Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Proyecto Fin de Carrera
Ingeniería en Informática

Curso 2011/2012
Mayo 2012

DISEÑO Y EVALUACIÓN DEL API DE CONTROL DE UN RADIO 802.11 b/g PARA SU USO EN DISPOSITIVOS EMBEBIDOS

Autor: Joaquín Ruiz Lite
Director: David Gascón Cabrejas
Ponente: Javier Martínez Rodríguez

Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza

Dep. de Ingeniería e Informática de Sistemas.
Escuela de Ingeniería y Arquitectura
María de Luna 1, 50018 Zaragoza

Libelium Comunicaciones Distribuidas S.L.
María de Luna 11, nave 5 (CEEIARAGON)
50018 Zaragoza

Resumen

Las redes sensoriales inalámbricas están formadas por una serie de nodos que miden parámetros de interés mediante sensores y trabajan en conjunto para enviar esa información. El dispositivo *Waspmote*, es un equipo de comunicaciones diseñado para implementarse en redes sensoriales basado en un microprocesador de bajo consumo, al que se pueden incorporar módulos a través de sus entradas analógicas y digitales.

Dichos dispositivos, se comunican mediante tecnología *ZigBee*, utilizando una máquina que dispone de radio *ZigBee* para controlar y recibir información de los nodos sensoriales. Sin embargo, si incorporamos un módulo Wi-Fi a los nodos, podremos comunicarlos con otro dispositivo que disponga de tecnología Wi-Fi o con un servidor de Internet, sin necesidad de hardware adicional.

Este Proyecto Fin de Carrera consiste en la selección de un módulo Wi-Fi 802.11 b/g para su integración en el dispositivo *Waspmote*, así como del desarrollo de las librerías necesarias para el control y manejo a bajo nivel (API).

Las principales funcionalidades de la biblioteca desarrollada son transmisiones en modo cliente para comunicar los nodos sensoriales con Internet y transmisiones en modo Ad-hoc con otro dispositivo que disponga de tecnología Wi-Fi. Dichas transmisiones se realizan mediante diferentes protocolos y transmisiones.

También se ha desarrollado una serie de aplicaciones para probar el gran abanico de opciones de conexión que ofrece la tecnología 802.11 b/g mediante transmisiones en modo cliente y en modo Ad-hoc.

Se han desarrollado aplicaciones específicas para las plataformas iPhone y Android que permiten capturar las tramas enviadas desde los nodos sensoriales.

Finalmente, se ha hecho un informe de prestaciones del módulo definitivo, tanto pruebas de distancia/cobertura de la señal, como pruebas de consumo del módulo, incluyendo algoritmos que permiten optimizar el tiempo de vida de las redes, poniendo el radio en modo dormir, o cambiando características como la potencia de transmisión o el ratio de transmisión.

Agradecimientos

A Mamá, Yaya, Estí, Papá, y a los compañeros de Libelium

Índice General

Índice de Figuras

Índice de Tablas

Acrónimos

API. Application Programming Interface

PFC. Proyecto Fin de Carrera

I+D. Investigación y Desarrollo

CTO. Chief Technical Officer

DIIS. Departamento de Informática e Ingeniería de Sistemas

CEEI Aragón. Centro Europeo de Empresas e Innovación de Aragón

WSN. Wireless Sensor Networks

IEEE. Institute of Electrical and Electronics Engineers

iOS. iPhone Operating System

IDE. Integrated Development Environment

WiMAX. Worldwide Interoperability for Microwave Access

GPRS. General Packet Radio Service

AP. Access Point

UART. Universal Asynchronous Receiver-Transmitter

SPI. Serial Peripheral Interface

WEP. Wired Equivalent Privacy

WPA. Wi-Fi Protected Access

WPS. Wi-Fi Protected Setup

PSK. Pre-Shared Key

TCP. Transmission Control Protocol

DHCP. Dynamic Host Configuration Protocol

UDP. User Datagram Protocol

DNS. Domain Name System

ARP. Address Resolution Protocol

ICMP. Internet Control Message Protocol

FTP. File Transfer Protocol

HTTP. HyperText Transfer Protocol

HTML. HyperText Markup Language

RFID. Radio Frequency IDentification

SMA. SubMiniature version A

RAM. Random Access Memory

I. MEMORIA

Capítulo 1

Introducción

1.1. Marco de trabajo

El Proyecto Fin de Carrera (PFC) se ha desarrollado en colaboración con la empresa Libelium Comunicaciones Distribuidas, S.L., spin off de la Universidad de Zaragoza. Libelium es una empresa dedicada al diseño y fabricación de hardware para la implementación de redes sensoriales inalámbricas, redes malladas y protocolos de comunicación para todo tipo de redes inalámbricas distribuidas.

