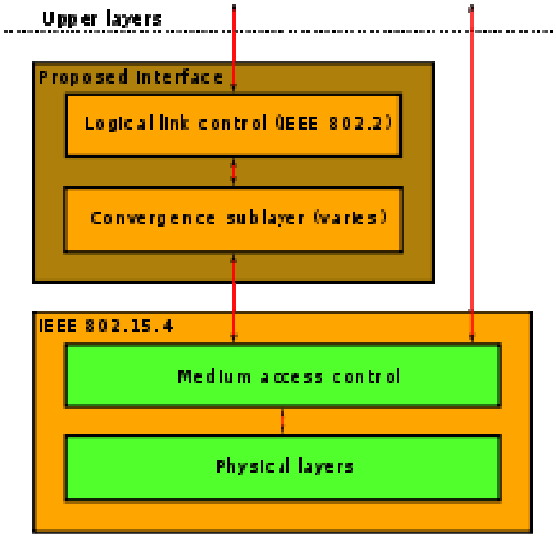
802.15.4 ofrece a las capas superiores los fundamentos para la comunicación de dispositivos en un área de poco alcance, centrada en bajo coste, poca infraestructura, baja velocidad y requisito estrictos de consumo de batería.

Las características más importantes de este estándar son:

- Es adecuado para la comunicación en tiempo real mediante el uso de *slots* temporales.

- Evita colisiones mediante CSMA/CA.
- Presenta un soporte integrado para comunicaciones seguras.
- Los dispositivos también incluyen funciones de control de energía en aspectos como calidad del enlace y detección de consumo.

2.2.1 Arquitectura de 802.15.4



1. Arquitectura IEEE 802.15.4

Los dispositivos están concebidos para interactuar entre sí sobre una red inalámbrica sencilla.

La definición de las capas está basada en OSI, aunque solo están definidas las dos capas más bajas en el estándar: la capa física (PHY) y la capa de control de acceso al medio (MAC).

El resto de niveles superiores no están definidos en este estándar. Esto es labor de otros protocolos, alguno de los cuales se detallarán en el apartado 2.3.

La interacción con las capas superiores es posible gracias a la subcapa de enlace lógico *(IEEE 802.2 logical link control sublayer*) que accede a la capa de control de acceso al medio (MAC) mediante una subcapa de convergencia.

2.2.1.1 La capa física (PHY)

Esta capa proporciona el servicio de transmisión de datos y el interfaz a la entidad de administración de la capa física *(physical layer managemente entity)*.

La entidad de administración de la capa física ofrece el acceso a cada una de las funciones de administración de cada capa y mantiene una base de datos con información relacionada de las redes de área personal a la que pertenece.

La capa PHY administra el transceptor RF físico y realiza la selección de canal y las funciones de control de energía y señal.

Esta capa opera en una de las tres bandas reconocidas en el estándar:

- En Europa trabaja en la banda de los 868.0-868.6 MHz para Europa, lo que permitía solo canal de comunicación en el estándar de 2003. Ha sido aumentado a 3 canales en la versión del estándar de 2006.
- En Norte América opera en la banda de los 902-928 MHz. En el primer estándar de 2003, esto permitía hasta 10 canales de comunicación. En el estándar de 2006 el número de canales aumentó a 30.
- En general, en todo mundo, incluyendo Europa y Norte América, opera en la banda de los 2400-2483.5 MHz, donde puede tener hasta 16 canales (estándares 2003 y 2006). Esta banda es la más utilizada.

El estándar de 2003 [5] especificaba dos capas físicas basadas en técnicas de DSSS:

- Una banda de 868/915 MHz con tasas de transferencia de 20 y 40 kbps. La versión de 2006 mejora estas tasas, llegando a soportar 100 y 250kbps.
- Otra banda de 2450 MHz con una tasa de 250 kbps.

La revisión de 2006 [6] sustituye en este aspecto al estándar de 2003 y define cuatro nuevas capas físicas según la modulación que utilicen. Por ello, las de 2003 quedan obsoletas y las nuevas son las siguientes:

- Una capa basada en DSSS que opera en las bandas de 868/915 MHz con codificación binaria.
- Otra capa PHY también basada en DSSS y operando en las mismas bandas, 868/915 MHz, pero con codificación OQPSK. Esta capa es opcional.
- Una tercera capa también basada en DSSS. Trabaja en la banda de los 2450 MHz y utiliza OQPSK
- Por último, se define otra capa opcional en las bandas de los 868/915 MHz. Utiliza una combinación de codificación binaria y de *amplitude shift keying* (ASK) basado en PSSS. Permite el cambio dinámico entre las capas físicas de 868 y 915 MHz.

En 2007 se publicó la especificación IEEE 802.15.4a, [7] que aumenta el número de capas físicas a seis:

- Incorpora una quinta capa que utiliza *Direct Sequence Ultra-Wideband* (UWB). Está localizada en tres rangos de frecuencias: por debajo de 1 GHz, entre 3 y 5 GHz y entre 6 y 10 GHz.
- La sexta capa, utiliza *Chirp Spread Spectrum* (CSS) y trabaja en la banda 2450 MHz.

En abril de 2009 se publicaron dos revisiones más del estándar, IEEE 802.15.4c [8] e IEEE 802.15.4d [9], que aumentaban los tipos de capas físicas:

- Una capa en los 780 MHz, con codificación OQPSK o MPSK.
- Otra capa en la banda de los 950 MHz, que utiliza GFSK o BPSK.

Los tipos de capas físicas contempladas en IEEE 802.15.4 serían por tanto las siguientes:

Modo de modulación	**Banda (MHz)**	**Modulación/es**	**Publicación**
DSSS	868/915 MHz	Binaria	802.15.4- 2006
DSSS	868/915 MHz	OQPSK	802.15.4- 2006
DSSS	868/915 MHz	OQPSK	802.15.4- 2006
PSSS	868/915 MHz	Binaria y ASK	802.15.4- 2006
DSSS UWB	< 1GHz, 3-5 GHz, 6-10 GHz	*Sin especificar*	802.15.4- 2006
CSS	2450 MHz	DPSK	IEEE 802.15.4a
DSSS	780 MHz	OQPSK, MPSK	IEEE 802.15.4c-2009
DSSS	950 MHz	GFSK, BPSK	IEEE 802.15.4d-2009

Tabla 1. Capas PHY en IEEE 802.15.4

2.2.1.2 La capa de control de acceso al medio (MAC)

La capa de control de acceso al medio es la inmediatamente superior a la PHY. Esta capa permite la transmisión de tramas a través del canal físico, accede por sí misma al canal físico y también al *beaconing* de la red. Además controla la validación de tramas, garantiza *slots* temporales y maneja la asociación de nodos.

La capa MAC proporciona dos servicios a las capas superiores a través de dos puntos de acceso a servicios (SAPs):

- El servicio de datos hacía las capas superiores, al que de accede por la subcapa MCPS-SAP.
- El manejo de servicios MAC, al que se accede por la subcapa encargada del manejo de entidades (MLME-SAP).

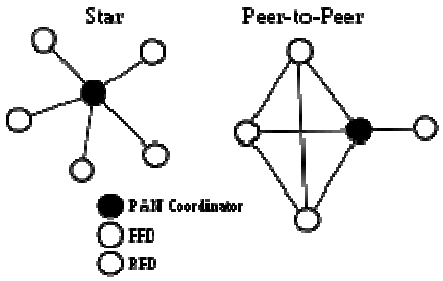
2.2.2 Redes 802.15.4

Las redes IEEE 802.15.4 se caracterizan por topologías sencillas. El estándar contempla dos tipos de nodos de red:

- *Full-Function Devices* (FFD): este tipo de nodo puede ejercer tanto de coordinador de una red como de nodo corriente. Es el que implementa el modelo de comunicación que le permite comunicarse con otros dispositivos. También puede enrutar mensajes.

- *Reduced- Function Devices* (RFD) o *endpoints*: este tipo de nodos están diseñados para ser dispositivos sencillos con pocos requerimientos tanto en comunicaciones como en energía, por ello solo pueden comunicarse con dispositivos FFDs y nunca podrán actuar como coordinadores.

En cuanto a la arquitectura de la red, el estándar contempla las topologías en estrella, *peer-to-peer* o en modo *cluster*.
Las topología en estrella o *peer-to-peer* necesitan al menos un nodo coordinador en la red, que tendrá que ser un dispositivo FFD.



2. Topologías en estrella y peer-to-peer

Las redes punto a punto pueden formar estructuras arbitrarias o conexiones que estarán limitadas por la distancia entre cada dos nodos. Son la base de redes ad-hoc, capaces de organizarse y administrarse por sí mismas. Como el estándar no define una

capa de red, el enrutamiento no está soportado y serán otros protocolos de nivel superior los que se encarguen de esto.

La estructura puede ampliarse mediante subredes, a través del concepto de red mallada. Cada subred tendrá un coordinador local por clúster, además habrá un coordinador global para el total de las subredes que controlará al total de los coordinadores locales.

La topología en estrella está formada por un nodo coordinador que es el nodo central. Esta red se origina cuando un FFD decide crear su propia PAN y se defina así mismo como coordinador, escogiendo él el identificador de la red local creada (PAN ID). Posteriormente, otros dispositivos se incorporarían a esta red.

Cada dispositivo tiene un identificador único de 64 o 16 bits, que facilita la comunicación dentro de entornos restringidos.

2.2.3 Estructura de la información

El estándar especifica cuatro tipos de tramas: *data*, *acknowledgment*, *beacon* y *MAC command frames*.

Adicionalmente existe una estructura superior definida por el coordinador: la supertrama. Está delimitada por dos *beacons* y proporciona información de sincronización y coordinación a otros dispositivos. Estas tramas se suelen utilizar en contextos donde la comunicación debe permanecer inactiva durante largos períodos de tiempo

La supertrama está formada por 16 slots de la misma longitud que pueden dividirse en activos e inactivos. Durante los inactivos el nodo coordinador podrá entrar en modo de

ahorro de energía, no teniendo así que controlar su red. Para evitar colisiones entre supertramas se utiliza CSMA/CA.

El envío de datos de *endpoint* a coordinador requiere sincronización de fase por *beacon* seguida de transmisión CSMA/CA, el asentimiento (ACK) es opcional. Estas transferencias suelen seguir a una petición (*request*): si se están utilizando *beacons*, se utilizarán para indicar la petición, el coordinador asentirá el *request* y se enviará la información en paquetes confirmados por el dispositivo. Lo mismo ocurrirá cuando no se utilizan supertramas, solo que sin *beacons* que controlen los mensajes pendientes.

2.2.4 Fiabilidad y seguridad

Como ya se ha indicado, al medio físico se accede mediante el protocolo CSMA/CA:

Las redes que no utilizan *beaconing* utilizan una variación sin slots basada en escuchar el medio en conjunción con un algoritmo de *random exponential backoff,* que espacia las retransmisiones del mismo datagrama. Cuando hay *beaconing*, la transmisión de datagramas utiliza *slots* que no están previamente fijados.

Los mensajes de confirmación funcionan de manera distinta: dado que se busca minimizar el consumo energético, se implementan comprobaciones periódicas de mensajes de error dependiendo de necesidades del sistema.

Los mensajes de confirmación pueden ser opcionales, en cuyo caso se asume en éxito en la transmisión.

En cuanto a la seguridad, la subcapa MAC ofrece facilidades para ello. Las capas superiores pueden especificar claves para realizar criptografía simétrica con el objetivo de proteger el *payload*. También pueden restringir la comunicación a un cierto grupo de dispositivos o permitir solo la comunicación mediante enlace punto a punto. Estos dispositivos estarán especificados en listas de control de acceso.

Además, la capa MAC realiza controles entre llegadas sucesivas para asegurarse de que paquetes antiguos con información que ya no es válida no lleguen a las capas superiores.

2.3 Tecnologías inalámbricas para WSNs

A continuación detallaremos algunas tecnologías que se basan en 802.15.4 e implementan los niveles por encima de ésta para proporcionar comunicaciones inalámbricas.

Todas estas tecnologías buscan a la comunicación entre pequeños dispositivos que emitan a baja potencia y se sitúen en redes de sensores inalámbricas, además de ser aplicaciones con baja velocidad de transferencia y poco consumo de batería.

2.3.1 ZigBee

ZigBee [10] es una especificación para la comunicación de redes inalámbricas de sensores. Se basa en el estándar IEEE estándar 802.15.4-2003 para definir las capas superiores que éste no define. La especificación está mantenida por la *ZigBee Alliance* que desarrolla la especificación y certifica su correcta implementación. La última especificación es del año 2007 y contiene dos *stacks* distintas: *stack profile*, llamada ZigBee, para uso doméstico e iluminación comercial y *stack profile 2*, ZigBee Pro, para usos más avanzados.

Para poder desarrollar productos comerciales con esta tecnología es necesario registrarse en la ZigBee Alliance, sin embargo, para otros usos la especificación está disponible de un modo gratuito.

Como utiliza 802.15.4, opera en las bandas industriales, científicas y médicas (ISM), que están en los 868 MHz en Europa, en los 915 MHz en Estados Unidos y Australia y

en los 2.4 GHz para todo el mundo, incluido Europa y EE.UU. En la frecuencia de los 2.4 GHz hay 16 canales.

La especificación contempla tres tipos de dispositivos ZigBee:

- El nodo coordinador (ZC): es la base de la red y puede también enlazar con otras redes distintas. Hay un nodo coordinador por cada red, dado que es el que crea la red a la que pertenece. Se corresponde con el F*ull- Fuctioning Device* de IEEE 802.15.4. Puede almacenar información sobre la red.
- *ZigBee Router (ZR)*: puede actuar como un router intermedio, pasando información a otros dispositivos. Además puede funcionar como un dispositivo normal.
- Dispositivo final ZigBee *(ZigBee End Device, ZED)*: tiene la funcionalidad justa para comunicarse con el nodo padre. Se corresponde con el *Reduced-Functioning Device* de IEEE 802.15.4.

ZigBee proporciona métodos para establecer comunicaciones seguras mediante la protección en la creación y en el transporte de claves criptográficas, mediante el cifrado de tramas y a través de control en el dispositivo. Se basa en el marco de seguridad definido por IEEE 802.15.4 y utiliza pares de claves simétricas.

Por último, en un solo dispositivo ZigBee pueden existir hasta 240 aplicaciones.

2.3.2 6LOWPAN

6lowpan es otra tecnología para redes inalámbricas domésticas basadas en IP. 6LoWPAN es un acrónimo de *IPv6 over Low power Wireless Personal Area Networks*.

Una red 6LoWPAN es una red sencilla de bajo coste que permite conectividad inalámbrica con aplicaciones con limitaciones de gasto energético. Además, incluye dispositivos que la conectan con el mundo físico, por ejemplo, sensores. Esta red sigue el estándar IEEE802.15.4.

Algunas características [11][12] de este tipo de redes son:

- Tamaño de paquetes pequeño. El máximo tamaño en la capa física es de 127 bytes, en el nivel de enlace es 102 bytes. La seguridad a nivel de enlace incluye una cabecera que puede llegar a ser de 21 bytes, dejando un tamaño para almacenar información de 81 octetos.
- Soporte para direcciones cortas de 16-bits y de direcciones IEEE de 64 bits.
- Bajo ancho de banda: velocidades de 250 kbps, 40 kbps, y 20 kbps según lo definido en cada una de las capas físicas.
- Topologías en estrella y red mallada.
- Bajo consumo energético.
- Bajo coste de dispositivo, lo que implica pocos requisitos de procesamiento y memoria.
- Capacidad para un gran número de dispositivos en una red.
- Los dispositivos suelen desplegarse de modo ad-hoc, por ello no tiene definido un servicio de localización de dispositivos.
- Seguridad basada en AES.

Las principales ventajas de las redes 6LoWPAN se basan en la aplicación de la tecnología IP, cuyo uso está ampliamente extendido. Por ello existen herramientas para el diagnóstico y administración de redes. Además la interacción entre dispositivos 6LoWPan con otros dispositivos IP es inmediata.

Otro aspecto importante es que el amplio número de dispositivos en una red 6LoWPAN hace necesarios la autoconfiguración de las redes y un gran espacio de direccionamiento. Estos requisitos ya los cumple IPV6.

Sin embargo, el principal inconveniente que presentan es la necesidad de comprimir las cabeceras IPV6 y las capas que lo forman, dada la limitación del tamaño de paquete.

Otros problemas es que los protocolos utilizados en estas redes deben tener una configuración mínima y ser capaces de reparar problemas en las redes por sí mismos dados. Además es necesario un servicio de descubrimiento de servicios, aunque sea muy sencillo.

Por estas razones, los objetivos prioritarios para 6LoWPAN son los siguientes:

- Una capa de fragmentación y reensamblado dado que el tamaño de paquete es menor que el mínimo de 1280 octetos de IPV6.

- Compresión de cabecera. Cuanto más se reduzca esto, más capacidad habrá para datos.
- *Address Autoconfiguration* y un método para generar un identificador de interfaz desde EUI-64. Esto además reduciría la cabecera.
- Los nodos IPv6 tienen asignadas direcciones IP de 128 bits. Los dispositivos IEEE 802.15.4 pueden utilizar direcciones extendidas IEEE de 64 bits o, una vez asociadas, direcciones de 16 bits únicas en una PAN. Es necesario encontrar un modo de pasar de una longitud de dirección a otra y mantener el direccionamiento.
- Queda por definir un protocolo de enrutamiento en red mallada que soporte redes multi-hop.
- En cuanto a capas de más alto nivel, es vital buscar codificaciones los más ligeras posibles para la implementación de las capas superiores.

2.3.3 DASH7

DASH7 es una tecnología de comunicación inalámbrica basada el estándar ISO/IEC 18000-7. Es de estándar abierto y está promovida por la *DASH7 Alliance* [13] .

El estándar i ISO/IEC 18000 [14] define una serie de tecnologías de RFID, cada una en un rango de frecuencia. La parte 7 se centra en la banda de los 433 MHz y es en la que se basa DASH7, de ahí su nombre.

En este estándar dispositivo final es conocido cómo *tag* y al coordinador o el que empieza la comunicación es el interrogador. Se especifican las siguientes características mínimas:

- Identificar un *tag* en su área de alcance.
- Leer datos.
- Escribir datos o manejar sistemas de solo lectura.
- Seleccionar por grupo o dirección.
- Manejar varios *tag* en la misma área.
- Detección de errores.

En cuanto a la banda de frecuencia, especifica un rango de operación de 433,92 MHz y modulación FSK.

El paquete de datos que transmite la información entre el dispositivo interrogador (el que empieza la comunicación) y el *tag* está formado por tres campos: preámbulo, datos y final.

DASH7 opera en la frecuencia de los 433.92 MHz, y por ello puede atravesar agua y hormigón y no verse afectado por las radiaciones de entornos domésticos como podrían ser las redes WiFi.

Además consume menor energía que ZigBee y presenta un alcance ajustable entre los 10 metros y los 10 kilómetros, con velocidades de transferencia están entre los 28 kbps y los 200 kbps y poca latencia en la comunicación. Su tamaño de paquete es de 256 bytes.

Soporta comunicación *tag-to-tag* por lo que puede llegar a sustituir las redes malladas. Aunque su torre de protocolos es pequeña, también contempla sensores, encriptación e IPV6.

DASH7 trabaja a nivel de sesión pero dado lo esporádicas que pueden ser las comunicaciones también trabaja a ráfagas. Su modo de comunicación principal es asíncrono: comando-respuesta.

El principal cliente de esta tecnología es el Departamento de Defensa de los Estados Unidos. Otras grandes empresas detrás de esta tecnología son Texas Instruments.

2.3.4 WirelessHART

WirelessHART es otro protocolo inalámbrico utilizado para aplicaciones de control, medición de procesos y entornos domésticos [15]. Está basado en el protocolo de comunicación HART y ha sido desarrollado por la *HART Communication Foundation*.

El protocolo HART (*Highway Addressable Remote Transducer* Protocol) [16] es una implementación de un *fieldbus* utilizado para automatización industrial digital. Se comunica sobre el cableado de instrumentos analógicos de 4-20mA.

Tiene dos modos de comunicación: *peer-to- peer y multi-drop*, que permite hasta 15 instrumentos por cable.

Los paquetes de HART, ocupan entre 20 y 282 bytes y están formados por los campos preámbulo, bit de inicio, dirección, comando, número de bytes de datos, estado, datos y *checksum*.

WirelessHART es la implementación inalámbrica de HART y tiene las siguientes características:

- Soporta redes malladas.
- Contempla salto de canales y sincronización de mensajes.
- Es fiable incluso en presencia de interferencia. Además, proporciona seguridad mediante, entre otras cosas, encriptación, verificación, autenticación, manejo de claves,
- Bajo consumo energético.
- Su *payload* máximo es de 127 bytes.
- Y, en base al estándar IEEE 802.15.4-2006, tiene una velocidad de transmisión de 250 kbps y opera en la banda de frecuencia de 2400-2483.5 MHz bajo una modulación OQPSK y de secuencia directa.

2.3.5 ZWAVE

Z-Wave es un protocolo inalámbrico diseñado para la automatización doméstica, en concreto para el control remoto de de aplicaciones e iluminación en entornos [18]

residenciales y comerciales. Es un protocolo propietario de un consorcio internacional de fabricantes.

Al ser propietaria, su estándar no es abierto y sólo está disponible para clientes de Zensys.

La tecnología utiliza comunicación radio de baja potencia en dispositivos electrónicos y aparatos domésticos. Está optimizado para comandos con poca estructura, del estilo ON/OFF o subir/bajar (en interruptores o termostatos, por ejemplo) pero con la capacidad de incluir metadatos del dispositivo en las comunicaciones.

Z-Wave es una tecnología de red mallada, en la que cada nodo puede enviar y recibir comandos de control y utilizar nodos intermedios para enrutar.

Admite velocidades de 9600bps o 40bps y su alcance puede llegar a ser de 30 metros en espacio abierto, siendo menor en entornos cerrados o con obstáculos.

Trabaja en las bandas de frecuencia: 908.42MHz en EE.UU., 868.42MHz en Europa, 919.82MHz en Hong Kong y 921.42MHz en Australia y Nueva Zelanda.

Su modulación el GFSK.

Es apto para grandes redes de sensores, no solo por su tamaño de alcance sino por que soporta hasta 232 dispositivos por red, con la opción de unir redes si fuesen necesarios más dispositivos. Además, es compatible con ZigBee.

Por estas características sus posibles aplicaciones son las habituales de este tipo de tecnologías de baja frecuencia para redes de área personal: control remoto doméstico (iluminación), control ambiental (temperatura), seguridad, control de riego, etc.

2.3.6 MiWi

MiWi [19]es otro ejemplo de protocolos inalámbricos para redes de área personal basados en IEEE 802.15.4.

Es un proyecto de código abierto desarrollado por Microchip Technology, pero para su uso es necesario utilizar un microcontrolador en concreto: el transceptor MRF24J4 de Microchip Technology.

Algunas de sus características son:

- Posee una tasa de transferencia es de 250 kbps, llegando a alcanzar 100 metros.
- A través del transceptor MRF4J4, trabaja en el espectro de los 2,4Ghz a través del transceptor MRF4J4 realizando un bajo consumo energético.
- Soporta topologías punto a punto, en estrella o malladas.
- Puede tener un máximo de 8 coordinadores por red, y 127 hijos por coordinador. Es decir, 1024 nodos por red.

A pesar de no ser compatible con ZigBee, MiWi copia alguna de sus funciones. De todos modos, es un estándar menos complejo y de código reducido: el stack MiWi de un nodo coordinador ocupa 10k de ROM y en los nodos finales ocupa unos 3k de ROM.

3 Ejemplos de aplicaciones

A continuación se detallan dos aplicaciones en la vida real de redes inalámbricas de sensores, ambas sobre ZigBee. Uno es un ejemplo de uso para entornos domésticos y ya está implantado y en uso.

El otro es una propuesta del uso de IEEE 802.15.4 para entornos sanitarios que solamente ha sido comprobado en la fase de pruebas previa a la realización del *paper* donde se expone.

3.1 WSNs en entornos domésticos: Lectura de contadores

La lectura de contadores en un trabajo mecánico que supone además un coste asociado al personal encargado ir casa por casa leyendo el consumo realizado.

ZigBee es un estándar adecuado para realizar estas medidas de un modo telemático: la cantidad de datos a transmitir es pequeña y los lectores de contadores son dispositivos idóneos para esta tecnología por no requerir de excesiva complejidad.

En Estados Unidos, la empresa de suministro eléctrico *Southern California Edison*, que sirve a unos 13 millones de personas en un área de unos 129500 kilómetros cuadrados, lanzó un piloto para probar la viabilidad de instalar este tipo de lectores, que llamó *Smart meters*, y así mejorar la experiencia de usuario y reducir el consumo energético total y máximo de la compañía [20].

Los cálculos iniciales estimaron una compatibilidad del 94% con la estructura ya presente, para el resto de casos fue necesarios considerar una alternativa como por ejemplo puentes y repetidores entre redes.

En este piloto se planteó la colocación de sensores en diferentes aparatos, como electrodomésticos y termostatos, que proporcionasen información sobre el consumo en

cada momento. Esto permitió detectar aquellos electrodomésticos con un gasto energético no óptimo o intentar reducir el consumo medio del hogar.

La primera fase de pruebas se realizó en 2008. En ella, el equipamiento estuvo formado por:

- un ordenador portátil,
- un nodo coordinador conectado al portátil,
- un medidor residencial ZigBee, localizado en el lugar del antiguo contador eléctrico
- un nodo sensor
- no había repetidores

Las pruebas se realizaron tanto en presencia de tráfico WiFi como en ausencia de este. En ambos casos se probó comunicación bidireccional entre medidores proporcionados por un vendedor y dispositivos ZigBee dentro de la red doméstica.

La captura de datos se respaldó por una comprobación visual y anotación manual para poder contrastarla estos datos con los recogidos mediante la red inalámbrica doméstica.

Además, se introdujo en la red un analizador de red inalámbrica para poder observar el tráfico entre los nodos sensores (termostatos) y el medidor. También se midió la radiación en el rango de los 2.4 a los 2.5 GHz antes, durante y después de cada batería de pruebas, tanto con tráfico WiFi como sin él.

Los primeros resultados fueron favorables para ZigBee, y por tanto para la aplicación de este sistema en casos reales:

- ZigBee es capaz de migrar a diferentes canales si hay demasiado tráfico en su canal por defecto. En las pruebas se buscó forzar interferencia con WiFi.
- Cuatro edificios tenían unas características adversas de cara la radiofrecuencia. En algunos casos la señal tuvo que viajar durante más de 80 minutos a través de 3 paredes exteriores y 2 interiores.
- ZigBee tiene un canal en la parte alta de los 2.4GHz que no solapa con WiFi, haciendo capaz la comunicación en presencia de tráfico WiFi.

La primera instalación de uso comercial se realizó en 14 de septiembre de 2009 en

Downey, California, Estados Unidos [21]. La implantación continua a día de hoy, habiendo sido instalados en febrero de 2011 más de 2 millones de dispositivos ZigBee, conocidos como *Smart meters* [22].

3.2 WSNs en entornos industriales: monitorización de cultivos

IEEE 802.15.4 también puede utilizarse para la monitorización de campos de cultivo. En la universidad De la Salle, en Manila, Filipinas, han desarrollado un sistema de monitorización y administración de plantaciones mediante IEEE 802.15.4 y ZigBee[23].

Este proyecto, conocido como CropWise, pretendía comprobar que mediante la monitorización de parámetros, en concreto la humedad del terreno, y mediante unas válvulas para control de riego se podía optimizar una plantación tanto en ahorro de agua como en beneficios para la plantación.

El material para realizar las pruebas está dividido en:

- Dispositivos finales, llamados vainas, divididos en dispositivos de control y dispositivos sensores. Están compuestos por un módulo de comunicación ZigBee y la electrónica necesaria para realizar sus funciones. Si es un nodo sensor, parametrizará la humedad del suelo mientras que si es un dispositivo de control se encargará del control de riego. Las vainas tienen capacidad para almacenar sensores de diferentes parámetros.
- Una estación base, que será el nodo coordinador de la red inalámbrica y estará conectada al equipo que contenga la aplicación vía puerto serie.
- Un equipo final en el que se podrá interactuar con una aplicación especifica diseñada para consultar los datos de la plantación y realizar las acciones necesarias sobre ella (abrir el riego, establecer los límites de humedad del terreno, etc.).

Para la comunicación entre la aplicación y los nodos hardware, se ha desarrollado un protocolo intermedio que contiene una serie de campos:

- Delimitador, define si el paquete es de petición o repuesta

- Identificador del paquete.
- Parte de datos: contiene un código de operación, el sensor al que va dirigida la petición, el puerto final y un campo valor según el tipo de operación (por ejemplo, el sensor número 3 tendrá un límite de humedad de un 35%).

Las pruebas se realizaron mediante dos plantaciones en paralelo: una monitorizada mediante ZigBee y otra que se regaba de modo tradicional. Estas pruebas duraron los 45 días del ciclo de plantación del pechay, un tipo de verdura.

El funcionamiento era el siguiente: cuando los sensores detectaban que el porcentaje de humedad del suelo descendía del 30%, las válvulas de riego se abrían hasta alcanzar ese umbral de modo que solo se reemplazase el agua necesaria.

Además de comprobar el consumo de agua, se midió también el consumo energético derivado del gasto eléctrico de la red de sensores.

Al final de los 45 días se pudo observar que:

- El consumo de agua en la plantación monitorizada era un 34,31% menor.
- La productividad de pechays, en kilos por pies al cuadrado, era mayor en un 59,09% en la zona monitorizada.
- Según al precio de venta del mercado y descontando el gasto eléctrico, la plantación monitorizada proporcionaba un 34,42% más de beneficios.

Así queda comprobada que el uso de tecnologías basadas en IEEE 802.15.4 es aplicable a campos como la producción agrícola.

4 Tecnologías utilizadas

En este apartado hablaremos de las tecnologías específicas de este proyecto, ambas expuestas en los trabajos *LWESP: Light-Weight Exterior Sensornet Protocol* [1] y en *eXtensible Binary Encoding (XBE32)* [2] .

Dado que existen varias tecnologías inalámbricas para WSNs, cada una con su propio intercambio de mensajes, en estos modelos es necesario un modo de de unificar redes y lenguajes.

Una aproximación para unificar este modelo es SENCOMLNG, que es un lenguaje XML orientado a comunicaciones de redes de sensores inalámbricas. Por estar orientado a estas comunicaciones contempla etiquetas para campos como dirección destino, tipo de datos, identificador de mensaje, etc.

Pero se parte del hecho de que XML ocupa gran cantidad de bits y transmitirlo tal cual por cualquier tipo de red sería pesado, por ello una alternativa es codificar la información representada en XML en otra codificación más ligera.

Una propuesta para ello es XBE32, que mediante el protocolo LWESP asigna un valor binario a las etiquetas contempladas en SENCOMNLG y así conseguir minimizar el ancho de banda.

Con SENCOMNLG y XBE32 se consigue que una petición inicial en XML se transmita por la red reducida hasta el un 35% [1].

4.1 SENCOMLNG

SENCOMLNG (SENsor COMmunication LANGuage) es un lenguaje basado en XML específico para la comunicación entre aplicaciones y redes de sensores [1]. Este lenguaje define cuatro operaciones:

- GetData: se utiliza para solicitar una medida a un sensor.

- SetData: se utiliza para establecer un valor en un sensor. Por ejemplo, encender una bombilla.

- EventSet o Trap: está pensada para que un sensor notifique cuando algunas de sus medidas traspase un límite preestablecido.

- Monitoring: se define para realizar mediciones periódicas de un sensor específico dentro de la red.

La raíz del documento en SENCOMLNG es <LWESPv1> (posteriormente modificado a lp1v) y contiene la estructura completa.

El siguiente elemento que contienen todos los mensajes SENCOMLNG es la cabecera, definida con <header>. Está formada por cuatro campos que definen el identificador del mensaje, los identificadores de origen y destino y una marca de tiempo. Están representados por etiquetas <msgID>, <srcID>, <dstID> y <tstamp>, respectivamente.

El elemento siguiente contendrá uno de los siguientes tres tipos de operaciones: requests, replies (ambas operaciones síncronas) y mensajes de datos (asíncronos). Estos últimos se utilizan en las operaciones de tipo monitoring y eventSet.

Una petición de tipo request puede contener más de una operación, pero aún así contendrá solo una cabecera, derivando esto en ahorro de bytes.

Las etiquetas de cada tipo de operación son request son <getData>, <setData>, <monitorStart> y <eventSet>. Estas operaciones conforman elementos XML y contienen los siguientes campos:

- Un identificador de operación, que llevará la etiqueta <sensorXID>, donde X especifica el número de bits utilizados para la dirección del sensor.

- Un campo de datos identificado por las etiquetas <data> y <thresholdData>. Tiene distintos campos según la petición es cuestión aunque todos incluyen la etiqueta <dataCat> que define el tipo de datos solicitados (temperatura, humedad, etc.). La longitud de este campo son 32 bits donde los primeros 24 definen el tipo de datos mientras que los 8 últimos especifican la unidad de medida (por ejemplo, temperatura en grados Celsius o Kelvin).

En la operación setData es necesario añadir dentro de campo dataCat una etiqueta y un valor, el que se quiere establecer en el sensor. Para ello podrá utilizarse cualquier tipo de datos definido en XBE32. Cuando se utilicen datos del tipo <intXValues>, podrá utilizarse un campo que haga referencia a la escala, representado por la etiqueta <dataScale>. Este campo funcionará del siguiente modo: si dataScale vale -1 y el valor recogido es 35, el valor real de la medición será 3,5.

- Después del sensorID y el elemento de la operación que estamos realizando, en las operaciones de eventSet se incluye otro campo más que establece el umbral que activará la operación. Para ello se utiliza la etiqueta <op>, que contendrá un valor numérico según el tipo de operación (>, <, >=, <=, == ó !=). Esta etiqueta funciona junto con el valor recogido en <thresholdData>.

- Las operaciones de monitorización incluyen dos elementos adicionales: <measurePeriod> <reportPeriod>. Estos especifican cada cuanto hay que realizar una medida y con qué frecuencia se enviará esta información desde el sensor al coordinador.

De un modo opcional, estas operaciones podrán llevan un campo <lifetime> que define el tiempo de vida de la operación.

Además de estas cuatro operaciones básicas, se definen operaciones para iniciar y finalizar las operaciones de monitorización y eventSet:

- La operación de refresh, identificada por las etiquetas <monitorRefresh> y <eventSetRefresh>, se lanzará periódicamente a la red, desde la aplicación, para que quede patente que las operaciones de eventSet o monitorización deben seguir en activo.

- La operación de stop finalizará las dos operaciones ya nombradas. Viene definida por las etiquetas <monitorStop> y <eventSetStop>.

También se definen mensajes de respuesta para cada una de las operaciones básicas. Vienen identificados por las etiquetas <getDataReply>, <setDataAck>, <monitorStartAck> y <eventSetAck>.

Para las operaciones de respuesta a eventos y de monitorización, se define una etiqueta adicional, <processID> , que junto a <sensorID> representará un proceso en concreto. Esto facilitará la sintaxis de las operaciones de refresh y stop, minimizando la carga de información en los mensajes. Adicionalmente, se contempla un mensaje de error que incluirá la operación que ha fallado y la razón del error.

Por último, para las operaciones asíncronas se especifican mensajes de datos asíncronos generados. Están representados por la etiqueta <eventData> e incluyen los campos <sensorID>, <processID>, <data> que incluye <dataCat> y nuevo campo <timeDelta>, etiquetas del tipo XBE32 y <dataScale>.

El campo timeDelta indica la diferencia de tiempo entre el valor el <tstamp> y el momento en el que la información del sensor fue obtenida.

Si el mensaje de datos asíncronos el fruto de una operación del tipo <monitorStart>, el campo <eventData> pasará a ser <monitorData>, manteniendo el formato de la operación eventData pero concatenando las distintas medias obtenidas en el proceso de monitorización.

A continuación se muestran ejemplos de codificación en SENCOMLNG de algunas de las operaciones definidas para el caso particular de este proyecto. Las codificaciones ajenas a SENCOMLNG se explicarán en el apartado 5.

- La figura 3 muestra un ejemplo de una operación Get al sensor número 1 de la red de sensores. Se está solicitando la luz ambiental, codificada con el valor 7:

```
<?xml version="1.0" encoding="UTF-8" ?>
<lpv1>
- <header>
    <msgID>0x3A468A7D</msgID>
    <srcID>0x2742389798</srcID>
    <dstID>0x2742389772</dstID>
    <tstamp>978018678</tstamp>
  </header>
- <getData>
    <sensorID>0x1</sensorID>
  - <data>
      <dataCat>0x00000700</dataCat>
    </data>
    <lifetime>1000</lifetime>
  </getData>
</lpv1>
```

3. Ejemplo de Get en SENCOMLNG

- La figura 4 detalla un ejemplo en SENCOMLNG de la solicitud de encender una bombilla, operación codificado con el valor 3, en el primer sensor:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <lpv1>
  - <header>
      <msgID>0x3A61BB34</msgID>
      <srcID>0x2742389798</srcID>
      <dstID>0x2742389772</dstID>
      <tstamp>979551568</tstamp>
    </header>
  - <setData>
      <sensorID>0x1</sensorID>
    - <data>
        <dataCat>0x00000311</dataCat>
        <data1Values>0x1</data1Values>
      </data>
    </setData>
  </lpv1>
```

4. Ejemplo de Set en SENCOMLNG

- La figura 5 muestra una solicitud a la red para que devuelva cada 1000 milisegundos los valores de medir la luz ambiental cada 200 milisegundos:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <lpv1>
  - <header>
      <msgID>0x3EF25772</msgID>
      <srcID>0x1045921068</srcID>
      <dstID>0x2742389772</dstID>
      <tstamp>1056071393</tstamp>
    </header>
  - <monitorStart>
      <sensorID>0x1</sensorID>
    - <data>
        <dataCat>0x00000700</dataCat>
      </data>
      <measurePeriod>200</measurePeriod>
      <reportPeriod>1000</reportPeriod>
    </monitorStart>
  </lpv1>
```