El proyecto se ha desarrollado bajo la supervisión, como director del proyecto, de D. David Gascón Cabrejas, Director de I+D y CTO de Libelium, y del Dr. Javier Martínez Rodríguez, Catedrático del área de Lenguajes y Sistemas Informáticos del Departamento de Informática e Ingeniería de Sistemas (DIIS) de la Universidad de Zaragoza, como ponente.

El trabajo se ha llevado a cabo entre los meses de Septiembre de 2011 y Febrero de 2012 en las instalaciones de la empresa sitas en el Centro Europeo de Empresas e innovación de Aragón (CEEI Aragón), bajo el formato de proyecto fin de carrera acogido a un convenio de colaboración entre la Universidad de Zaragoza y la empresa gestionado por el servicio UNIVERSA.

1.2. Estado del arte y motivación

Las redes sensoriales inalámbricas o redes WSN (*wireless sensor networks*) [1], están formadas por una serie de nodos que tienen acceso al mundo exterior por medio de sensores. El nombre que se le da a este tipo de dispositivos es el de "mote" -traducción inglesa de mota de polvo-, debido a su pequeño tamaño y la idea de que pueden ser colocados en cualquier parte.

Estos dispositivos tienen la capacidad de comunicarse entre sí mediante la creación de redes malladas utilizando el protocolo *ZigBee* [2]. De este modo, es posible transmitir la información adquirida a través de la red hasta un punto de control que registre los valores observados y pueda tomar decisiones.

Una limitación de estas redes es la necesidad de un dispositivo o un equipo que disponga dicha tecnología *ZigBee* para comunicar los resultados al usuario final. Ésta es la principal motivación del desarrollo del radio Wi-Fi [3], ya que dicha tecnología es más conocida y extendida que la *ZigBee*.

Disponiendo de un radio Wi-Fi en los nodos se puede interactuar con un dispositivo que tenga la tecnología Wi-Fi (un Smartphone, Tablet, ordenador portátil...) mediante conexión Ad-hoc [4]. Además, con dicho radio Wi-Fi y un router estándar,

se puede enviar información a un servidor en Internet (Nube) mediante conexión en modo cliente (infraestructurada [5]).

Así mismo, la utilización del radio Wi-Fi junto con las redes WSN ya implementadas con *ZigBee,* nos ofrecen un amplio abanico de posibilidades para el futuro.

1.3. Objetivos del proyecto

El **objetivo** principal de este PFC es dotar a la plataforma Libelium Waspmote [6] de un módulo emisor/receptor Wi-Fi bajo el estándar IEEE 802.11 b/g.

Para ello, se realizan las siguientes tareas:

1. Selección de un integrado Wi-Fi para su integración en el dispositivo *Waspmote*.

2. Desarrollo de la biblioteca de control y manejo a bajo nivel necesaria para programar el funcionamiento del módulo Wi-Fi. Teniendo en cuenta limitaciones de un dispositivo empotrado, básicamente relativas a memoria y consumo.

3. Desarrollo de aplicaciones de alto nivel para la comunicación a través de dicho módulo Wi-Fi desde:
   - PC: programa multiplataforma que se comunique con el radio Wi-Fi utilizando los diferentes protocolos de comunicación implementados.
   - Smartphone: aplicaciones para dispositivos *Android* y dispositivos *iOS* (iPhone/iPad*)* para comunicación mediante una red Ad-hoc.

También se ha estudiado de rendimiento del módulo Wi-Fi en un entorno real, así como programas de prueba interactuando con otros módulos ya existentes.