5. Ejemplo de Monitoring en SENCOMLNG

- Por último, la imagen 6 muestra una petición de recibir una notificación desde la red si en el segundo sensor se sobrepasan los 40 grados centígrados (0x28). La operaciones de temperatura están codificadas con el valor 2:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <lpv1>
  - <header>
      <msgID>0x3EF25772</msgID>
      <srcID>0x1045921068</srcID>
      <dstID>0x2742389772</dstID>
      <tstamp>1056216161</tstamp>
    </header>
  - <eventSet>
      <sensorID>0x2</sensorID>
      <op>0x3</op>
    - <thresholdData>
        <dataCat>0x00000201</dataCat>
        <int32Values>0x28</int32Values>
      </thresholdData>
    </eventSet>
  </lpv1>
```

6. Ejemplo de eventSet en SENCOMLNG

4.2 eXtensible Binary Encoding (XBE32)

XBE32 (*eXtensible Binary Encoding*) [1] es una codificación binaria de SENCOMLNG. Se crea con el objetivo de minimizar el consumo de ancho de banda en las comunicaciones entre redes de sensores inalámbricos sin perder la flexibilidad del lenguaje SENCOMLNG.

XBE32 almacena información jerárquica, conservando la idea de estructuración de XML.

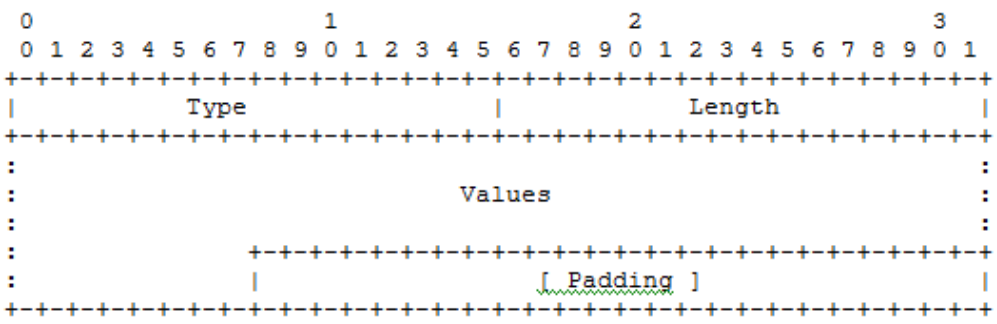
Sus elementos están codificados en binario dentro de estructuras TLV (tipo, longitud, valor) alineadas en palabras de 32 bits. XBE32 no es una codificación binaria de documentos XML sino una serialización del mismo para protocolos ligeros.

Para poder representar las información almacenada en el XML que queramos codificar, se definen dos tipos de elementos XBE32: elementos atributo y elementos complejos. Los atributos almacenarán tipos primitivos de datos mientras que los elementos complejos contendrán otros atributos XBE32 u otras estructuras complejas.

XBE32 utiliza los tipos de datos más comunes. Los elementos tipo atributo puede ser Strings, booleanos, enteros, valores en coma flotante o arrays. Además, mediante el tipo de datos binary opaque value se pueden codificar otros tipos de datos.

El cuanto a la trama TLV, el campo tipo (type) tiene una longitud de 16 bits y describe las reglas de procesado de la trama. El tipo de datos que se transporta está almacenado en el campo Values. Finalmente, el campo Length indica la longitud de la información transportada.

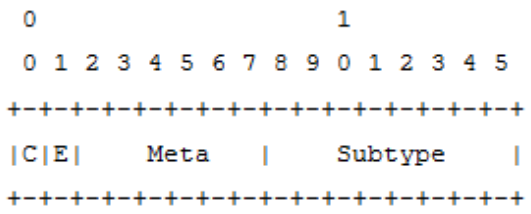
El formato de la trama es el siguiente:



7. Formato trama XBE32

Los campos se interpretan del siguiente modo:

- Type (16 bits): describe las reglas de procesado y que tipo de datos habrá dentro de values. Su estructura es la siguiente:



8. Campo Type en trama XBE32

  - Los bits C y E especifican las medidas a tomar si el valor de Type es desconocido.
  - El subcampo meta: ocupa 6 bits y describe la estructura interna de Values y el tipo de datos primitivos. Todos los tipos en meta tienen una estructura definida salvo los de tipo opaco. La X del tipo opaco de datos indicar la longitud del tipo de datos. Las estructuras definidas son:

Valor de Type.meta	Estructura de TLV values
0x00-0x1F	TLVS múltiples de longitud variable
0x20	Un único valor opaco de longitud variable
0x21	Un único string de longitud variable
0x24	Varios campos values de tipo opaque1
0x25	Varios campos values de tipo int8
0x26	Varios campos values de tipo booleano
0x28	Varios campos values de tipo opaque2
0x29	Varios campos values de tipo int16
0x2C	Varios campos values de tipo opaque4
0x2D	Varios campos values de tipo int32
0x2E	Varios campos values de tipo coma flotante de 32 bits
0x30	Varios campos values de tipo opaque8
0x31	Varios campos values de tipo int64
0x32	Varios campos values de tipo compa flotante de 64bits
0x34	Varios campos values de tipo opaque12
0x38	Varios campos values de tipo opaque16

Tabla 2. Estructuras definidas para el campo Values en XBE32

- El subcampo Subtype: ocupa 8 bits y junto con meta identifica el significado de este TLV o la información que almacena. Estos valores deben estar definidos por la capa superior.

- Campo Length (16 bits): este campo especifica el tamaño en octetos de la estructura TLV completa, sin contar el padding. Ha de ser mayor o igual a 4 octetos, salvo que estemos ante un TLV complejo con longitud sin determinar, cuyo valor será 0x0000.

- Campo Values: puede contener un solo valor de longitud variable, varios valores de longitud fija y otros TLVs. Para conseguir alineación a 32 bits, hasta 3 octetos de relleno pueden incluirse.

Los paquetes TLV de XBE32 tienen dos campos de 2 octetos para Tipo y Longitud. Pero, para poder ser extensible, los elementos pueden tener longitud variable e incluso soportar una longitud sin especificar para poder codificar mensajes grandes y comenzar a enviar antes de saber el tamaño total.

Las tramas pueden ser simples, si contienen datos primitivos, o complejas, si contienen otros TLVs.

4.2.1 Estructuras TLV

Se permiten varios tipos de estructuras TLV:

- TLVs complejos con TLVS internos.
- TLVs simples con un Value de longitud variable.
- TLVs simples con Values de 1 octeto.
- TLVs simples con Values de 2 octetos.
- TLVs simples con Values de 4 octetos
- TLVs simples con Values de 8 octetos.
- TLVs simples con Values de 12 octetos.
- TLVs simples con Values de 16 octetos.
- TLVs con Values de tipo opaco.
- TLVs con Value de tipo String.
- TLVs con Value de tipo booleano.
- TLVs con Value de tipo entero.

- TLVs con Value de tipo coma flotante.

4.2.2 Elementos XBE32

La representación jerárquica de datos en XML puede hacerse mediante un árbol, donde cada nodo tiene un identificador. En su codificación en XBE32, esta representación jerárquica se mantendrá y serán los nodos hoja los únicos que lleven tipos de datos primitivos.

Los nodos serán los elementos, teniendo cada uno de ellos un identificador que podrá ser binario o en otro formato legible. Los elementos complejos serán los padres de los elementos atributos.

Los elementos compactos están codificados dentro un solo TLV mientras que los elementos extensibles requieren uno o más TLVs.

4.2.2.1 Elementos compactos

Cada aplicación que utilice XBE32 puede utilizar los valores ya especificados, así los elementos compactos pueden utilizar las siguientes combinaciones de los subcampos meta y subtype:

Meta	Subtype	Descripción del TLV
0x00-0x1F	0x01-0xFE	Compact Complex Element
0x20-0x38	0x01-0xFE	Compact Attribute Element

Tabla 3. Valores de Meta y Subtype para elementos compactos

4.2.2.2 Elementos Extensibles: Nombres e Identificadores.

Un campo de dos octetos puede ser insuficiente para el tipo de datos que queramos usar, por ellos es necesario soportar también elementos extensibles.

Estos se codificarán utilizando varios TLVs, que se almacenarían dentro de un TLV complejo con un tipo Type reservado:

Meta	Subtype	Descripción del TLV
0x1F	0xFF	Extensible Complex Element
0x1F	0x00	Extensible Attribute Element

Tabla 4. Valores de Meta y Subtype para elementos extensibles

Cada elemento extensible debe tener un identificador que puede ser un solo valor opaco de 4 octetos llamado *Extensible Identifier* o un string utf-8 no vacío llamado *Extensible Name*.

Los Complex TLV que codifican Extensible Element tiene dos valores reservados para Extensible Names e identificadores:

Type	Descripción del TLV
0x21FF	Extensible Name Element
0x2CFF	Extensible Identifier Element

Tabla 5. Valores de Type para elementos extensibles

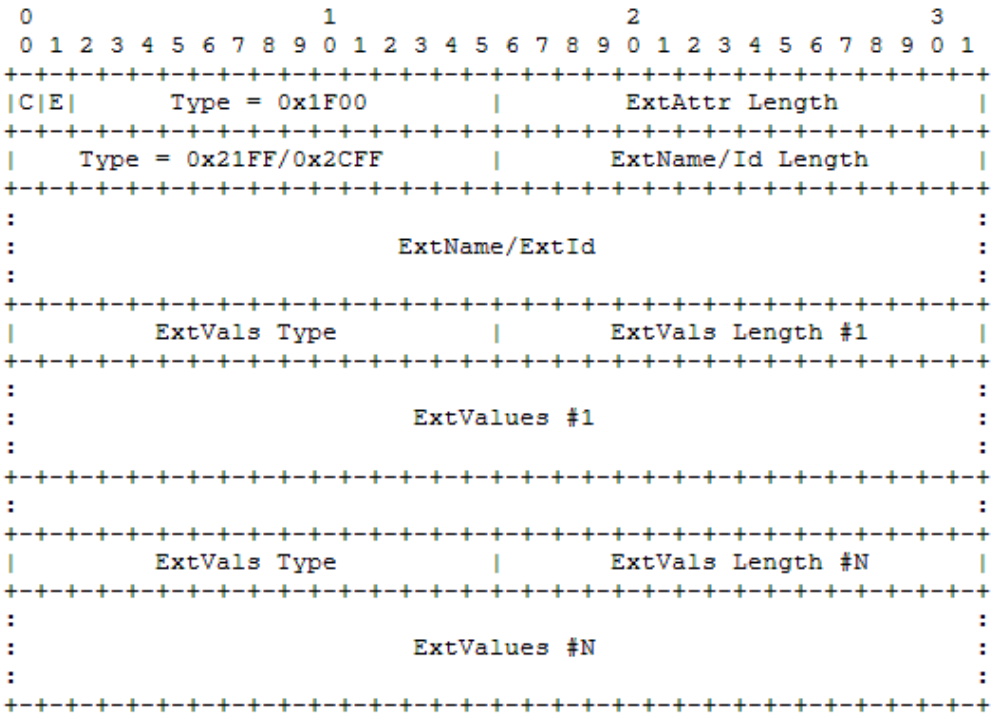
4.2.2.3 Elementos extensibles complejos

Un elemento extensible complejo está codificado dentro de un en TLV de tipo extensible complejo que contiene o un elemento de nombre extensible o un TLV de identificador extensible. Además, estará seguido de cero o más TLV codificados con elementos compactos o extensibles, atributos o complejos o alguna combinación de ellos.

4.2.2.4 Elementos extensibles de tipo atributo

La estructura de un elemento extensible de tipo atributo es similar al caso anterior: está codificado dentro un TLV de Atributo extensible que debe incluir al menos dos TLVs. Primero contendrá o un elemento de nombre extensible o un elemento de identificador extensible primeros. Estaré seguido de uno o más TLVs que contengan el campo Values correspondiente al elemento extensible en cuestión.

A continuación se incluye un ejemplo:

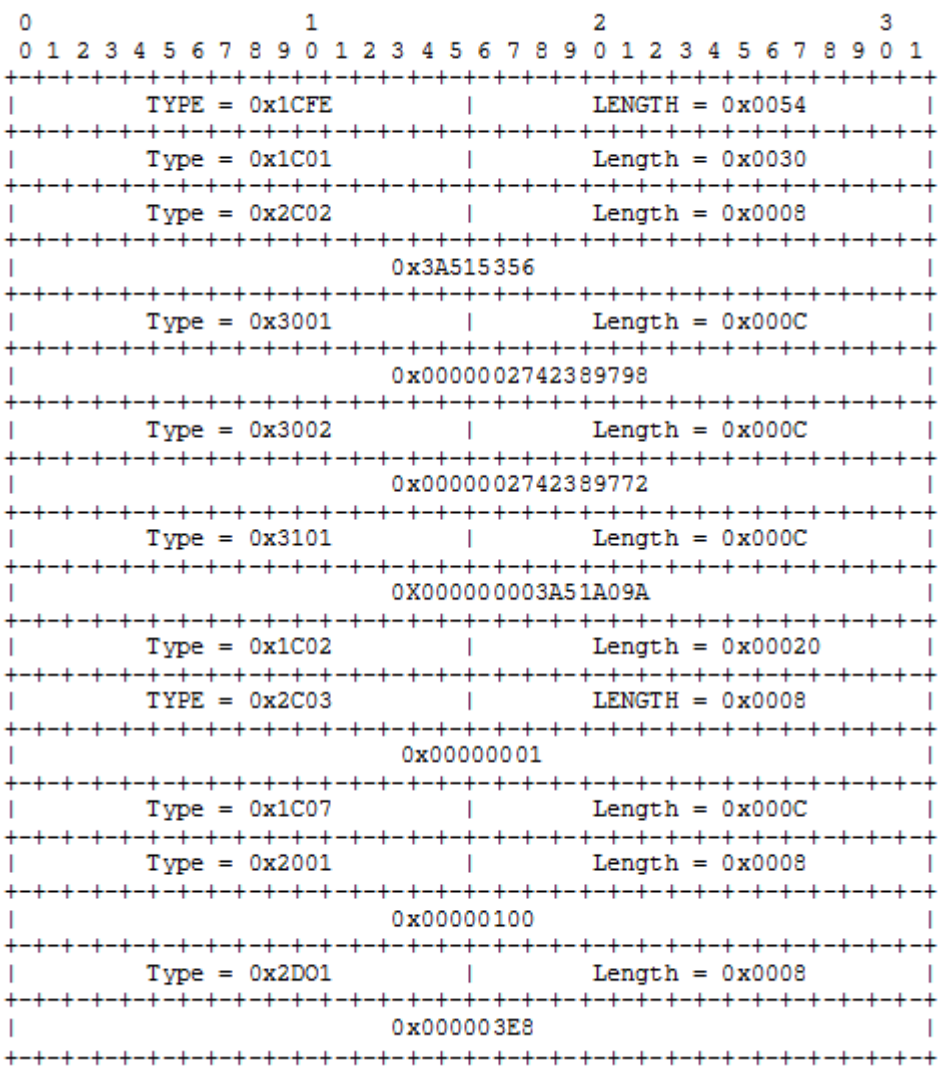


9. Ejemplo de elementos extensibles tipo atributo

4.2.3 Ejemplo de XBE32

En este apartado se muestra un ejemplo de una representación XBE32 del un fichero XML codificado con SENCOMLNG.

Empezamos mostrando el resultado XBE32 una vez pasado a un formato legible:



10. Codificación XBE32 de la operación Get

Que es la codificación correspondiente al siguiente fichero XML:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <lpv1>
  - <header>
      <msgID>0x3A515356</msgID>
      <srcID>0x2742389798</srcID>
      <dstID>0x2742389772</dstID>
      <tstamp>978428058</tstamp>
    </header>
  - <getData>
      <sensorID>0x1</sensorID>
    - <data>
        <dataCat>0x00000100</dataCat>
      </data>
      <lifetime>1000</lifetime>
    </getData>
  </lpv1>
```

11. Ejemplo de operación Get en SENCOMLNG

4.3 Light-Weight Exterior Sensornet Protocol (LWESP)

LWESP es el protocolo que permite la comunicación entre el Gateway de una red inalámbrica de sensores (WSN-GW) y una aplicación externa mediante un proceso de traducción de SENCOMLNG a XBE32.

Para realizar esta transformación será necesario un diccionario. Este diccionario es el *Lightweight_Protocol_XBE32.dict* e incluye cada una de las etiquetas contempladas por SENCOMLNG y su equivalente en binario. Este diccionario sigue el formato:

```
label = XBE32 Type
```

Todas las etiquetas de SENCOMLNG que definen elementos que contienen otras etiquetas, equivaldrán a tipos complejos de XBE32. El resto, solo contendrán valores y se corresponderán con elementos tipo atributo.

En este protocolo el que permite pasar de la pesada codificación XML a un formato más ligero apto para redes inalámbricas de sensores.

A continuación detallamos la comparación entre un documento XML y su equivalente en LWESP para poder ilustrar como funcionar LWESP.

Tomando como referencia la siguiente operación de tipo Get:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <lpv1>
  - <header>
      <msgID>0x3A468A7D</msgID>
      <srcID>0x2742389798</srcID>
      <dstID>0x2742389772</dstID>
      <tstamp>978018678</tstamp>
    </header>
  - <getData>
      <sensorID>0x1</sensorID>
    - <data>
        <dataCat>0x00000700</dataCat>
      </data>
      <lifetime>1000</lifetime>
    </getData>
  </lpv1>
```

12. Operación Get en SENCOMLNG

Su equivalente en XBE32 es el siguiente:

```
1CFE00541C0100302C0200083A468A7D3001000C00000027423897983002000C0000
0027423897723101000C000000003A4B61761C0200202C030008000000011C07000C
2C010008000007002D010008000003E8
```

13. Salida XBE32 de operación Get

Y el proceso de traducción sería el que se muestra en la imagen 14:

1CFE 0054	0x1CFE= Ipv1. La trama ocupa 84 (0x54) bytes
1C01 0030	0x1C01= header. La cabecera ocupa 48 (0x30) bytes.
2C02 0008 3A468A7D	El msg ID (0x2C02) ocupa 4 bytes y es 0x3A468A7D
3001 000C 0000002742389798	El origen de la petición (srcID=0x3001) es 0x0000002742389798 y ocupa 16 bits.
3002 000C 000002742389772	El destino de la petición (dstID=0x3002) es 0x0000002742389772y ocupa 16 bits.
3101 000C 00000003A4B6176	El momento en el que se realizó la petición (tstamp=0x3101) en hexadecimal tiene el valor 0x00000003A4B6176 y ocupa 16 bits
1C02 0020	La petición es un getData (0x1C02) y ocupa 32 bytes (0x20).
2C03 0008 00000001	El sensor al que se le realiza la petición (sensorID=0x2c03) es el primero.
1C07 000C	El campo de datos (data=0x1C07) contiene :
2C01 0008 00000700	el data category (0x2C01) al valor 7
2D01 0008 000003E8	y un tiempo de vida de la operación (lifetime= 0x2D01) de 100 ms (0x3E8).

14. Ejemplo de traducción de XB32 a SENCOMLNG

Si comparamos el tamaño de ambos documentos vemos que el documento XBE32 generado a partir de la petición ocupa 84 bytes, mientras que la petición SENCOMLNG ocupa 364 bytes. Para este caso la reducción obtenida es de 280 bytes, es decir, un 76,9%.

4.4 Sensores

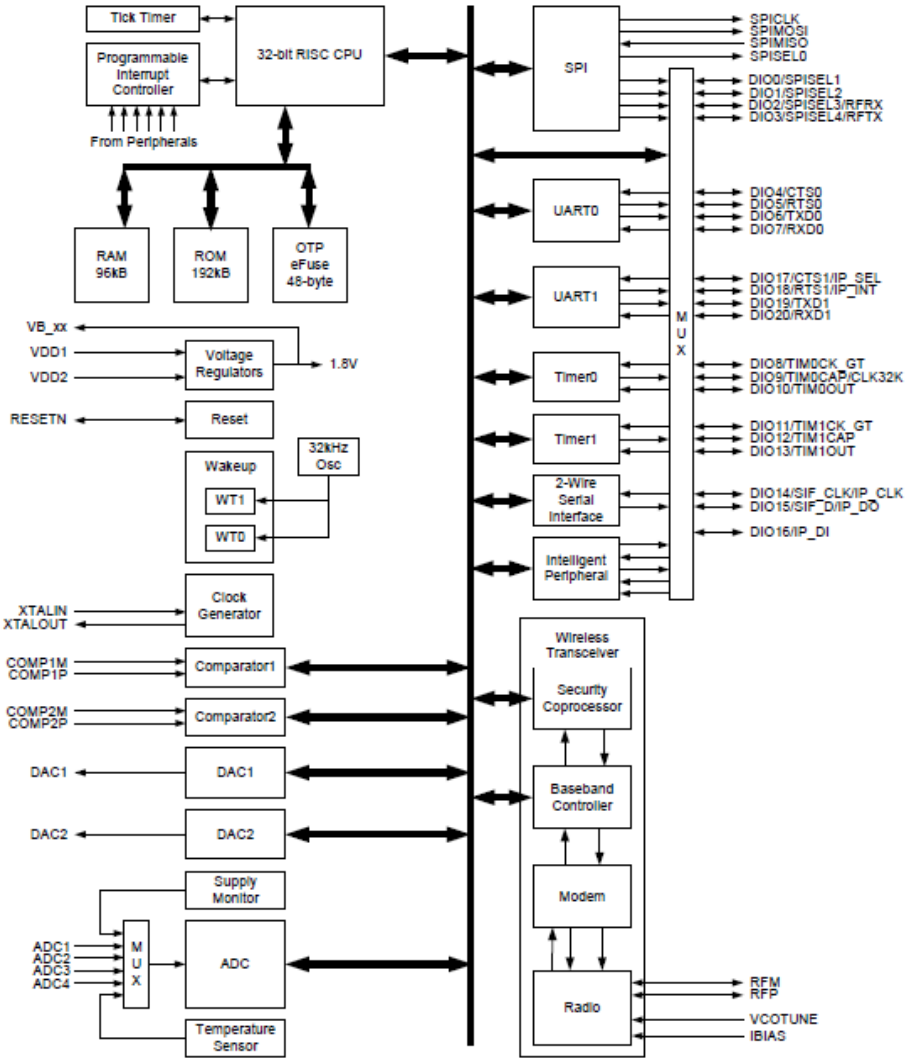
Un elemento clave una red de sensores son los sensores en sí.

Los dispositivos con los que se ha trabajado son del fabricante Jennic y son los que conforman JN5139-EK000 IEEE 802.15.4/JenNet Evaluation Kit. Este kit viene con un dispositivo más completo y cuatro dispositivos más reducidos.

Los dispositivos [24] están formados por un microprocesador, un transceptor y una serie de sensores, todos integrados en una misma placa.

El transceptor trabaja en la banda los 2.4GHz, acorde a IEEE802.15.4. La seguridad es mediante AES de 128 bits. Además soporta energía desde los 2.2V a los 3.6V.

Su sensibilidad de recepción de -97dBm y la de de transmisión es de +3dBm

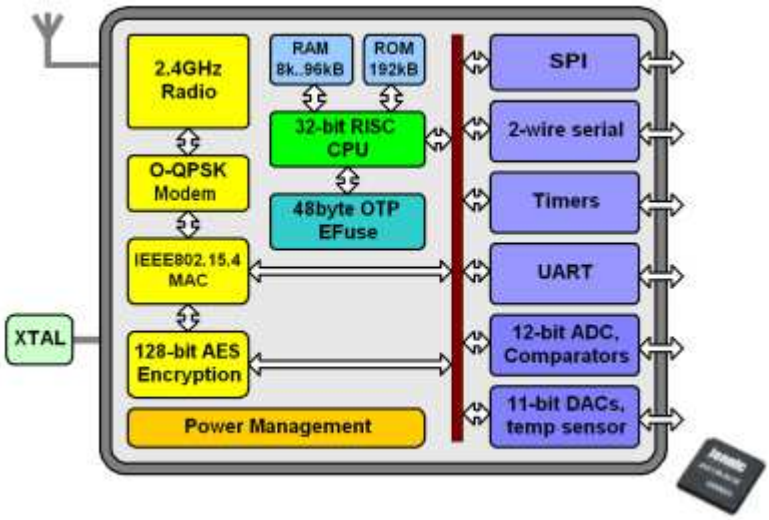


15. Diagrama de bloques del JN5139

En cuanto al microcontrolador, presenta las siguientes características:

- El procesador de 32 bits de tipo RISC, tiene 192kB ROM y 96kB RAM.
- La encriptación está basada en AES
- Presenta cuatro conversores ADC de 12 bits, dos conversores DAC de 11 bits y dos comparadores.

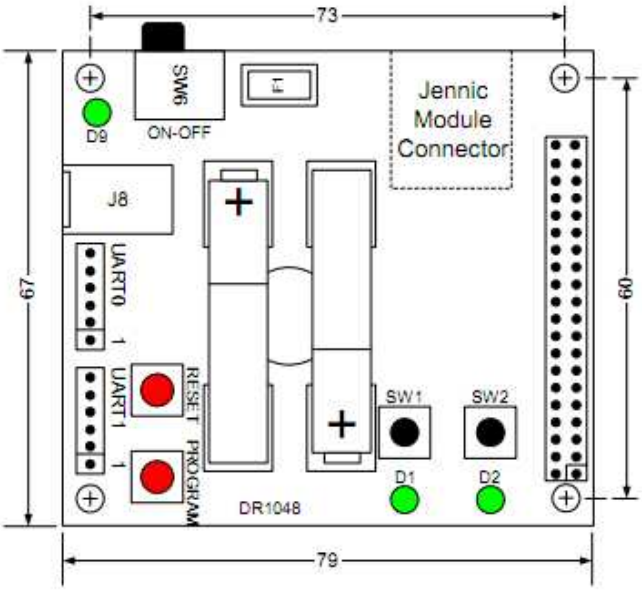
- Alberga 2 *timer* para aplicaciones y 3 *timers* para el sistema.
- También tiene 2 UARTs, un puerto SPI y interfaz serie de 2 cables
- Tiene hasta 21 GPIO



16. Estructura del microcontrolador JN5139

La red presentará dos tipos de nodos, los dispositivos finales y el nodo coordinador. El kit nos proporciona cinco dispositivos, cuatro serán iguales y uno presentará mayores características hardware.

Los cuatro nodos más pequeños llamados *sensor board* en el manual [25], tienen la estructura siguiente:



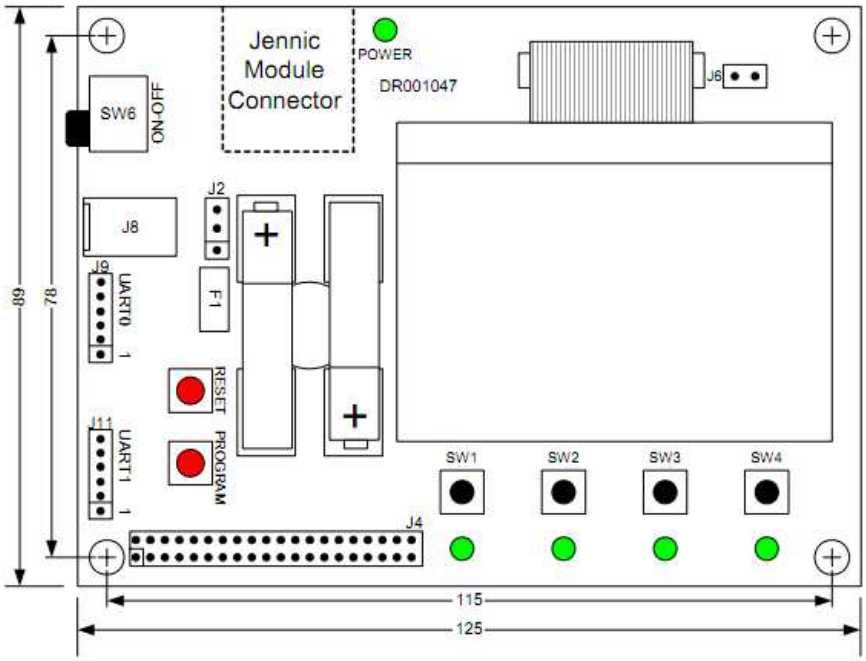
17. Estructura de un *sensor board*

Utilizan las DIOS 12, 13, 14 y 15 para su configuración. Además pueden medir los siguientes parámetros:

- Humedad, que se mide en un porcentaje de humedad relativa, de 0 a 160%.
- Temperatura, que soporta el intervalo de –40 °C a 85 °C.
- Luz ambiental se mide de forma lineal en un rango de 0 a 4015. Para medir luxes hay que medir en los dos canales que presenta y transformar a dicha unidad.

Además la placa contiene dos botones y dos LEDs.

El nodo más completo, conocido como *controller board* [26]. Presenta la siguiente estructura:



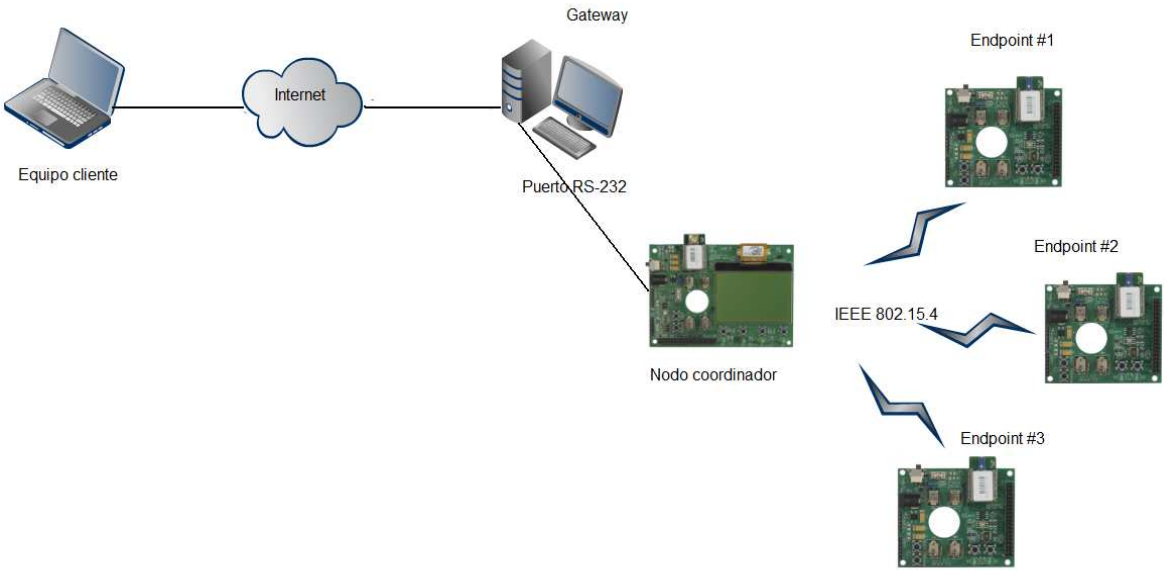
18. Estructura de un *controller board*

El coordinador, además de los mismos sensores que un *sensor board*, presenta 4 LEDs y 4 interruptores.

5 Desarrollo del proyecto

En este apartado se especifican los pasos que se han seguido para el desarrollo de la implementación del proyecto.

Antes de nada, es necesario tener una visión general de la arquitectura de la red en la que vamos a trabajar, que será así:



19. Arquitectura general de la red

En el equipo cliente crearemos una aplicación que genere las peticiones que queramos enviar a la de sensores. Para ello, mediante una interfaz gráfica se mostrarán los campos necesarios para cada tipo de operación (Get o Set). Cuando se completen los campos necesarios, la petición se almacenará en un documento codificado con una versión de XML conocida como SENCOMLNG[2].

Como un documento XML puede llegar a ocupar muchos bytes, éste será transformado a una codificación más ligera, XBE32, que consiste en asignar a cada etiqueta SENCOMLN un valor binario.

La petición codificada en XBE32 será enviada por la red y será el equipo que hace las veces de Gateway de la red inalámbrica de sensores el que la recibirá, extraerá la información y la pasará a un formato reconocible por el nodo coordinador de la red inalámbrica, iLWESP (*internal Light-Weight Sensornet Protocol*).

A continuación el nodo coordinador analizará la información recibida, decidirá si es una operación conocida, y si es así, realizará otra transformación sobre los datos de modo que ocupen aún menos para que la comunicación inalámbrica sea lo más ligera posible. Tras esto, procederá a enviar la petición al nodo final indicado en la misma.

El nodo final recibirá la petición, realizará la acción que se le indique (consultar un parámetro o encender o apagar un LED) y devolverá al nodo coordinador la respuesta correspondiente a la petición enviada. El nodo coordinador recibirá la respuesta, la transformará de nuevo a iLWESP y se la enviará mediante puerto serie al Gateway, que enviará la respuesta por la red hacia la aplicación externa.

Finalmente la aplicación externa recibirá la respuesta, generará un fichero XB32 y un fichero XML con la misma y además mostrará los datos obtenidos por pantalla.

5.1 Parte cliente

Para poder interactuar a distancia con la red de sensores se ha creado una aplicación cliente que tendrá que estar instalada en el equipo desde el que vayamos a realizar las peticiones y recibir las respuestas.

Las peticiones se crean mediante una interfaz gráfica realizada con Java Swing. Esta interfaz recoge los datos necesarios y con para generar un documento XML que sigue la estructura definida por SENCOMLNG [2].

Una vez realizado el documento XML, se codifica en XBE32. Posteriormente se enviará esta petición por la red hacia el Gateway de la red de sensores mediante una trama UDP.

También es esta parte la encargada de comprobar la recepción de ficheros respuesta, y de mostrar la información de las respuestas que vayan llegando.

5.1.1 Creación del documento XML por línea de comandos

La implementación de este proyecto comienza mediante la elaboración de una aplicación por línea de comandos sencilla que tenga la capacidad de crear documentos XML.
Para ello, se ofrece un menú con la información a rellenar, sin comprobar que la información introducida cumpla el protocolo SENCOMLNG, el único objetivo es la creación de documento XML.

Para realizar estos documentos, es necesario escoger una librería para crear y leer documentos. XML. En este caso nos encontramos antes dos alternativas claras: JDOM y SAX.
Se escoge JDOM v1.1 por estar totalmente orientado a Java.

Una vez conocido el modo de crear documentos XML siguiendo lo especificado en SENCOMLNG, se implementa la aplicación sencilla que permite formar documentos mediante la entrada de parámetros por línea de comandos.

Esta lógica es la que se incluye posteriormente en la aplicación gráfica, para poder así generar las peticiones a la red de sensores.

5.1.2 Codificación de la información y envío de cliente a Gateway

Siguiendo con la aplicación sencilla, se incluye la lógica para pasar de XML a XBE32 y después envío de esta información hacia el Gateway.
Para ello, mediante unos métodos definidos en la librería XBE32 [1] y con el diccionario que se nos proporciona (Lightweight_Protocol_XBE32.dict), realizamos la transformación a codificación XBE32 .

Una vez realizada esta traducción, se prueba el envío y recepción de XBE32 mediante

socket.

Este envío del fichero XBE32 desde el equipo cliente hasta el equipo conectado al Gateway va a ser a través de la red Internet.

Para poder comunicarse, cliente y Gateway han de establecer una conexión entre sí, esta comunicación será mediante *socket* y la implementaremos mediante la librería java.net.

Una vez comprobado que el envío se produce, y que el otro extremo recibe este datagrama, se integrará posteriormente con la interfaz gráfica.

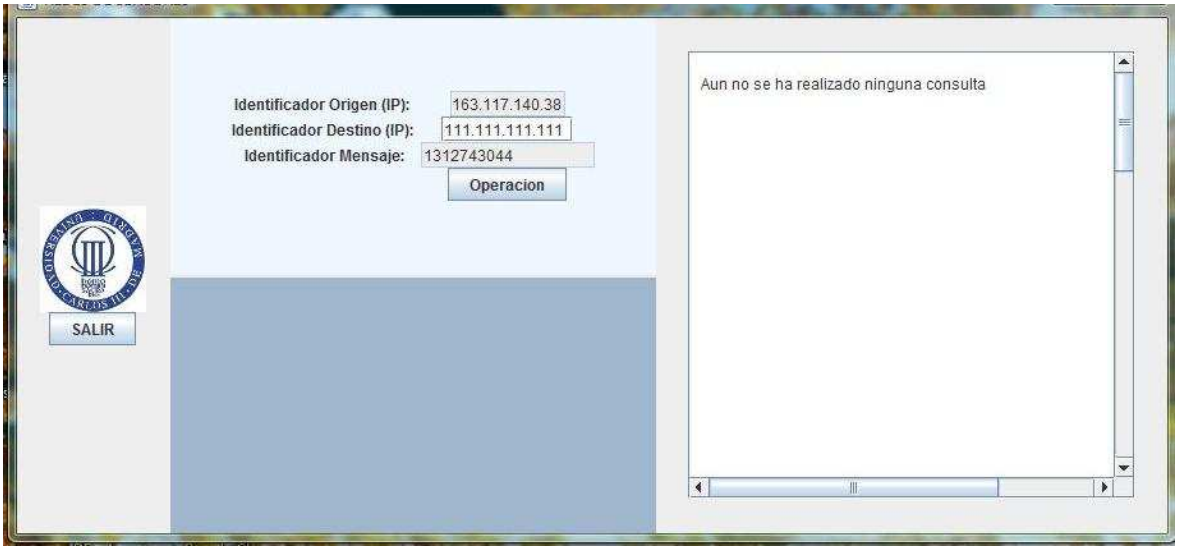
5.1.3 Creación de XML por interfaz gráfica: Swing

Una vez sabiendo que somos capaces de generar XMLs, codificarlos a XBE32 y enviarlos por la red, el siguiente paso es la interfaz gráfica.

Esta interfaz ha evolucionado en paralelo a la implementación del resto del proyecto. Está formada por tres elementos:

- Un menú a la izquierda con la opción de salir de la aplicación.
- Una parte central formada por:
  - Un parte superior que incluye la cabecera del documento que se está generando: IP destino, IP origen e identificador del mensaje.
  - Una parte inferior que será con la que se interactuará para generar peticiones. En este apartado se seleccionará la operación a realizar, se rellenarán los datos necesarios y cuando haya un documento generado, se dará la opción de enviar la petición.
- Un apartado a la derecha que irá mostrando las respuestas recibidas.

El aspecto final de la interfaz gráfica es el siguiente:



20. Pantalla inicial de la aplicación

La elección de Java Swing para realizar este paso se debe a que las primeras aproximaciones y al parseo de documentos XML y las librerías para hacerlo estaban en Java. Por lo tanto, se prefirió realizar toda la parte Cliente con la misma tecnología.

5.2 Parte de la red de sensores

En este apartado ya nos encontramos en la red de sensores.

La información la recibirá un equipo que hará las veces de Gateway y que será el que se comunique directamente con el coordinador de la red de sensores.

5.2.1 Extracción de información de trama de XBE32 a formato String.

Una vez recibido un fichero XBE32 que contenga una petición, es necesario extraer la información que almacena.
Volver a un XML es un paso atrás innecesario: lo que buscamos es minimizar el coste computacional.

Por ello, se opta por trabajar directamente a nivel de byte con el array de bytes recibido por *socket*.

Dado que ya conocemos el formato de la trama en bytes por estar definido por el protocolo XBE32, extraeremos esta información y la representaremos en una cadena de caracteres. Localizaremos en esta cadena los datos que buscamos.

Esta cadena que representa una petición habrá que enviársela más adelante al coordinador de la red de sensores, pero para ello habrá que codificarla de un modo común.

En este punto se crea un protocolo interno, al que llamaremos iLWESP. Esta codificación será la que haya entre el Gateway y el nodo coordinador. Dentro de la red inalámbrica, la comunicación será mediante una versión adaptada de iLWESP.

5.2.1.1 Interior Light-Weight Exterior Sensornet Protocol (iLWESP)

Este protocolo es interno a la red de sensores y según la petición que llegue tendrá un formato. Sus tramas están alineadas a 16 bits y todas tienen una cabecera común. Para las distintas operaciones las tramas contempladas son las siguientes:

La cabecera, común para todos los tipos de trama, seguirá el siguiente formato:

0 15 16 31

Proto **E**	IdSensor	opCode
DataCat **l**		Unidad

21. Cabecera iLWESP

- El campo Proto está reservado para operaciones de control
- El campo IdSensor especifica a qué sensor de la red está referida esta petición.
- El campo opCode especificará el tipo de operación que incluye esta trama: set, setAck, get, getReply, eventSet, MonitorStart, etc.

- El campo dataCat especifica el parámetro a medir (temperatura, humedad). Actualmente solo el último byte (el sexto) contiene el valor a medir, dado que para este protocolo es suficiente, pero se observa que el número de parámetros a medir tiene espació suficiente para aumentar.
- El campo Unidad especifica la unidad en la que se querrán obtener los datos. Esta información es para niveles superiores, que procesarán la información y la adecuaran a la unidad solicitada.

  Por ejemplo, los sensores con lo que se ha trabajado solo miden la temperatura en grados Celsius. Es la parte cliente la que transforma los datos a grados Kelvin cuando así se solicita.

Las tramas correspondientes a Get, Set y SetAck solo incluyen la cabecera.

Para el caso de Set y SetAck, el caso de dataCat y Unidad es especial: lo único que podemos estableces en el valor a encendido o apagado de un LED, por ello, el campo unidad incluirá este valor. Por ejemplo, estos dos campos valdrían 00000511 en el caso de querer encender un LED en un sensor.

La trama getReply tiene el siguiente formato:

0 15 16 31

Proto	IdSensor		opCode
DataCat			Unidad
Resultado, parte alta	Resultado, parte baja	Escala, parte alta	Escala, parte baja

22**. Trama getReply de iLWESP**

- El resultado es un número de 16 bytes, por ello viene dividido en parte más significativa y parte menos significativa.
- La escala es un número entero, pero también se divide en partes más y menos significativas.

Para un getReply de iluminación ambiental, esta trama se amplía puesto que es necesario realizar dos medidas distintas (en el canal 0 y canal 1 del sensor correspondiente) para poder presentar la unidad en luxes [28][29]. La trama getReply para este caso sería:

0		15 16	31
Proto	IdSensor		opCode
DataCat			Unidad
Resultado canal 0, parte alta	Resultado canal 0, parte baja	Escala canal 0, parte alta	Escala canal 0, parte baja
Resultado canal 1, parte alta	Resultado canal 1, parte baja	Escala canal 1, parte alta	Escala canal 1, parte baja

23. Trama getReply de luz ambiental en iLWESP

Las tramas eventSet, eventSetAck, eventStop y eventStopAck siguen el siguiente formato:

0		15 16	31
Proto	IdSensor		opCode
DataCat			Unidad
Threshold, parte alta	Threshold, parte baja	Escala, parte alta	Escala, parte baja
00	Operando	00	00

24**. Trama para eventSet, eventSetAck, eventStop y eventStopAck en iLWESP**

Se observa que es la misma trama que la getReply pero incluyendo además el campo operando, que será un valor lógico (mayor, menor, igual, etc.).
El campo resultado se sustituye por threshold, que el valor umbral que desatará el evento.

La trama eventData será la que notifique qué valor ha desatado el evento. Por ello, será una ampliación la trama eventSet:

0		15 16	31
Proto	IdSensor		opCode
DataCat			Unidad
Threshold, parte alta	Threshold, parte Baja	Escala, parte alta	Escala, parte baja
00	Operando	00	00
Dato, parte alta	Dato, parte baja	Escala, parte alta	Escala, parte baja

25**. Trama eventData en iLWESP**

Donde Dato es el valor que ha desatado el evento.

Las tramas para las operaciones de monitorización monitorStart, monitorStartAck, monitorStop y monitorStopAck son las mismas:

0		15 16	31
Proto	IdSensor		opCode
DataCat			Unidad
Período medida		Período notificación	

26**. Trama para las operaciones de monitorización en iLWESP**

- Período de medida la periodicidad con la que los sensores habrás de medir el parámetro pedido en dataCat.
- El período de notificación es el tiempo tras el cual habrá que enviar las medidas realizadas, del sensor al coordinador.

La trama correspondiente a la operación monitorData contendrá esta información , tendrá tantas medidas como quepan en la operación PerNotificacion/PerMedida y seguirá el siguiente formato:

0	15	16	31
Proto	IdSensor		opCode
DataCat			Unidad
Período medida		Período notificación	
Medida 1, parte alta	Medida 1, parte baja	Escala, parte alta	Escala, parte baja
Medida 2, parte alta	Medida 2, parte baja	Escala, parte alta	Escala, parte baja
…			
Medida n, parte alta	Medida n, parte baja	Escala, parte alta	Escala, parte baja

27**. Trama monitorData en iLWESP**

Los códigos que cada uno de las operaciones se resumen en la siguiente tabla:

opCode	**Operación**
0x01	Get
0x02	GetReply
0x03	Set
0x04	SetAck
0x05	MonitorStart
0x06	MonitorStartAck
0x07	MonitorStop
0x08	MonitorStopAck
0x09	EventSet
0x0A	EventSetAck
0x0B	EventStop
0x0C	EventStopAck
0x0D	Data
0x0E	MonitorData
0x0F	EventData
0x10	ThresholdData

Tabla 5. Códigos de operación en iLWESP

Y los valores de dataCategory son:

Parámetro	dataCategory	Unidad	
Humedad	0x000001	0x00	En porcentaje (%)
		0x01	
Temperatura	0x000002	0x00	Por defecto (ºC)
		0x01	ºC
		0x02	ºF
		0x03	K
Switch	0x000003	0x00	OFF
		0x11	ON
Luz ambiental	0x0000007	0x00	Luxes

Tabla 6. Valores de dataCategory en iLWESP

De este modo, según iLWESP, una trama de tipo Get, temperatura, en grados centígrados, al sensor 2, sería así:

0 **15** **16** **31**

00	0002	01
000005		01

28**. Ejemplo de trama Get**

Y la respuesta, suponiendo 25 grados (0x19):

0 **15** **16** **31**

00	0002		01
000005			01
00	19	00	00

29**. Ejemplo de trama GetReply**

5.2.2 Comunicación básica entre PC y Coordinador por UART.

Una vez definido el protocolo de comunicación entre el equipo Gateway el nodo coordinador, será necesario encontrar un modo de transmitir la información recibida.

La comunicación entre el equipo Gateway y el nodo coordinador es mediante puerto serie.
El modelo de comunicación es por interrupciones: en el programa Java recibe una interrupción cada vez que el coordinador le envía un carácter.
La comunicación también es por interrupciones en el nodo coordinador: cuando le llega un carácter, la UART lanza una interrupción que será tratada del modo que hayamos definido.

Para poder desarrollar esta comunicación, se ha utilizado la librería RXTX(25), que es una librería nativa que proporciona comunicación tanto serie como paralela para el Java Development Toolkit (JDK). Está almacenada es el fichero RXTXcomm.jar

Una vez implementada la lógica para esto, se inicia una el desarrollo de un modo sencillo:

- Cuando el coordinador está listo, es decir, cuando se ha iniciado y tiene al menos un *endpoint* registrado, envía por UART un mensaje de “OK” al PC.
- El PC cuando recibe una trama, tramita la información, la pasa a iLWESP y se la envía por UART al nodo coordinador.
- El nodo coordinador, al recibir esta cadena, transforma estos caracteres a los valores numéricos que representan y la envía al único nodo que tiene en este momento.

5.2.3 Comunicación dentro de la red inalámbrica

Los sensores se programan en un lenguaje similar a C. Es necesario recordar que en las redes inalámbricas de sensores se busca reducir el consumo energético a costa de perder potencial computacional. Por ello, si bien la programación es similar al

desarrollo de aplicaciones con C, ciertas limitaciones surgen como, por ejemplo, la falta de ciertas librerías adicionales para el manejo de strings.

La información en los sensores se recibe puerto serie y si es un formato esperado, es decir, si cumple iLWESP se trata. Para ello, se almacena en estructuras que serán versiones reducidas de iLWESP. Por motivos de sencillez, se va a coger solo aquel carácter que representa el valor numérico que necesitamos. De esto modo, en la comunicación conseguimos aún una mayor reducción.

La cabecera estará definida del siguiente modo:

```
Struct{
Uint8 proto;
Uint16 sensorID;
Uint8 dataCat;
Uint8 unidad;
}header
```

30**. Estructura de la cabecera en la red se sensores**

Se observa que estos campos ocupan menos que la información codificada en iLWESP. Por ahora no es necesario extraer más información, por ello, en la trama a enviar se mete exclusivamente el byte del string que contiene el número.
Estos datos se almacenan en un array de unsigned int de 8 bits cada campo.

Para ilustrar cómo se rellena este array, siguiendo la trama ejemplo anterior, mostramos como se rellena este array. El string recibido por puerto serie sería “0000020100000301” y la trama a enviar desde el coordinador tendría los siguientes valores:

```
tramaEnvio[0]=0 //numero representado en pos 2
tramaEnvio[1]=0 //numero representado en pos 5
tramaEnvio[2]=2 //numero representado en pos 6
tramaEnvio[3]=1 //numero representado en pos 8
tramaEnvio[4]=0 //numero representado en pos 12
tramaEnvio[5]=0 //numero representado en pos 13
tramaEnvio[6]=3 //numero representado en pos 14
tramaEnvio[7]=1 //numero representado en pos 16
```

El nodo sensor, una vez recibida está trama, mirará el valor correspondiente al código de operación y con esta información junto con el parámetro solicitado, hará lo correspondiente: bien solicitará un dato, bien encenderá o apagará un LED.

Para ello activará el método necesario dentro del API de Jennic 26 [2] para consultar estos datos, por ejemplo, para un caso de solicitar la humedad:

```
vHTSstartReadHumidity();
lecturaValor=u16HTSreadHumidityResult();
```

32. Ejemplo de código en endpoint

Una vez realizada la operación, se rellenarán los datos en la estructura correspondiente al tipo de trama que está creando y enviará esta trama respuesta al coordinador.

El coordinador leerá estos datos y los pasará a un formato adecuado para la comunicación por UART, es decir, a iLWESP.

Finalmente, se enviará este string al equipo Gateway.

5.2.4 Envío de la información desde la red hasta el cliente

El último paso será labor del equipo Gateway.

Cuando recibe una respuesta vía UART desde el nodo coordinador, saltará una interrupción en el programa del Gateway. Cogerá estos datos y comprobará si lo que le pasa la UART es algo conforme a iLWESP.

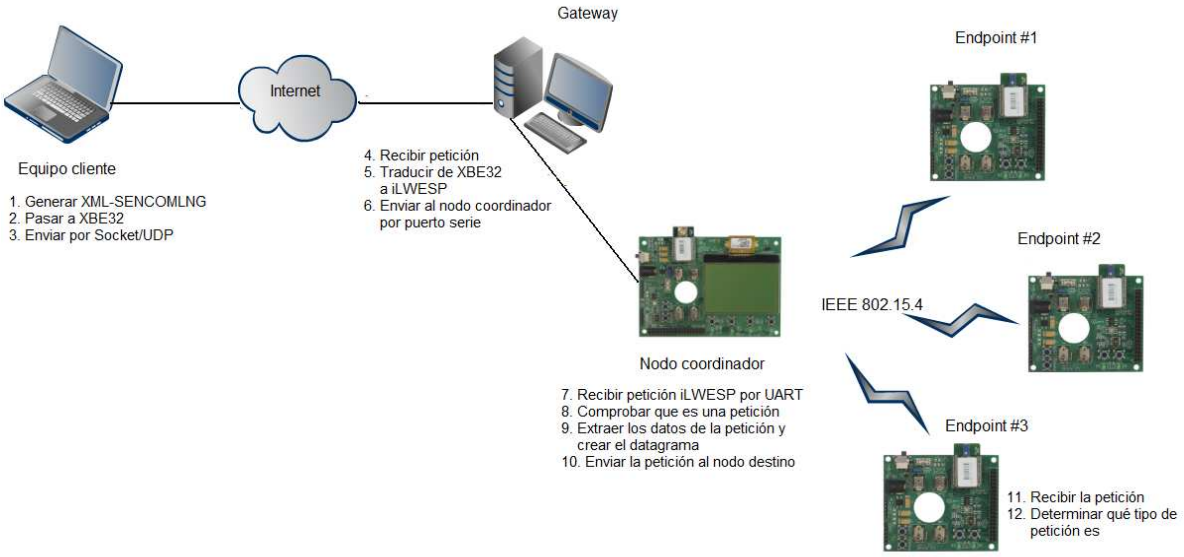
Si es así, tendrá que generar una respuesta acorde a XBE32, generar una respuesta y enviárselo al coordinador mediante un datagrama UDP. No se espera respuesta.

La recepción de datagramas en ambos extremos es mediante *polling*.

5.3 Arquitectura detallada de la red

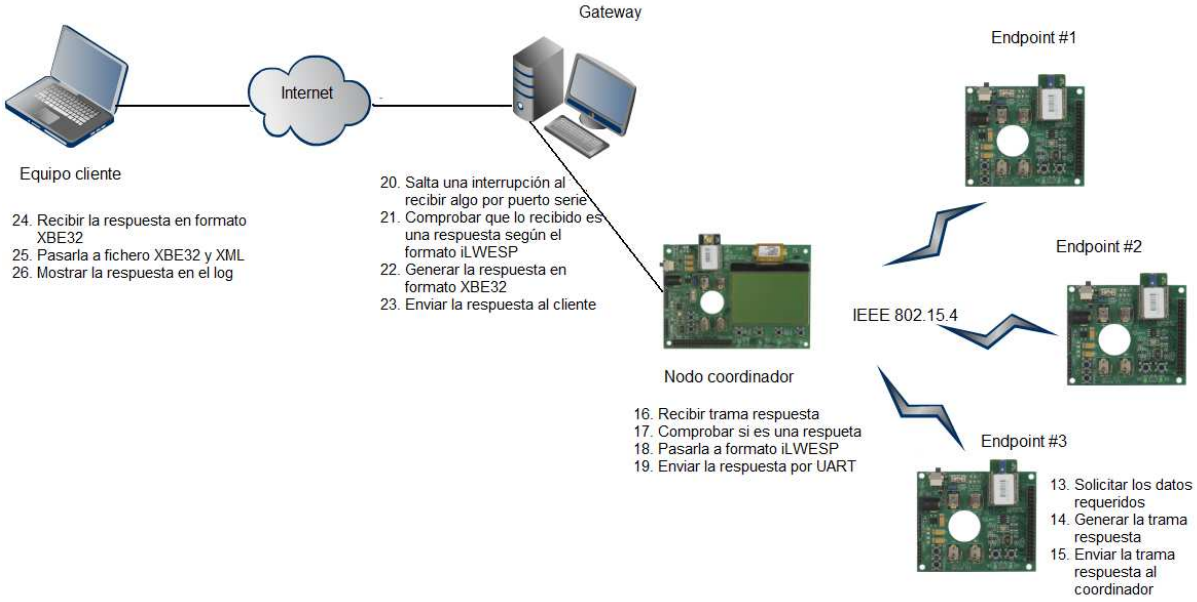
Una vez explicado los procesos que ocurren a lo largo de la aplicación, la arquitectura detallada de los pasos que se realizan se ilustra en las siguientes dos imágenes:

Inicialmente se ilustra el proceso de petición:



33. Proceso de petición

El proceso de respuesta sería en dirección contraria, desde el *endpoint* hasta el nodo equipo cliente. Se ilustra en la imagen 34:



34. Proceso de respuesta

6 Funcionamiento detallado de la solución

En este apartado se explicará detalladamente cómo funciona la aplicación mediante un ejemplo de una operación Get en la que el parámetro solicitado es la luz ambiental seguida de otra operación de Set.

6.1 Primer paso: descargar el programa en los sensores

Una vez programados y compilados los programas que irán embebidos en los dispositivos de la red de sensores, será necesario descargarlos a los mismos. Para ello habrá que poner el microcontrolador en modo programación, esto se consigue:

- Pulsando el botón de *program* (SW7)
- Pulsando el botón de *reset* (SW5)
- Soltando el botón de *reset*
- Soltando el botón de *program*

Una vez en modo programación, el modo de descargar los programas será mediante el programa *JN-SW-4007 Flash Programmer* que se puede encontrar en la página de Jennic y que además está incluido dentro del *Development Kit*.

Para ello, primero habrá que detectar el microcontrolador en el puerto COM en el que esté. Una vez hecho esto, habrá que seleccionar el archivo a descargar y darle a programación.

6.2 Segundo paso: iniciar los programas

Para iniciar la red de sensores, en el Gateway habrá que iniciar el programa de comunicación con el nodo coordinador.

Además habrá que pulsar el botón reset en cada uno de los nodos, primero en los en *endpoints* y finalmente en el coordinador de la red.

El nodo coordinador mandará un mensaje al Gateway. Este mensaje será “OKX”, donde X es el número de nodos registrados en la red. Por ejemplo, si hay dos nodos en la red, el mensaje será “OK2”.

6.3 Tercer paso: realizar peticiones a la red de sensores

Una vez tenemos la red de sensores corriendo, ya podemos empezar a interactuar con ella.

El primer paso será lanzar la aplicación cliente que permite realizar esto. Es importante detallar que la aplicación gráfica con la que trabajaremos no es el objetivo de este proyecto, y por ello su interfaz es sencilla. Lo que buscamos es poder mostrar un log de las peticiones realizadas mientras la aplicación cliente esté en ejecución.

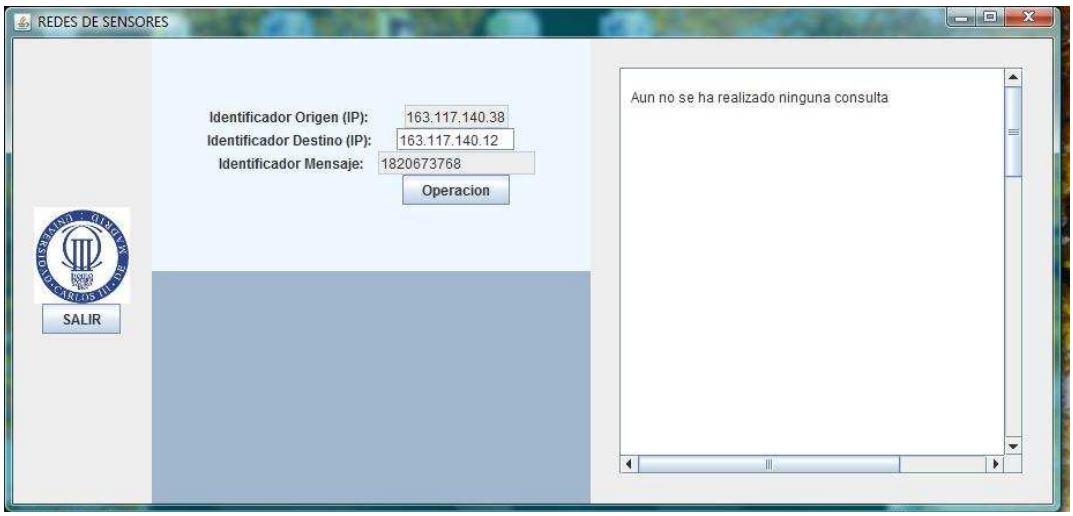
Las peticiones se pueden realizar de modo consecutivo y las respuestas se irán refrescando cada 5 segundos, sin embargo es necesario tener en cuenta que los microcontroladores en los sensores tienen un cierto retardo y que el programa se ha diseñado que hasta que los *endpoints* no hayan terminado de enviar la respuesta, no se realice ninguna otra petición.

Además, en ocasiones los *endpoints* mandan al coordinador dos veces seguidas la misma respuesta, por lo que el log muestra dos veces la misma petición. Como ambas respuestas tienen el mismo identificador de mensaje, no será difícil diferenciar cuando ha ocurrido esto.

Para ilustrar el funcionamiento de la aplicación, se detallará paso a paso cómo se realiza una operación tipo Get, seguida de una operación set en un Led.

6.3.1 Ejemplo de operación Get

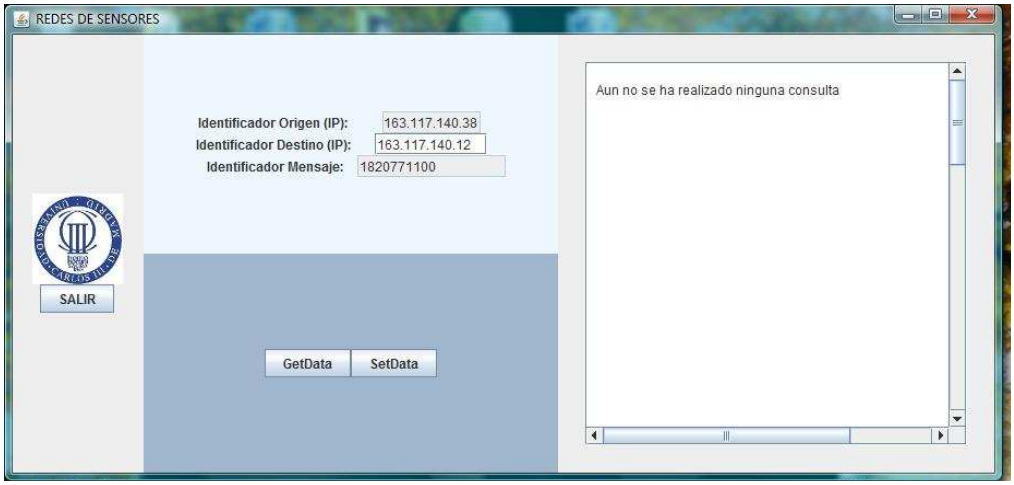
Al lanzar la aplicación cliente nos encontraremos con esta pantalla:



35. Ventana principal de la aplicación

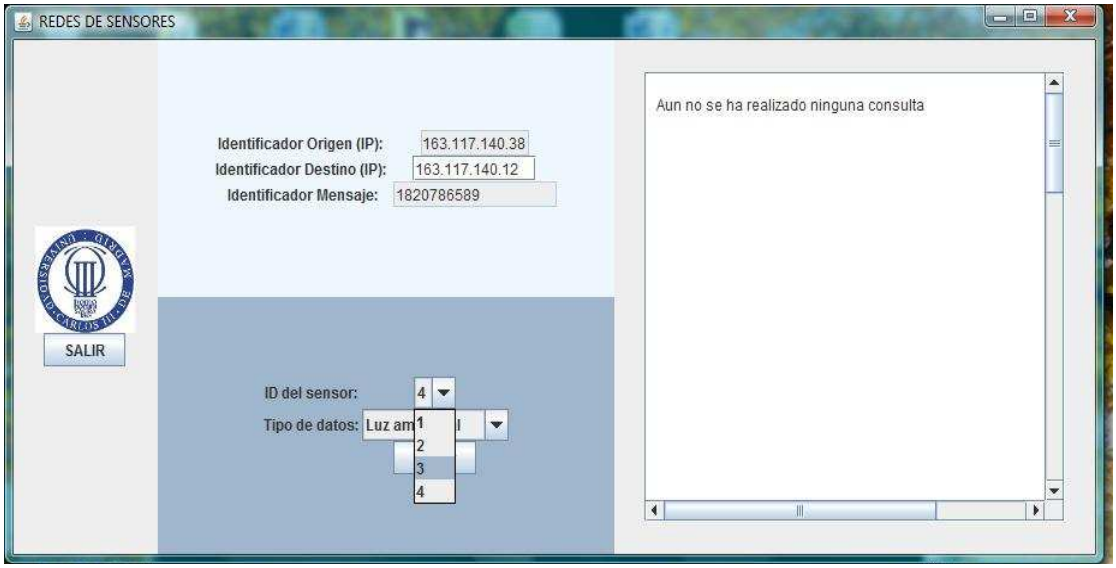
La dirección destino es un campo editable, y será la dirección del equipo Gateway de la red de sensores. El identificador del mensaje es un valor derivado de la fecha en el momento de crear dicha petición.

Una vez introducida la dirección destino, habrá que pulsar el botón *Operación* para acceder a la pantalla que nos muestra las posibles operaciones a realizar:

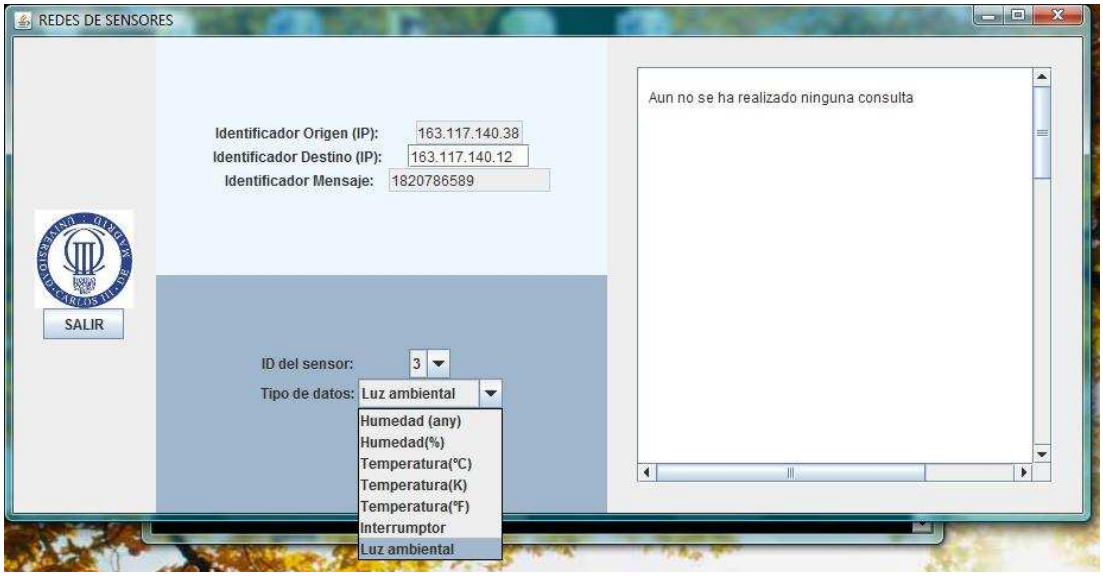


36. Pantalla de peticiones

En este caso queremos realizar una operación tipo Get, así que pulsaremos dicho botón, accediendo así a la pantalla correspondiente, y seleccionamos qué operación queremos realizar y en qué sensor:



37. Operación Get: selección de endpoint



38. Operación Get: selección de parámetro

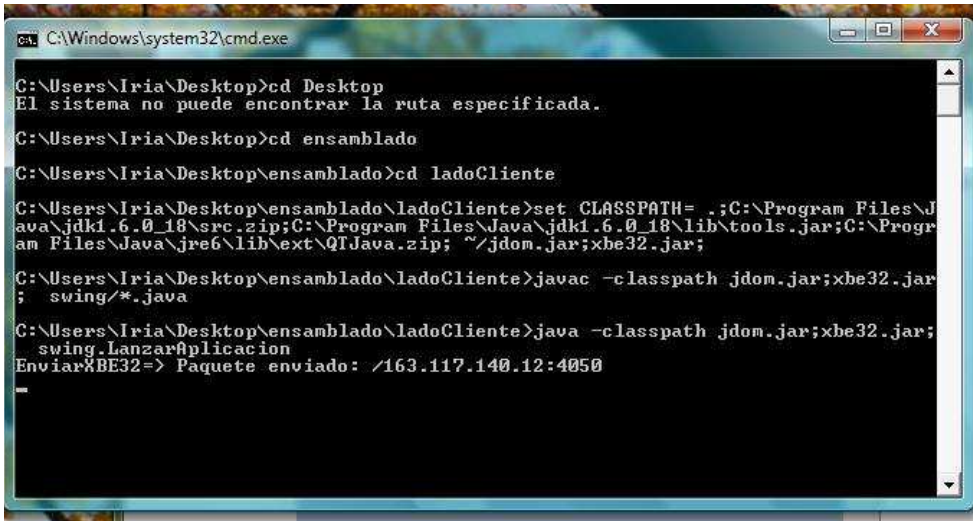
Una vez seleccionada la opción de medir luz ambiental en el tercer endpoint, pulsamos el botón enviar. Esto generará el siguiente documento en SENCOMLNG con la petición:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <lpv1>
  - <header>
      <msgID>0x6C86FF9D</msgID>
      <srcID>0x2742389798</srcID>
      <dstID>0x2742389772</dstID>
      <tstamp>1820832226</tstamp>
    </header>
  - <getData>
      <sensorID>0x3</sensorID>
    - <data>
        <dataCat>0x00000700</dataCat>
      </data>
      <lifetime>1000</lifetime>
    </getData>
</lpv1>
```

39. Petición SENCOMLNG con petición Get

Podemos observar que el código de operación es 0x00000700, donde el 7 corresponde al parámetro luz ambiental. El *timestamp* que se muestra es el del momento de la generación del fichero en SENCOMLNG.

Una vez generada esta petición, se transformara en un fichero XBE32 y se enviará por la red mediante datagrama UDP al equipo GW. Este envío podremos apreciarlo mediante las trazas de la aplicación:



40. Trazas de envío de petición desde cliente

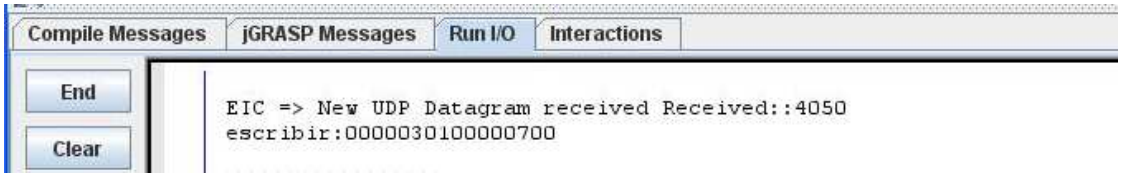
El Gateway recibirá esta petición y la tratará del siguiente modo:

- Transformara el array de bytes a un string
- Lo pasará a iLWESP
- Enviará la petición en formato iLWESP por UART al nodo coordinador:

0	**15** **16**	**31**
0x00	0x0003 (sensor nº 3)	0x01 (get)
0x000007 (luz ambiental)		0x00

41. Petición Get en iLWESP

Que se reflejará en las trazas del Gateway del siguiente modo:



42.Trazas de la petición Get en el Gateway

En el nodo coordinador, saltará una interrupción cada vez que se reciba algo por UART. Por ello, cuando salte esta interrupción será necesario comprobar que lo recibido sea realmente una petición de iLWESP. Si es así, se enviará la petición al *endpoint* indicado mediante el método VTxDataFrame:

```
PUBLIC void Uart0ServiceInterrupt(uint32 u32Device, uint32 u32ItemBitmap)
{
    char *pcstring=0;

   /* Lee el caracter recibido por la UART*/
    uint8 a=u8AHI_UartReadData(E_AHI_UART_0); //devuelve byte

     //comprueba que sea un caracter valido
    if( (a>='0'&& a<='9') || (a>='A'&& a<='Z') || (a>='a'&& a<='z') || a ==' ' || a =='\r'){
        pcstring=&a;
     }

    while (*pcstring)
    {
     if( (*pcstring>='0'&& *pcstring<='9') ||  (a>='A'&& a<='F') ){//si es un caracter hexadecimal
            holaRv[posRv]=(char)*pcstring;//lo almacena
            posRv++;
       }
      *pcstring++;
    }

    if(posRv >=16  ){//tamaño de la trama  Request en miProto
        aEnviar= holaRv;
        vPrintStringLn(aEnviar);
        vTxDataFrame(0, aEnviar);//enviar peticion
        posRv=0;
    }

   }
```

43. Coordinador: código que procesa la recepción por UART

El método vTxDataFrame rellenará las estructuras definidas y enviará al nodo en cuestión la petición solicitada:

```
cabecera.proto=(uint8)valorReal((char)tramaAEnviar[1]);
cabecera.sensorID= (uint8)valorReal((char)tramaAEnviar[5]);
cabecera.opCode=(uint8)valorReal((char)tramaAEnviar[7]);
cabecera.dataCat= (uint8)valorReal((char)tramaAEnviar[13]);
cabecera.dataCatUnidad=(uint8)valorReal((char)tramaAEnviar[15]);
//segun el tipo de request
switch(cabecera.opCode){
        case 1:case 3:{//get, set
            longitudTrama=8;
            uint8 tramaEnvio[longitudTrama];
            //rellenamos el paquete inalámbrico a enviar
            tramaEnvio[0]= (uint8)cabecera.proto;
            tramaEnvio[1]=(uint8)((cabecera.sensorID)>>8);
            tramaEnvio[2]= (uint8)(cabecera.sensorID);
            tramaEnvio[3]= (uint8) cabecera.opCode;
            tramaEnvio[4]= (uint8)(cabecera.dataCat>>16);
            tramaEnvio[5]= (uint8)(cabecera.dataCat>>8);
            tramaEnvio[6]= (uint8)cabecera.dataCat;
            tramaEnvio[7]= (uint8)cabecera.dataCatUnidad;

            int i=0;
            //copiamos al payload
               while(i<longitudTrama){
                pu8Payload[i]=tramaEnvio[i];
                i++;
            }
            sMcpsReqRsp.uParam.sReqData.sFrame.u8SduLength = longitudTrama;
            //limpiamos el array
            int limpiarTrama=0;
            for(limpiarTrama=0; limpiarTrama<24; limpiarTrama++) tramaEnvio[limpiarTrama]=0;
            break;
        }
        default:
         //   para futuras operaciones
        break;
    }

    vAppApiMcpsRequest(&sMcpsReqRsp, &sMcpsSyncCfm);//peticion de envio a la pila
```

44. Coordinador: código preparación trama Request

Por ello, el array tramaEnvio copiaría en payload los siguientes valores:

- tramaEnvio[0]=0x00;
- tramaEnvio[1]=0x00;
- tramaEnvio[2]=0x03;
- tramaEnvio[3]=0x01;
- tramaEnvio[4]=0x00;
- tramaEnvio[5]=0x00;

- tramaEnvio[6]=0x07;

- tramaEnvio[7]=0x00;

Cuando el nodo especificado reciba la petición, discriminará si lo recibido es un paquete de datos. De ser así, dentro del método vProcessIncomingData interpretará los campos del payload y según ello realizará la operación necesaria:

```
uint16 resultadoGet= (uint16)getData( tramaGetReply.cabecera.opCode, cabecera.dataCat);
tramaGetReply.resulHigh=(uint8)(resultadoGet>>8);
tramaGetReply.resulLow=(uint8)(resultadoGet);
tramaGetReply.escalaHigh=(uint8)0;
tramaGetReply.escalaLow=(uint8)0;
int longitudTrama=12;
if (cabecera.dataCat==7) longitudTrama=16;
uint8 cadenaGetReply[longitudTrama];
cadenaGetReply[0]=(uint8)tramaGetReply.cabecera.proto;
cadenaGetReply[1]=(uint8)((cabecera.sensorID)>>8);
cadenaGetReply[2]=(uint8)(cabecera.sensorID);
cadenaGetReply[3]= (uint8)tramaGetReply.cabecera.opCode;
cadenaGetReply[4]=(uint8)((cabecera.dataCat)>>24);
cadenaGetReply[5]=(uint8)((cabecera.dataCat)>>16);
cadenaGetReply[6]=(uint8)((cabecera.dataCat));
if(cabecera.dataCat!=3) cadenaGetReply[7]=(uint8)tramaGetReply.cabecera.dataCatUnidad;
else if(cabecera.dataCat==3){
    if(bLightBulbState1)  cadenaGetReply[7]=0x11;
    else   cadenaGetReply[7]=0x00;
}
cadenaGetReply[8]=(uint8)tramaGetReply.resulHigh;
cadenaGetReply[9]=(uint8)tramaGetReply.resulLow;
cadenaGetReply[10]=(uint8)tramaGetReply.escalaHigh;
cadenaGetReply[11]=(uint8)tramaGetReply.escalaLow;

if (cabecera.dataCat==7){
    vALSreset();//
    vALSstartReadChannel(1);
    uint16 valorIR= (uint16)u16ALSreadChannelResult();
    cadenaGetReply[12]=(uint8)(valorIR>>8);
    cadenaGetReply[13]=(uint8)(valorIR);
    cadenaGetReply[14]=(uint8)tramaGetReply.escalaHigh;
    cadenaGetReply[15]=(uint8)tramaGetReply.escalaLow;
}

vTxDataFrame(0x01, cadenaGetReply);
```

```
PUBLIC uint16 getData(uint8 opCode, uint8 dataCat){
    uint16 lecturaValor=-1;

    switch(opCode){

        case 0x02:{//GETReply

        switch(dataCat){
                case 0x01: {//getHumidity
                    vHTSstartReadHumidity();
                    lecturaValor=u16HTSreadHumidityResult();//bloqueante
                    break;
                }
                case 0x02:{//getTemperature
                    vHTSstartReadTemp();
                    lecturaValor=u16HTSreadTempResult();//bloqueante
                    break;
                }
                case 0x03:{//switch
                    if(bLightBulbState1)  lecturaValor=0x11;
                    else lecturaValor=0x00;
                    break;
                }
                case 0x07:{//getAmbientLigth
                    vALSstartReadChannel(0);//uno de los canales
                    lecturaValor=u16ALSreadChannelResult();
                    break;
                }
                default:{
                    break;
                }
            }//switch dataCat
            break;
        }

        case 0x04:{//SETReply
            lecturaValor=0;//
```

45. Coordinador: código de Request y Reply

Los valores de los campos de la trama respuesta serán los siguientes:

- cadenaGetReply[0]=0x00;

- cadenaGetReply [1]=0x00;

- cadenaGetReply [2]=0x03;

- cadenaGetReply [3]=0x02; (operación getReply)

- cadenaGetReply [4]=0x00;
- cadenaGetReply [5]=0x00;
- cadenaGetReply [6]=0x07;
- cadenaGetReply [7]=0x00;
- cadenaGetReply [8]=0x01;
- cadenaGetReply [9]=0x27;
- cadenaGetReply [10]=0x00;
- cadenaGetReply [11]=0x00;
- cadenaGetReply [12]=0x00;
- cadenaGetReply [13]=0x2A;
- cadenaGetReply [14]=0x00;
- cadenaGetReply [15]=0x00;

Una vez el *endpoint* envíe la respuesta, el nodo coordinador la recibirá y mediante el método vProcessIncomingData la tramitará. Para ello, tendrá que transformarla a formato iLWESP, antes de enviarla por la UART, esto lo haré mediante el método generarSalida:

```
if ((u16NodeAddr == sCoordData.sNode.asAssocNodes[0].u16ShortAddr)|| (u16NodeAddr == sCoordData.sNode.asAssocNodes[1].u16ShortAddr))
{
   int opCodeRv=(psFrame->au8Sdu[8]);
   if (opCodeRv !=0){
       char *resultado=0;

       cabecera.proto=(psFrame->au8Sdu[0]);
       uint16 sensorIDIn=(uint16)u16NodeAddr;
       if(sCoordData.sNode.asAssocNodes[0].u16ShortAddr==sensorIDIn) {
          cabecera.sensorID=1;
       }
       else if(sCoordData.sNode.asAssocNodes[1].u16ShortAddr==sensorIDIn){
           cabecera.sensorID=2;
       }else cabecera.sensorID=1;

        cabecera.opCode=(uint8)(opCodeRv);
        cabecera.dataCat=(uint8)(psFrame->au8Sdu[6]);
        cabecera.dataCatUnidad=(uint8)(psFrame->au8Sdu[7]);

        if(opCodeRv==2 && cabecera.dataCat!=8 ){//getReply
            tramaGetReply.cabecera.proto= cabecera.proto;
            tramaGetReply.cabecera.sensorID= cabecera.sensorID;
            tramaGetReply.cabecera.opCode= cabecera.opCode;
            tramaGetReply.cabecera.dataCat= cabecera.dataCat;
            tramaGetReply.cabecera.dataCatUnidad= cabecera.dataCatUnidad;
            tramaGetReply.resulHigh=(psFrame->au8Sdu[8]);
            tramaGetReply.resulLow=(psFrame->au8Sdu[9]);
            tramaGetReply.escalaHigh=(psFrame->au8Sdu[10]);
            tramaGetReply.escalaLow=(psFrame->au8Sdu[11]);
            if(cabecera.dataCat==7){//si es luz ambiental
                resultado= generarSalida( tramaGetReply.cabecera.proto, tramaGetReply.cabecera.sensorID, tramaGetReply.cabecera.opCode, tramaGetReply.cabecera.
dataCat, tramaGetReply.cabecera.dataCatUnidad,  tramaGetReply.resulHigh, tramaGetReply.resulLow, tramaGetReply.escalaLow, (uint8)(psFrame->au8Sdu[12]), (uint8)(psFrame
->au8Sdu[13]), (uint8)(psFrame->au8Sdu[15]) );
            }else resultado= generarSalida( tramaGetReply.cabecera.proto, tramaGetReply.cabecera.sensorID, tramaGetReply.cabecera.opCode, tramaGetReply.cabecera
.dataCat, tramaGetReply.cabecera.dataCatUnidad,  tramaGetReply.resulHigh, tramaGetReply.resulLow, tramaGetReply.escalaLow, 0, 0, 0 );

            vPrintStringLn(resultado);//envio por uart hacia el pc
        }
```

46. Coordinador: código que tramita la respuesta recibida

```
PUBLIC char *generarSalida(int protoRv, int sensorIDRv, int opCodeRv, int dataCatRv, int dataCatUnidadRv, int resulHigRv, int resulLowRv, int dataScale, int
resulHigRv2, int resulLowRv2, int dataScale2){
    int longSalida=0;
    if(opCodeRv==2) longSalida=24;//getReply
    if(opCodeRv==4) longSalida=16;//setReply
    char stringSalida[longSalida];
    int limpiarSalida=0;
    for(limpiarSalida; limpiarSalida<24; limpiarSalida++) stringSalida[limpiarSalida]='\0';
    int posArray=0;

     char *stringProto=resultadoOperacionHex(2, (int)protoRv);
     while(*stringProto){
        stringSalida[posArray]=(*stringProto);
        stringProto++;
        posArray++;
    }
    char *stringSensorID=resultadoOperacionHex(4, (int)sensorIDRv);
     while(*stringSensorID){
        stringSalida[posArray]=*stringSensorID;
        stringSensorID++;
        posArray++;
    }
     char *opCodeRvd= resultadoOperacionHex(2, (int)opCodeRv);
      while(*opCodeRvd){
        stringSalida[posArray]=*opCodeRvd;
        opCodeRvd++;
        posArray++;
    }
    char *dataCatpre="0000";
       while(*dataCatpre){
        stringSalida[posArray]=*dataCatpre;
        dataCatpre++;
        posArray++;
    }
    char *dataCatRvd= resultadoOperacionHex(2, (int)dataCatRv);
    while(*dataCatRvd){
        stringSalida[posArray]=*dataCatRvd;
        dataCatRvd++;
```

47. Coordinador: código que genera la salida

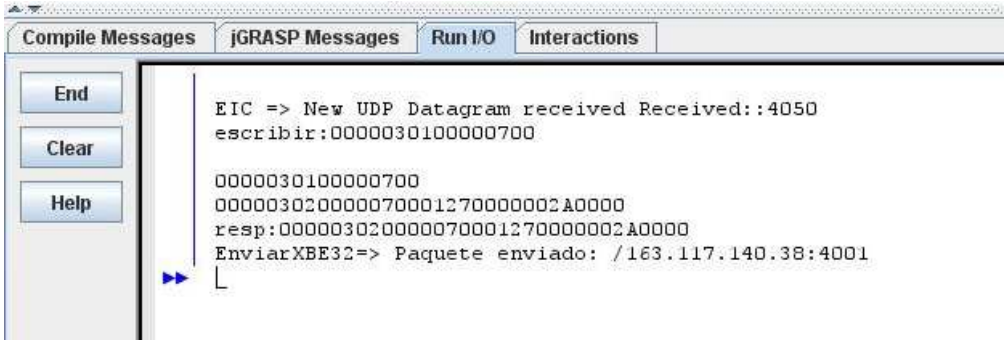
Este código genera la siguiente salida:

0		**15 16**	**31**
0x00	0x0003 (sensor nº3)		0x02(getReply)
0x000007 (luz ambiental)			0x00
0x01	0x27	0x00	0x00
0x00	0x2A	0x00	0x00

48. Respuesta de petición Get por UART

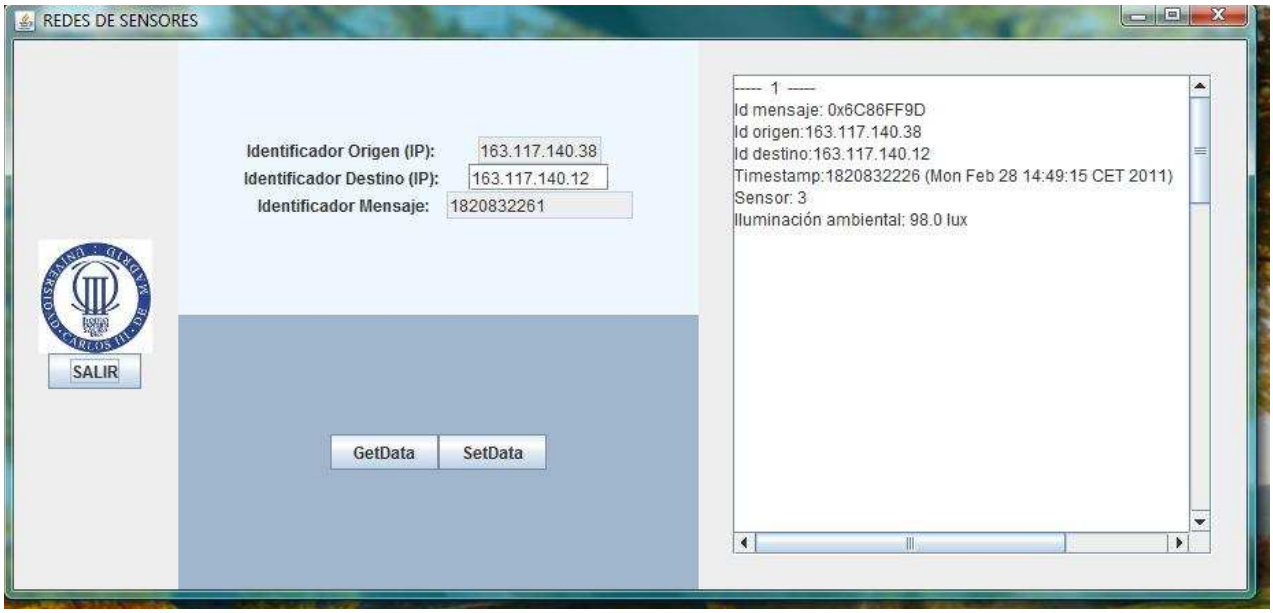
Esta repuesta se recibe en el Gateway por el puerto serie. Esto causará una interrupción. Una vez se compruebe que esta respuesta tiene un formato válido, se tramitará la respuesta, y se generará una trama XBE32 que se enviará mediante un datagrama UDP al equipo.

En el Gateway, el proceso de la recepción de la petición, el envío en formato iLWESP desde el Gateway al coordinador, la respuesta desde el coordinador al Gateway y el envío de la respuesta mediante datagrama UDP se refleja en las siguientes trazas:



49. Gateway: trazas de reply y request

Finalmente, la parte cliente recibirá esta respuesta y conforme la reciba generará un archivo XBE32 con esta respuesta. Cada 5 segundos la aplicación cliente comprobará si ha recibido respuestas, y de ser así, lo mostrará:



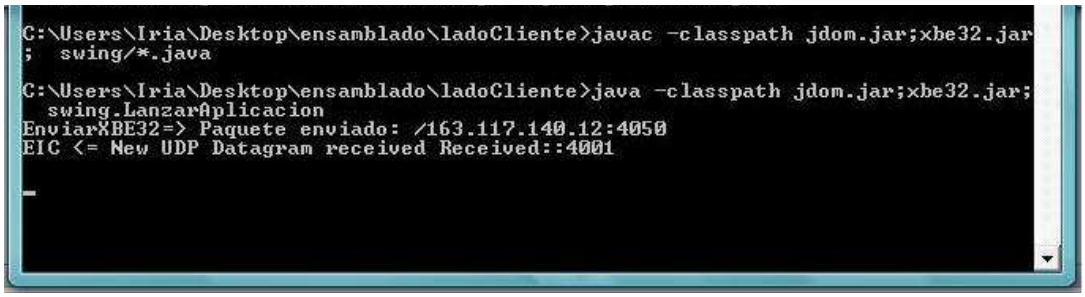
50. Log getReply

Y el XML generado con la respuesta será el siguiente:

```
- <lpv1>
  - <header>
      <msgID>0x6C86FF9D</msgID>
      <srcID>0x0000002742389798</srcID>
      <dstID>0x0000002742389772</dstID>
      <tstamp>1820832226</tstamp>
    </header>
  - <getDataReply>
      <sensorID>0x00000003</sensorID>
    - <data>
        <dataCat>0x00000700</dataCat>
        <int32Values>98</int32Values>
        <dataScale>0</dataScale>
      </data>
      <lifetime>1000</lifetime>
    </getDataReply>
  </lpv1>
```

51. XML con getReply

Por último, las trazas completas de esta operación de getRequest en el equipo cliente serían las siguientes:

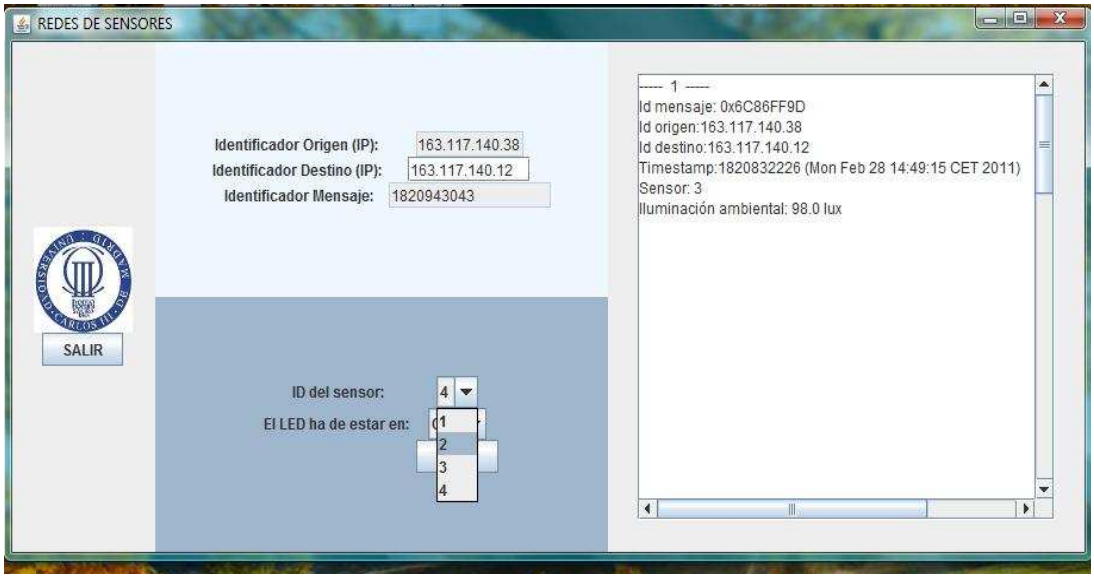


52. Trazas en el cliente del get y getReply

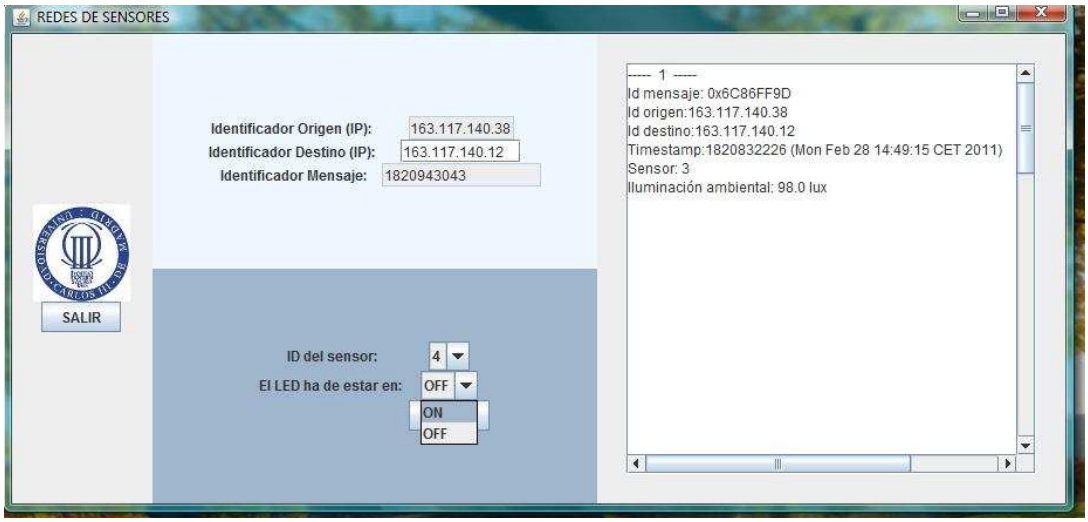
6.3.2 Ejemplo de operación Set

A continuación de la operación anterior realizamos una operación de set para el Led del sensor 3.