1.4. Desarrollo del proyecto

El trabajo se ha desarrollado en diferentes etapas, las cuales se presentan a continuación:

- **Definición del trabajo y estudio del entorno**
  En primer lugar, se definió y acotó el trabajo a realizar. A tal fin se estudiaron los diferentes protocolos de transmisión inalámbrica existentes en la plataforma de redes sensoriales y el protocolo Wi-Fi a desarrollar, así como el funcionamiento del entorno de programación *Waspmote IDE* [7] y de las API ya implementadas.

- **Elección Hardware**
  La compañía adquirió dos integrados Wi-Fi. Nuestro trabajo consisitó en la elección del más adecuado para la plataforma *Waspmote*. Acto seguido, se adaptó el prototipo final, el cual se integra en el dispositivo *Waspmote*. Y finalmente se comprobaron las conectividades utilizando el *Waspmote IDE*, desarrollando una primera versión de lo que sería la API del radio Wi-Fi.

- **Desarrollo de la API**
  La fase principal es el desarrollo de las librerías para la comunicación con el integrado Wi-Fi utilizando diferentes protocolos de comunicación de una manera sencilla. Teniendo en cuenta las limitaciones de memoria y consumo de energía de los dispositivos empotrados a la hora de desarrollar esta API.

- **Software Asociado**
  A continuación, se desarrolla software para la comprobación del funcionamiento de las funciones implementadas en cada tipo de protocolo (software para PC y aplicaciones para *Android y iPhone*).

- **Evaluación**
  Finalmente, se han realizado las pruebas en un entorno real para comprobar el correcto funcionamiento y determinar el rendimiento y prestaciones del radio Wi-Fi.

Las fases de desarrollo se explican en detalle en el Anexo F.

1.5. Tecnologías y herramientas empleadas

Durante el desarrollo del Proyecto se han empleado diversos lenguajes de programación y herramientas de desarrollo de los cuales, a continuación, se destacan los más relevantes:

- Para la programación de la API, se ha utilizado el entorno de desarrollo Waspmote IDE, y el lenguaje de programación C++.

  Waspmote IDE Entorno de programación para programar dispositivos *Waspmote*. Aquí se incluyen las librerías y se trabajan con ellas.
  C++ Lenguaje de programación orientado a objetos, utilizado para desarrollar la API.

- Para el desarrollo de las aplicaciones con iOS, se ha utilizado el entorno de programación XCode y el lenguaje de programación Objetive-C.

  XCode Entorno de programación para *MAC*. Aquí se han desarrollado las aplicaciones para dispositivos *iPhone/iPad*.
  Objetive-C Lenguaje de programación orientado a objetos, utilizado en el desarrollo de aplicaciones para dispositivos *iPhone/iPad*.

- Y finalmente, para el desarrollo de las aplicaciones para Android y para PC, se ha utilizado el entorno de programación Eclipse, y el lenguaje de programación Java.

  Eclipse Entorno de programación multilenguaje. Aquí se han desarrollado el software para PC y las aplicaciones para dispositivos *Android*.
  Java Lenguaje orientado a objetos y multiplataforma, utilizado en el desarrollo de software de prueba para PC y aplicaciones para dispositivos *Android*.

1.6. Estructura de la memoria

El presente documento está dividido en dos partes: **Memoria**, donde se explica el desarrollo del proyecto, y **Anexos**, donde se amplía la información de ciertos puntos relevantes.

Respecto a la Memoria, la misma se ha estructurado en capítulos:

- **Capítulo 1:** expone los objetivos del proyecto e introduce el documento.
- **Capítulo 2:** describe el sistema *Waspmote* actual, las distintas tecnologías inalámbricas y el sistema deseado con el desarrollo del módulo Wi-Fi.
- **Capítulo 3:** describe la elección de integrado Wi-Fi y se explican las principales características del integrado Wi-Fi final.
- **Capítulo 4:** describe el funcionamiento de las bibliotecas *Waspmote* y la biblioteca Wi-Fi desarrollada.
- **Capítulo 5:** describe las pruebas realizadas para comprobar las prestaciones y el correcto funcionamiento del dispositivo, junto con los resultados obtenidos.
- **Capítulo 6:** describe el programa PC y las aplicaciones móviles desarrolladas.
- **Capítulo 7:** analiza los resultados del proyecto y sus conclusiones y se exponen las líneas futuras de desarrollo.

Existen también 7 Anexos con la siguiente información:

- **Anexo A:** Descripción detallada del dispositivo *Waspmote*.
- **Anexo B**: Descripción hardware del módulo Wi-Fi final.
- **Anexo C**: Descripción detallada de la biblioteca Wi-Fi desarrollada, junto con códigos de ejemplo.
- **Anexo D**: Explicación y código de las pruebas realizadas.
- **Anexo E**: Descripción detallada del software desarrollado.
- **Anexo F**: Explicación en profundidad de las distintas fases de desarrollo.

Capítulo 2

Descripción del Sistema

2.1. Tecnologías existentes

Actualmente existen varias tecnologías para transmitir información de manera inalámbrica como ZigBee, Bluetooth [8], Wi-Fi, WiMAX [9]. Cada una de ellas es utilizada para distintas finalidades dependiendo esencialmente de su alcance y consumo.

- La tecnología **ZigBee** está basada en el estándar IEEE 802.15.4, utiliza la banda de frecuencia libre de 2,4GHz, proporciona un consumo muy reducido, con una tasa de transmisión de aproximadamente 250Kbps y un alcance de kilómetros.

- La tecnología con protocolo **Bluetooth** está basada en el estándar IEEE 802.15.1, utiliza la banda de frecuencia libre de 2,4 GHz, estableciendo la tasa máxima de transmisión en 3Mbps con un rango óptimo de alcance de 10 metros. Su uso está orientado para dispositivos de bajo consumo, con una cobertura baja y basada en receptores de bajo coste.

- La tecnología **Wi-Fi** utiliza las bandas de frecuencia libre de 2,4 Ghz y 5 Ghz, estableciendo la tasa máxima de transmisión en 108 Mbps (IEEE 802.11n) con una distancia máxima sin amplificación de aproximadamente 100 metros.

- La tecnología **WiMAX** está basada en el estándar IEEE 802.16, utiliza diferentes frecuencias según cada protocolo dentro de la familia (desde los 2GHz hasta los 66GHz). La máxima tasa de transmisión es de hasta 70 Mbps, con una cobertura de hasta 50 km aproximadamente.

2.2. Dispositivo Waspmote

Este proyecto trabaja con el dispositivo *Waspmote*, fabricado y distribuido por Libelium Comunicaciones Distribuidas S.L.

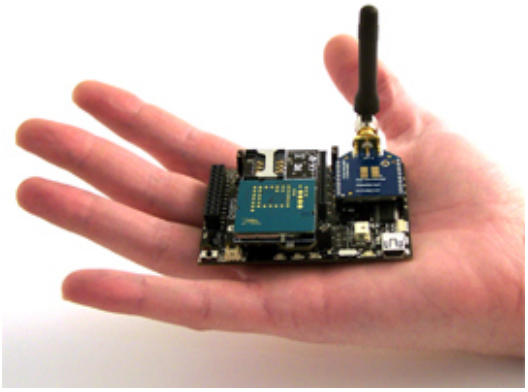

Figura 2.1: Dispositivo Waspmote

Waspmote es un dispositivo sensorial especialmente orientado a desarrolladores. Puede trabajar con diferentes protocolos (ZigBee, Bluetooth, GPRS) y frecuencias (2.4GHz, 868MHz, 900MHz) siendo capaz de establecer enlaces de hasta 12km. Cuenta con un modo de hibernación con un consumo de 0.7 µA que permite ahorrar batería cuando no está transmitiendo.

El dispositivo *Waspmote* dispone también de más de 50 sensores como temperatura, humedad o radiación entre otros, y de un entorno de desarrollo (IDE) completamente open-source (IDE + librerías + compilador).

El propósito de este dispositivo es facilitar la configuración de redes sensoriales en multitud de escenarios diferentes, desde una red de contadores industriales o una red de sensores en cultivos, hasta una red de sensores en un lugar público.