Para ello, partiendo de la pantalla de la pantalla *50. Log getReply* de la interfaz gráfica, pulsamos el botón setData. Se nos mostraría la siguiente pantalla, en la que seleccionamos el sensor y el estado (encendido/apagado) que queremos establecer en el LED:



53. Pantalla Set: elección de sensor destino



54. Pantalla Set: elección de estado del LED

Podemos apreciar que el identificador de mensaje ha cambiado, puesto que estamos generando una nueva petición. Ésta petición tendrá el siguiente formato en SENCOMLNG:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <lpv1>
  - <header>
      <msgID>0x6C8962C3</msgID>
      <srcID>0x2742389798</srcID>
      <dstID>0x2742389772</dstID>
      <tstamp>1821056262</tstamp>
    </header>
  - <setData>
      <sensorID>0x3</sensorID>
    - <data>
        <dataCat>0x00000311</dataCat>
        <data1Values>0x1</data1Values>
      </data>
    </setData>
  </lpv1>
```

55. Ejemplo de operación Set en SENCOMLNG

En el Gateway esto se traduce a iLWESP, resultado la siguiente petición a enviar al nodo coordinador a través de la UART:

0	**15 16**	**31**
0x00	0x0003 (sensor nº3)	0x03 (set)
0x000003 (LED)		0x11 (ON)

56. Ejemplo de SetReply en iLWESP

El código en el nodo coordinador sería el mismo que en el caso anterior, dado que las peticiones son las mismas. Para este caso los valores de la trama inalámbrica con la petición hacia el sensor serían:

- tramaEnvio[0]=0x00;
- tramaEnvio[1]=0x00;
- tramaEnvio[2]=0x03;
- tramaEnvio[3]=0x03 ; (set)
- tramaEnvio[4]=0x00;
- tramaEnvio[5]=0x00;

- tramaEnvio[6]=0x03;

- tramaEnvio[7]=0x11;

Donde cambiaría el proceso sería en el *endpoint*, el código que tramitaría esta petición sería el siguiente:

```
if( nDataCat==3){
    if( nDataCatUnidad==1) {
            if (bLightBulbState1 == OFF){
                vLightBulb(LED1, ON);
                 bLightBulbState1= ON;
            }
    }
    else{

        if (bLightBulbState1 == ON){
                vLightBulb(LED1, OFF);
                 bLightBulbState1= OFF;
            }
    }
}

    int longitudTrama=8;
    uint8 cadenaSetReply[longitudTrama];
    cadenaSetReply[0]=tramaSetReply.cabecera.proto;
    cadenaSetReply[1]=((cabecera.sensorID)>>8);
    cadenaSetReply[2]=(cabecera.sensorID);
    cadenaSetReply[3]= (uint8)tramaSetReply.cabecera.opCode;
    cadenaSetReply[4]=((cabecera.dataCat)>>16);
    cadenaSetReply[5]=((cabecera.dataCat)>>8);
    cadenaSetReply[6]=(uint8)tramaSetReply.cabecera.dataCat;
    cadenaSetReply[7]= (uint8)tramaSetReply.cabecera.dataCatUnidad;

    vTxDataFrame(0x01, cadenaSetReply);
```

57. Endpoint: código correspondiente a operación Set

Y los valores de la operación setReply serían:

- cadenaSetReply[0]=0x00;

- cadenaSetReply [1]=0x00;

- cadenaSetReply [2]=0x03;

- cadenaSetReply [3]=0x04;(Set ACK)

- cadenaSetReply [4]=0x00;
- cadenaSetReply [5]=0x00;
- cadenaSetReply [6]=0x03;
- cadenaSetReply [7]=0x01;

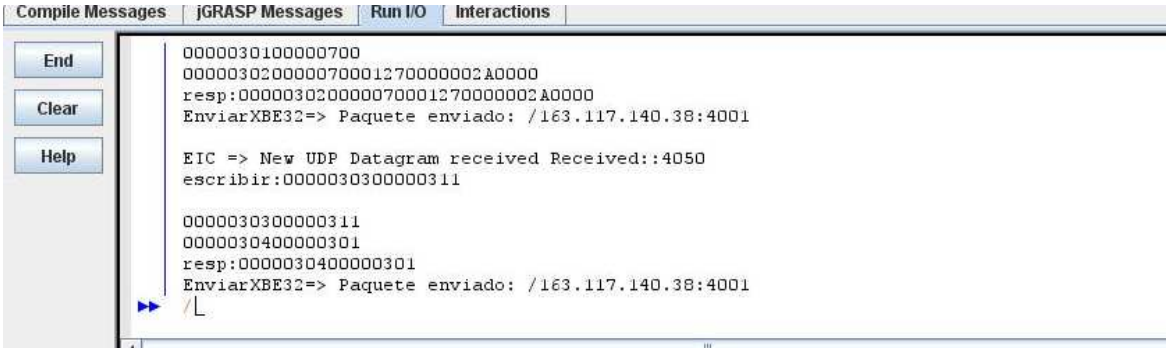
Este nuevo payload se mandaría al nodo coordinador, que tras tramitar la respuesta mediante el método generarSalida enviaría mediante UART al Gateway la respuesta que tendía el siguiente formato:

0	**15 16**	**31**
0x00	0x0003 (sensor nº3)	0x04 (setAck)
0x000003 (LED)		0x01 (ON)

58. SetAck en formato iLWESP

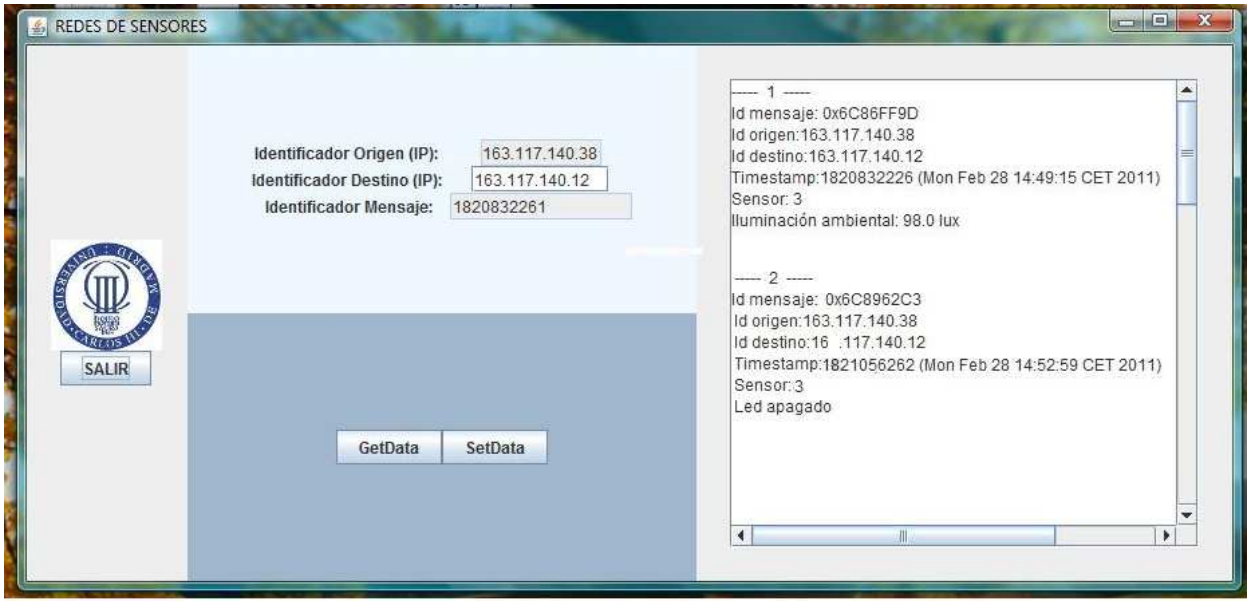
El Gateway, una vez recibida la interrupción de que le está llegando algo por el puerto serie, comprobará que efectivamente es una respuesta de tipo iLWESP. En el caso de que lo sea, generará la trama XBE32 y se la enviará al equipo de la parte cliente.

Esta operación de recibir tanto el Get como el Set y enviar las respuestas, se refleja en el Gateway mediante las siguientes trazas:



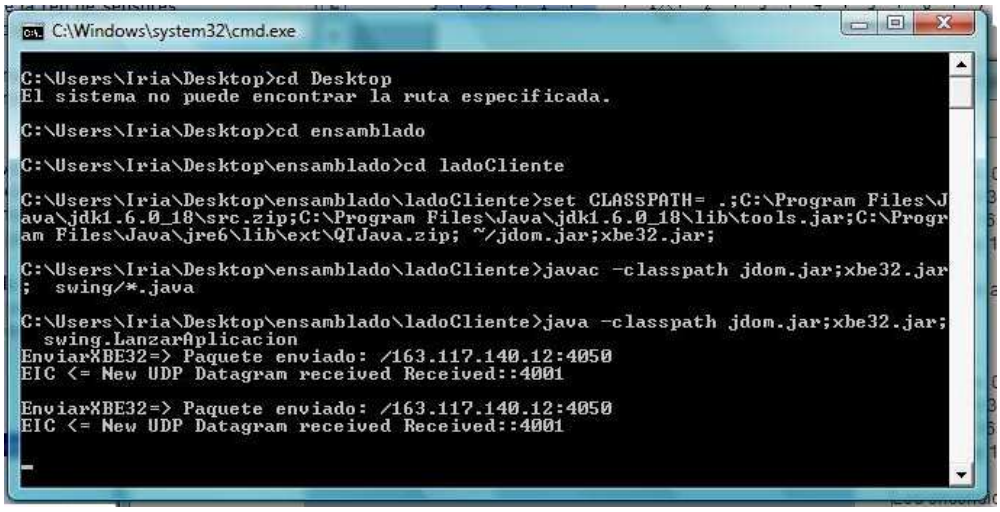
59. Trazas en Gateway de Get y Set

Finalmente, en el cliente, una vez refrescado el log de enviados y recibidos junto con la el log de la aplicación serían los siguientes:



60. Log acumulado de dos operaciones

Las trazas en el cliente serán las siguientes:



61. Trazas get y set en equipo cliente

Y el archivo XML generado con la respuesta a la operación de Set es el siguiente:

```
- <lpv1>
  - <header>
      <msgID>0x6C8962C3</msgID>
      <srcID>0x0000002742389798</srcID>
      <dstID>0x0000002742389772</dstID>
      <tstamp>1821056262</tstamp>
    </header>
  - <setDataAck>
      <sensorID>0x00000003</sensorID>
    - <data>
        <dataCat>0x00000311</dataCat>
        <data1Values>0x01</data1Values>
      </data>
    </setDataAck>
  </lpv1>
```

62. setAck en SENCOMLNG

7 El proyecto

7.1 Planificación temporal

La realización de este proyecto está concebida en bloques, los cuales listo a continuación:

7.1.1 Introducción a SENCOMLNG y XBE32

En esta etapa el objetivo es conocer con qué se va a trabajar, por ello se realiza un estudio de los *drafts* con los que vamos a trabajar, de la documentación asociada a ellos y de los protocolos IEEE 802.15.4 y ZigBee.

Para familiarizarse con la estructura de SENCOMLNG y XBE32 del código proporcionado, se realiza una aplicación Java sencilla que utilizando las librerías proporcionadas, forma documentos XML y según los datos introducidos por línea de comandos, los pasa a codificación XBE32.

Este apartado nos da un punto de partida para el diseño de la parte cliente, además de proporcionar una compresión profunda de la estructura de XBE32, necesaria para la parte cliente.

7.1.2 Estudio de la comunicación entre parte cliente y parte aplicación

Es necesario comprobar la viabilidad del envío bidireccional por la red del envío de tramas con los datos XBE32.

Se estudia el modo de hacerlo y se integra en el programa Java sencillo de la fase anterior.

7.1.3 Microprocesadores: introducción y desarrollo

Es vital conocer el funcionamiento de los microprocesadores que conforman los nodos y el coordinador de la red a utilizar. Para ello, una vez estudiado el manual de los mismos se pasa a probar programas de prueba en los sensores y realizar pequeños cambios sobre ellos.

Con esto, se pretende entender el funcionamiento de los mismos y de las funciones que utilizan.

El siguiente paso es el desarrollo del programa. Se va a comenzar con una operación de tipo Get y una vez conseguida, se irá incrementando el número de operaciones.

7.1.4 Realización interfaz gráfica en la parte cliente

Una vez finalizada la parte del microprocesador, será necesario definir e implementar la interfaz gráfica que generará las peticiones desarrolladas en los sensores.

7.1.5 Unión parte cliente con parte servidor: integración.

Una vez llegados a este punto, es necesario integrar la parte cliente, con la parte de la red de sensores y comunicarlos entre sí mediante la red.

Esto supone el fin de la parte de implementación.

7.1.6 Pruebas

Es necesario comprobar la robustez de lo desarrollado y realizar los ajustes necesarios si surge algún error.

7.1.7 Realización de la memoria: búsqueda de documentación y redacción final

Ya se tiene documentación de la fase preliminar de estudio pero, este es el momento de ponerla en orden.

La memoria no se realiza como un bloque totalmente aislado, sino que es algo que se va haciendo en paralelo a la implementación del proyecto.

El Estado del Arte se empieza a realizar desde el primer momento pero, en este punto final, habrá que añadir la información relativa al desarrollo, coste y maquetar el documento.

Si bien estos son los bloques definidos, no se han ido siguiendo de una manera secuencial. Por ejemplo, la interfaz gráfica ya tenía una primera versión antes de comenzar el desarrollo en la parte de la red de sensores.

Del mismo modo, una vez implementadas las operaciones Get y Set se ha realizado una primera versión final extremo a extremo del proyecto.

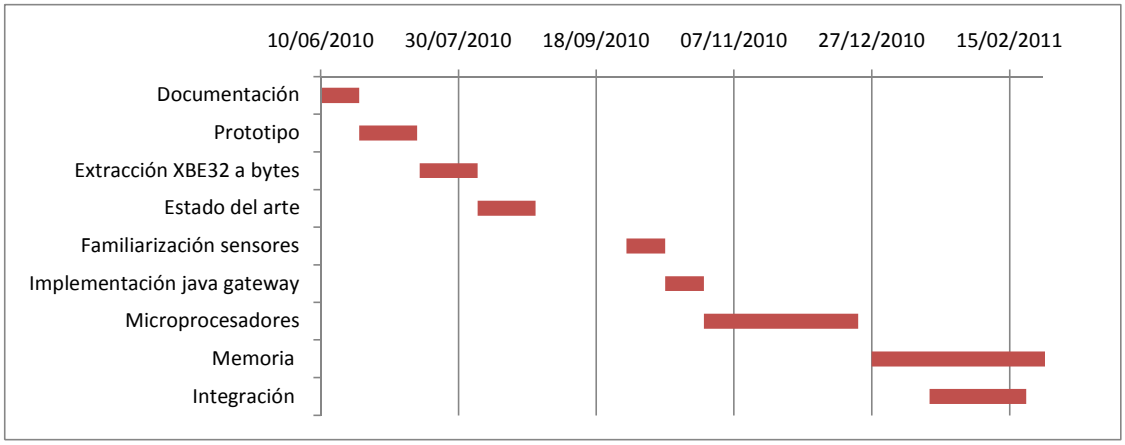
En cuanto a la dedicación, se ha dedicado unas 4 horas de media al día donde un día a la semana ha sido solo para la memoria o para la interfaz gráfica. Los mayores esfuerzos han estado centrados en la parte de la red de sensores.

A continuación se adjunta una tabla resumen de las horas dedicadas:

Actividad	**Fecha Inicio**	**Duración (semanas)**	**Duración (días)**	**Fecha Fin**	**Horas/día**	**Horas totales (sin fines de semana)**
Documentación	10/06/2010	2	14	23/06/2010	3	30
Prototipo	24/06/2010	3	21	14/07/2010	4	60
Extracción XBE32 a bytes	16/07/2010	3	21	05/08/2010	4	60
Estado del arte	06/08/2010	3	21	26/08/2010	3	45
Familiarización sensores	29/09/2010	2	14	12/10/2010	4	40
Implementación Java Gateway	13/10/2010	2	14	26/10/2010	4	40
Microprocesadores	27/10/2010	8	56	21/12/2010	4	160
Memoria	27/12/2010	9	63	27/02/2011	3	135
Integración	17/01/2011	5	35	20/02/2011	4	100
	Total	**37**	**259**			**670**

Tabla 7. Tabla temporal del proyecto

Esta tabla queda resumida en el siguiente diagrama de Gantt:



63. Diagrama de Gantt de la elaboración del proyecto

7.2 Análisis de los costes

En este apartado se detallarán los costes asociados al proyecto, divididos en tres secciones: costes hardware, costes software y costes de personal.

7.2.1 Costes Hardware

Esta sección se refiere al equipamiento utilizado para la realización del proyecto. Si bien, el material ya formaba parte del Departamento de Telemática y no se ha comprado específicamente para este proyecto, es necesario estimar el coste asociado. Además, se ha utilizado un ordenador personal para la realización del mismo.

Se toma como plantilla el documento proporcionado por la Universidad Carlos III en la sección de información sobre Proyectos de Fin de Carrera[30].

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable
PC sobremesa	800,00	50	9	60	60,00
Ordenador portátil	800,00	70	5	60	46,67
Sensores	250,00	100	5	60	20,83
				Total	127,50 €

Tabla 8. Costes Hardware

7.2.2 Costes Software

Este apartado detalle el coste de las licencias de instalación y uso de los diferentes programas utilizados para el desarrollo del proyecto y para la composición de la memoria.

Programa	Coste	IVA	Coste total	Coste /mes	Coste total
JDK 1.6.0	0€			0€	
JGrasp	0€			0€	
Notepad++	0€			0€	
JN-SW-4031 Code: Blocks Integrated Development Environment	0€			0€	
JN-SW-4007 Flash Programmer	0€			0€	
Microsoft Word 2007	86,87 €	15,64 €	102,51 €	2,85 €	25,65 €
				Total	25,65 €

Tabla 9. Costes Software

El único programa no gratuito es Microsoft Word 2007. Existen otras alternativas a este programa para la realización de documentos, pero se ha utilizado este por comodidad. Su coste es difícil de calcular puesto que se ha utilizado la versión instalada en los equipos de la universidad, pero el precio se calcula en base a un paquete con una sola licencia de uso doméstico[1].

Dado que hay versión de Office 2010, se calcula un tiempo de amortización de este programa de tres años.

7.2.3 Costes de personal

En este último apartado se detalla el coste generado por el personal encargado del proyecto, esto es, la proyectanda y el tutor.

La proyectanda ha estado dedicada a media jornada al proyecto mientras que el tutor tampoco ha estado en dedicación exclusiva al mismo.

Tomando como plantilla el documento proporcionado por la Universidad Carlos III en la sección de información sobre Proyectos de Fin de Carrera[30], que establece un salario mensual total de 4289,54 euros para un Ingeniero Senior y de 2694,30 para un Ingeniero Junior, obtenemos la siguiente tabla:

Categoría	Dedicación	Coste hombre/ mes	Número de meses	Total
Ingeniero Senior	10%	428,95	8	3431,60
Ingeniero Senior	30%	1286,96	1	1286,96
Ingeniero	50%	1347,20	9	12124,80
			Total	16843,36 €

Tabla 10. Costes de personal

7.2.4 Costes totales

Los costes totales del proyecto asciendes de 16886,81 euros. Se detallan en la siguiente tabla:

Concepto	**Coste**
Costes Hardware	127,80
Costes Software	25,65
Costes personal	16843,36
Total	**16996,81 €**

Tabla 11. Costes totales

8 Conclusiones y desarrollo futuro

8.1 Conclusiones

El lenguaje XML es ampliamente utilizado en aplicaciones de cualquier entorno, desde industrial a uso social, puesto que dada su flexibilidad, se puede adecuar a un uso concreto. Mediante SENCOMLNG se comprueba que XML se puede adecuar a la comunicación de redes de sensores inalámbricas mediante la definición de etiquetas con los campos más comunes en estos casos: dirección del nodo final, parámetro a consultar, etc.

Por otro lado, el lenguaje XML no es el idóneo para la comunicación con redes inalámbricas de sensores: estas redes suelen utilizarse para aplicaciones de monitorización, lo que puede llegar a generar grandes cantidades de tráfico. Mediante XBE32 es posible reducir una codificación de formato SENCOMLNG a otro formato más ligero y por tanto, más óptimo para la comunicación de este tipo.

Partiendo de estas dos consideraciones se ha implementado una aplicación externa para comunicarse con una red inalámbrica de sensores IEEE 802.15.4:

Se ha utilizado SENCOMLNG para codificar las peticiones, que posteriormente se han transformado a XBE32. En este formato han sido enviadas por la red hacia el Gateway de la red de sensores.
Con esto se ha comprobado que es posible codificar la información para la comunicación con una red de sensores mediante SENCOMLNG, independientemente de qué parámetros midan; para la realización de este proyecto no ha sido necesario disponer de los dispositivos hardware a la hora de realizar la codificación de las peticiones en XML, sino que la generación de la petición en SENCOMLNG ha sido totalmente independiente a estos.

Además, se ha podido observar que la transformación de SENCOMLNG a XBE32 efectivamente reduce el ancho de banda de la comunicación. Esto es se refleja en la diferencia en los tamaños en bytes de los archivos generados con cada petición y cada respuesta tanto en XML como en XBE32.

Por último, tras interactuar con los dispositivos inalámbricos se ha podido realizar la comunicación extremo a extremo comprobando que es posible monitorizar una red de sensores de una forma telemática, utilizando un único protocolo inalámbrico, 802.15.4, y mediante una implementación más bien sencilla.

8.2 Trabajos Futuros

Finalmente solo se han implementado las operaciones de Get y Set. Por ello, el primero trabajo futuro sería implementar el resto de las funciones:

La parte que ha quedado por desarrollar es la relativa a la programación en los microprocesadores y la integración posterior.

Para la operación de EventSet, una vez recibida la petición en el *endpoint*, lo único que haría falta sería activar un *timer* que periódicamente, por ejemplo, cada minuto, produjese una interrupción. Al producirse esta interrupción, habría que comprobar el límite marcado no se ha sobrepasado.

De haberse sobrepasado, se enviaría al nodo coordinador una trama con los datos del evento.

A partir de aquí el proceso de enviar esta información al cliente sería el mismo que para cualquier tipo de operación.

Esta comprobación se realizaría periódicamente hasta que llegase una trama de tipo EventStop, momento en el que se paralizaría el *timer*.

Para establecer estos temporizadores, Jennic proporciona Timers y TickTimers.

La opción más sencilla probablemente fuese TickTimer. La configuración del timer sería así:

```
//establecer contadores

vAHITickTimerConfigure(E_AHI_TICK_TIMER_DISABLE); //modo

                                                  configuración

vAHITickTimerWrite(0); // valor inicial del contador del contador a

                        cero

vAHI_TickTimerInterval(0x0E10); //el periodo cada cuanto se lanzará
                                  una interrupción, cada 3600
                                  segundos, 0xE10 en hexadecimal

//ejecutar el timer
vAHITickTimerConfigure(E_AHI_TICK_TIMER_RESTART); //el modo de
                               desde el valor establecido como inicial
vAHITickTimerIntEnable(true); // cuando se alcance el valor de
                               referencia (1 minuto), generar una
                               interrupción
```

64. Configuración del timer en el EventSet

Y el pseudocódigo para comprobar la interrupción sería:

```
//método que registra las interrupciones generadas por el ticktimer

void vAHI_TickTimerInit(PR_HWINT_APPCALLBACK prTickTimerCallback){

      //comprobar que la interrupción el del ticktimer y de una operación
      de eventSet

      //medir el parámetro que estemos monitorizando

      //comprobar si el valor sobrepasa algún umbral

      //si lo sobrepasa, llamar a vTxDataFrame con los datos que han

      generado el evento

      //poner a cero el contador

}
```

65. Tratamiento de las interrupciones en el EventSet

La otra operación a implementar sería la de monitorización. En el *endpoint* cuando se recibiese la petición, habría que configurar un temporizador (*timer*) que cada vez que el

tiempo marcado por el periodo de medida pasase, provocase una interrupción y se realizase una medida.

Cuanto esto hubiese pasado un número de veces igual a la constate creada a partir de dividir el periodo de notificación entre el periodo medida, estos datos se enviarían al nodo coordinador, el array que almacenase esta información se limpiaría y la constante se pondría a cero.

Cuando se recibiese un mensaje MonitorStop, se mandaría por última vez las medidas realizadas y se pararía el temporizador.

El código que configuraría las interrupciones sería el siguiente:

```
//establecer contadores

vAHITickTimerConfigure(E_AHI_TICK_TIMER_DISABLE); //modo
                                                  configuración

vAHITickTimerWrite(0); // valor inicial del contador del contador a
cero

vAHI_TickTimerInterval(periodoMedida); //el periodo cada cuanto se
                                       lanzará una interrupción

//ejecutar el timer
vAHITickTimerConfigure(E_AHI_TICK_TIMER_RESTART); //el modo de
                            funcionamiento será empezar
                            desde el valor inicial pero seguir
                            contando aunque se produzca la
                            interrupción

vAHITickTimerIntEnable(true); // cuando se alcance el valor de
                            referencia (periodoMedida), generar una
                            interrupción
```

66. Configuración del Timer en el MonitorStart

El pseudocódigo cada vez que saltase una interrupción sería el siguiente:

```
//método que registra las interrupciones generadas por el ticktimer

void vAHI_TickTimerInit(PR_HWINT_APPCALLBACK prTickTimerCallback){

      //comprobar que la interrupción el del ticktimer y de una operación

      de monitorización

      //medir el parámetro que estemos monitorizando

      //comprobar si el tiempo del contador sobrepasa o no el valor de
      periodo de notificación mediante u32AHI_TickTimerRead();

      //si lo sobrepasa, llamar a vTxDataFrame con los datos recolectados
      y ponerlo a cero

      //si no lo sobrepasa, almacenar los datos y seguir

}
```

67. Tratamiento de interrupciones en operaciones Monitor

Otra posible mejora que además contempla SENCOMLNG es realizar varias peticiones por envío desde el cliente:

Un modo de afrontar esto sería que fuese el Gateway el encargado de extraer cada una de las peticiones y enviarlas una a una al nodo coordinador, esperar la respuesta y enviar la siguiente. Estas respuestas se irían almacenando y conforme tuviese el total de las respuestas a las peticiones solicitadas, las enviaría de vuelta a la red cliente.

Esto no afectaría a iLWESP, sólo a la lógica de programación del Gateway.

Otra mejora sería desarrollar un protocolo de descubrimiento de servicios de la red de sensores para conocer cuántos sensores hay en la red y que parámetros monitorizan cada uno:

Para ello, la aplicación cliente debería mandar una petición de descubrimiento de servicios cada vez que se iniciase y debería ser el propio Gateway quien le contestase.

El problema es que en SENCOMLNG no hay definida ninguna operación de este estilo. Por ello habría que definir un formato de mensaje específico solo para esta causa.

Por ejemplo, se podría crear un formato de mensaje en *string*, dado que el intercambio de mensajes sería a nivel UDP con los siguientes campos:

- Dirección IP del Gateway
- Puerto en el escucha
- Puerto en el que envía
- Número de *endpoints*
- Dirección del coordinador
- Dirección del primer nodo
- Servicios que ofrece el primer nodo
- Dirección del segundo nodo
- Servicios que ofrece el segundo nodo
- …
- Dirección del último nodo
- Servicios que ofrece el segundo nodo

Un posible trabajo a desarrollar sería integrar XBE32 en el nodo coordinador de la red inalámbrica de sensores. De este modo el Gateway sólo tendría que recibir el datagrama UDP y enviarlo mediante puerto serie al nodo coordinador. Todo el proceso de extraer la información de XBE32 recaería en este nodo.

La ventaja de esto sería saltarse el paso de traducción intermedia de XBE32 a iLWESP que actualmente se está realizando y por tanto reducir el tiempo transcurrido desde que se realiza la petición hasta que llega la respuesta a la aplicación externa.

Por último, se podía integrar el trabajo realizado junto con una aplicación real de monitorización basada en IEEE 802.15.4 como, por ejemplo, la detallada en el apartado 3.2.

9 Glosario

Las definiciones se han obtenido de WIKIPEDIA (http://wikipedia.org y http:// http://es.wikipedia.org).

- 6LoWPAN, *IPv6 over Low power Wireless Personal Area Networks*: una tecnología inalámbrica para transportar información sobre IP.
- ACK, *ACKnowledgment*: En telecomunicaciones, la respuesta a una petición anterior.
- ADC, *Analogic to Digital Converter*: Conversor analógico-digital.
- AES, *Advanced Encription Standard*: un algoritmo de cifrado usado en seguridad.
- API, *Application Programming Interface*: una serie de reglas y especificaciones que un programa software ha de seguir.
- APS, A*pplication Support Sublayer*: en ZigBee, el subnivel de soporte a la aplicación.
- ASK, *Amplitude Shift Key*: una técnica de modulación utilizada en telecomunicaciones.
- BPSK, *Binary Phase Shift Keying* : una técnica de modulación utilizada en telecomunicaciones
- CSMA/CA, *Carrier Sense Multiple Access with Collision Avoidance*: acceso múltiple por detección de portadora con evasión de colisiones.
- CSS, *Chirp Spread Spectrum*: una técnica de espectro ensanchado utlilizada en telecomunicaciones.
- DAC, *Digital to Analogic Converter*: Conversor de digital a analógico.
- DSSS, *Direct Sequence Spread Spectrum*: una técnica de espectro ensanchado utilizada en telecomunicaciones.
- FFD, *Full-function devices*: En IEEE 802.15.4, los nodos con mayor capacidad de la red.

- FSK*, Frequency-shift keying*: una técnica de modulación utilizada en telecomunicaciones.
- GFSK, Gaussian FSK: una técnica de modulación utilizada en telecomunicaciones
- GTS, *Guaranteed Time Slots*: Slots temporales garantizados.
- GPIO, *General Purpose Input/Output*: es un pin genérico en los microcontroladores cuyo comportamiento puede ser programado vía software.
- HART, *Highway Addressable Remote Transducer Protocol*: un protocolo para la comunicación de equipos mediante fieldbus.
- IEEE, Institute of Electrical and Electronic Engineers.
- IEEE 802.15.4: Protocolo IEEE 802.15.4 del IEEE.
- iLWESP, internal *Light-Weight Exterior Sensornet Protocol* : ver apartado 5.2
- IrDA, *Infrared Data Association*: protocol que define especificaciones físicas para el intercambio de datos a corta distancia sobre luz infrarroja.
- ISM, *Industrial, Scientific and Medical*: hace referencia a aplicaciones industriales, científicas y médicas.
- IVA: Impuesto sobre el Valor Añadido
- JDOM, *Java Document Object Model*: es un API JAVA para interactuar con documentos XML.
- KVP, K*ey-Value Pair*: en seguridad, un par de claves.
- LED*, Light Emitting Diode*: dispositivo semiconductor de tipo diodo que emite luz.
- LWESP, *Light-Weight Exterior Sensornet Protocol* : ver sección 4.3.
- M2M: comunicación de máquina a máquina.
- MAC, *Medium Access Control*: En el protocolo IEEE 802.15.4, la capa de control de acceso al medio.
- MCPS, *MAC Common Part Sublayer:* En IEEE 802.15.4, una de sus subcapas.

- MLME, *MAC Sublayer Management Entity*: En IEEE 802.15.4, la entidad de control de la subcapa MAC.
- MPSK, *Multiple PSK*: una técnica de modulación utilizada en telecomunicaciones.
- OQPSK, *Offset quadrature phase-shift*: un tipo de modulación utilizada en telecomunicaciones.
- PAN, *Personal Area Network*: red de área personal.
- PLME, P*hysical Layer Managemente Entity*: en IEEE 802.15.4, el interfaz a la entidad de administración de la capa física.
- PHY: En el protocolo IEEE 802.15.4, la capa física.
- PSSS, *Parallel Sequence Spread Spectrum*: una técnica de espectro ensanchado utilizada en telecomunicaciones.
- RF: radiofrecuencia
- RFD*,* R*educed- Function Devices*: En iEEE 802.15.4, los dispositivos finales de la red.
- RISC, *Reduced Instruction Set Computer*: un tipo de microprocesador cuyas instrucciones son de tamaño fijo.
- SAX, *Simple API for XML*: Un API para interactuar de documentos XML
- SAPs*, Service Access* Points: En IEEE 802.15.4 los puntos de acceso a servicios.
- SENCOMLNG, *SENsor COMmunication LANGuage*: un lenguaje basado en XML específico para la comunicación entre aplicaciones y redes de sensores.
- SPI*, Serial Peripheral Interface*: un estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos.
- UART, *Universal Asincronous Receiver Transmiter*: permite controlar los puertos y dispositivos serie.
- UDP, *User Datagram Protocol*: un protocolo del nivel de transporte para el intercambio de paquetes.

- UWB, *Ultra Wide Band*: una tecnología radio que se usa en bajos niveles de energía pra comunicaciones a corto alcance.
- WHAN, *Wireless Home Area Network:* red inalámbrica de ámbito doméstico.
- WPAN, W*ireles Personal Area Network:* red inalámbrica de ámbito personal.
- WSNs, *Wireless Sensors Networks*: redes inalámbricas de sensores.
- XML, *eXtensible Markup Language*: un metalenguaje que permite definir la gramática de lenguajes específicos.
- ZC, *ZigBee Coordinator*: nodo coordinador de una red ZigBee.
- ZDO, *ZigBee Device Object*: En ZigBee, uno de los componentes del nivel de aplicación.
- ZED, *ZigBee End Device:* nodo final de una red ZigBee.
- ZR, *ZigBee Router*: nodo de una red ZigBee que hace las funciones de router.

10 Bibliografía

[1] **Manuel Urueña, David Larrabeiti,.** eXtensible Binary Encoding (XBE32). <draft-uruena-XBE32-02.txt>. [En línea] 7 de septiembre de 2007. Disponible en http://tools.ietf.org/html/draft-uruena-XBE32-02. [Consultado el 21 de febrero de 2011].

[2] **Ángel Cuevas, Manuel Urueña, Annett Laube, Laurent Gomez.** LWESP:Light-Weight Exterior Sensornet Protocol. [En línea] 5-8 de julio de 2009. Disponible en http://www.it.uc3m.es/~acuevasr/publicaciones/ISCC09.pdf. [Consultado el 21 de febrero de 2011].

[3] **Carles Gómez Montenegro, Josep Paradells Aspas, José Eugenio Caballero Herrero.** *Sensors everywhere. Wireless Network Technologies and Solutions.* s.l. : Fundación Vodafone España, 2010. 978-84-934740-5-8.

[4] **Tweed, K.** News: GE Says ZigBee is Better Than Wi-Fi. *htt.//www.greentechmedia.com.* [En línea] 9 de diciembre de 2010. Disponible en http://www.greentechmedia.com/articles/read/ge-says-ZigBee-is-better-than-wifi/ [Consultado el 21 de febrero de 2011].

[5] **Institute of Electrical and Electronic Engineers**. Part 15.4: Wireless Medium Access. Control (MAC) and Physical Layer (PHY). Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs) [En línea] 1 de octubre de 2003. Disponible en http://standards.ieee.org/getieee802/download/802.15.4-2003.pdf [Consultado el 22 de febrero de 2011].

[6] **Institute of Electrical and Electronic Engineers**. 802.15.4-2006 - IEEE Standard for Local and Metropolitan Area Networks - Specific Requirements - Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPANs). [En línea] 8 de septiembre de 2006. http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf [Consultado el 22 de febrero de 2011].

[7] **Institute of Electrical and Electronic Engineers**. 802.15.4a-2007 - IEEE Standard for Local and Metropolitan Area Networks - Specific Requirements - Wireless Medium

Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs). Amendment 1: Add Alternate PHYs. [En línea] 31 de agosto de 2007. Disponible en http://standards.ieee.org/getieee802/download/802.15.4a-2007.pdf [Consultado el 22 de febrero de 2011].

[8] **Institute of Electrical and Electronic Engineers**. 802.15.4c-2009 - IEEE Standard for Local and Metropolitan Area Networks - Specific Requirements - Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPANs). Amendment 2: Alternative Physical Layer Extension to support one or more of the Chinese 314–316 MHz, 430–434 MHz, and 779–787 MHz bands [En línea] 17 de abril de 2009. Disponible en http://standards.ieee.org/getieee802/download/802.15.4c-2009.pdf [Consultado el 22 de febrero de 2011].

[9] **Institute of Electrical and Electronic Engineers**. 802.15.4d-2009 - IEEE Standard for Local and Metropolitan Area Networks - Specific Requirements - Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPANs). Amendment 3: Alternative Physical Layer Extension to support the Japanese 950 MHz bands [En línea] 17 de abril de 2009 Disponible en http://standards.ieee.org/getieee802/download/802.15.4d-2009.pdf [Consultado el 22 de febrero de 2011] .

[10] **ZigBee Alliance.** ZigBee Specifications. [En línea] http://www.ZigBee.org/Specifications.aspx. [Consultado el 22 de febrero de 2011].

[11] **N. Kushalnagar, G. Montenegro y C. Schumacher.** RFC 4919. IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals. [En línea] agosto de 2007. Disponible en https://datatracker.ietf.org/doc/rfc4919/ [Consultado el 21 de febrero de 2011].

[12] **G. Montenegro, N. Kushalnagar, D. Culler.** RFC 4944. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. [En línea] septiembre de 2007. Disponible en https://datatracker.ietf.org/doc/rfc4944/ [Consultado el 21 de febrero de 2011].

[13] **Dash7 Alliance.** Dash7 technology. [En línea] http://www.dash7.org/index.php?option=com_content&view=article&id=9&Itemid=10. [Consultado el 9 de febrero de 2011].

[14]ISO/IEC 18000-7: Information technology- Radio frequency identification for item management. *Part 7: Parameters for active air interface communications at 433 MHz.* [En línea] 15 de agosto de 2004. Disponible en http://www.youwokeji.com.cn/down/18000-7.pdf[Consultado el 22 de febrero de 2011].

[15]**HART Communication Foundation.** WirelessHART Technical Data Sheet. [En línea] 15 de mayo de 2007. Diponible en http://www.hartcomm.org/protocol/training/resources/wiHART_resources/wirelesshart_datasheet.pdf. [Consultado el 22 de febrero de 2011].

[16]**HART Communication Foundation**. About the HART Protocol. [En línea] 2009. Disponible en http://www.hartcomm.org/protocol/about/aboutprotocol.html [Consultado el 22 de febrero de 2011].

[17]**Samson.** HART Communications. [En línea] Disponible en http://www.samson.de/pdf_en/l452en.pdf [Consultado el 22 de febrero de 2011].

[18]**Z-Wave Alliance.** The Z-Wave Alliance puts total control in your hands. [En línea] Disponible en http://www.z-wavealliance.org/modules/AllianceStart/ [Consultado el 22 de febrero de 2011].

[19]**Microchip.** MiWi. [En línea] http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2664&param=en520414 [Consultado el 22 de febrero de 2011].

[20]**Edison Smart Connect**. Real World ZigBee Testing. [En línea] Marzo de 2009. Disponible en http://www.zigbee.org/imwp/idms/popups/pop_download.asp?contentID=15571 [Consultado el 25 de febrero de 2010].

[21]**Southern California Edison.** News: Southern California Edison Installs “Smart” Meters in San Gabriel Valley. [En línea] 15 de diciembre de 2009. Disponible en http://www.edison.com/files/121509_news1.pdf [Consultado el 21 de febrero de 2011].

[22]**Southern California Edison..** SCE: Helping You Make Smarter Energy Choices. [En línea] Disponible en

http://www.sce.com/CustomerService/smartconnect/default.htm?from=redirect. [Consultado el 21 de febrero de 2011].

[23]Bryan Carl R. Chug, Iean Kenneth G. De Mesa, Kim Carl A. Filomeno, Vhalerie Lorraine L. Lee, Dino Antonio D. Poblete. CROPWISE: Wireless Crop Monitoring and Management System Utilizing Zigbee Wireless Technology [En línea] De la Salle University, Manila, Filipinas, agosto de 2009. Disponible en http://coethesis.com/index.php?option=com_docman&task=doc_details&gid=199&Itemid=60 [Consultado el 21 de febrero de 2011].

[24]**Jennic Ltd.** Product Brief – JN5139-EK000 . JN5139-EK000 IEEE802.15.4/JenNet Evaluation Kit. [En línea] 2008. Disponible en http://www.jennic.com/files/product_briefs/JN5139-EK000-PBv1.51.pdf [Consultado el 22 de febrero de 2011].

[25]**NXP Laboratories UK.** DR1048 Sensor Board. Reference Manual. JN-RM-2030 v1.4. [En línea] 16 de junio de 2010. Disponible en http://www.jennic.com/files/support_files/JN-RM-2030-DR1048-Sensor-Board-1v4.pdf [Consultado el 17 de febrero de 2011].

[26]**NXP Laboratories UK**. DR1047 Controller Board. Reference Manual, JN-RM-2029 v1.2. [En línea] 18 de junio de 2010. Disponible en http://www.jennic.com/files/support_files/JN-RM-2029-DR1047-Controller-Board-1v2.pdf [Consultado el 17 de febrero de 2011].

[27]RXTX Wiki. [En línea] http://rxtx.qbang.org/wiki/index.php/Main_Page [Consultado el 23 de febrero de 2011].

[28]**Jennic.** LPRF Board API. Refence Manual. [En línea] 22 de noviembre de 2010. Disponible en http://ftp.jennic.com/files/support_files/JN-RM-2003-LPRF-Board-API.pdf [Consultado el 21 de febrero de 2011].

[29]**TAOS.** Ambient Light Sensor (ALS) : TSL2550D. [En línea] Disponible en http://www.taosinc.com/ProductDetails.aspx?id=133 [Consultado el 22 de febrero de 2011].

[30]**Universidad Carlos III de Madrid.** Formulario de Presupuesto de Proyecto de Fin de Carrera. Disponible en

http://www.uc3m.es/portal/page/portal/administracion_campus_leganes_est_cg/proyecto_fin_carrera [Consultado el 25 de febrero de 2011].

[31] Projecto HeartCycle. Disponible en http://www.heartcycle.eu/ [Consultado el 15 de diciembre de 2010].

[1] http://www.tallerpc.net/Microsoft_Office_Home_and_Student_2007_42899_p.htm