A continuación se observa dos esquemas de la organización hardware y software. En la figura 2.2. se observa cómo diferentes módulos se acoplan al dispositivo *Waspmote* a través de distintas conexiones. Y la figura 2.3. muestra cómo están estructuradas las bibliotecas en el entorno de desarrollo.

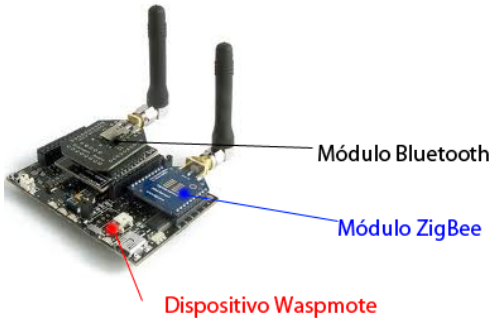

Figura 2.2: Organización hardware

Figura 2.3: Organización software

En el anexo A se explica con más detalle el diseño del dispositivo *Waspmote*.

2.3. Topologías Wi-Fi

El objetivo de este proyecto es la realización del módulo Wi-Fi para el sistema de redes sensoriales *Waspmote*. Dicho sistema ya dispone de módulos *ZigBee* y Bluetooth entre otros, por lo que el objetivo que se persigue en este módulo es optimizar las características que ofrece la tecnología Wi-Fi.

En este apartado se presentan las topologías disponibles utilizando el módulo Wi-Fi.

2.3.1. Infraestructura con Punto de Acceso

En esta topología los módulos Wi-Fi pueden conectarse a cualquier *router* estándar que esté configurado como Punto de Acceso (AP) y entonces enviar información a otros dispositivos que estén operando en la misma red como ordenadores portátiles o teléfonos inteligentes (Smartphone).

Esta es la topología más común cuando se implementan redes sensoriales en entornos pequeños y cuando se usan los datos en una Intranet.

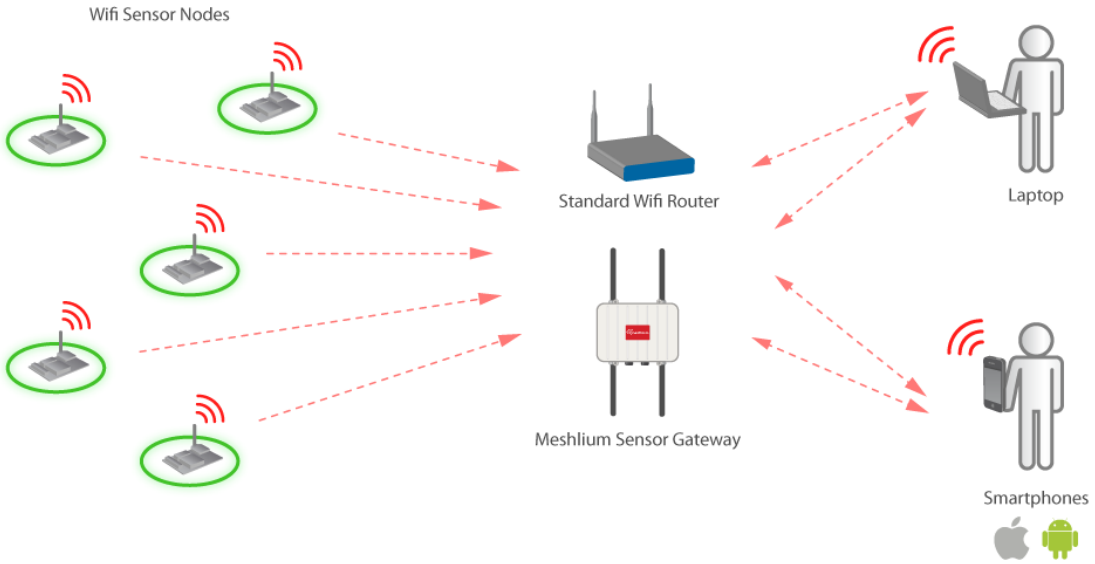

Figura 2.4: Esquema topología con Punto de Acceso

Los nodos también pueden conectarse a un *router* Wi-Fi estándar que disponga de conexión a Internet y enviar la información recogida a un servidor web en Internet. Entonces los usuarios pueden recibir esta información de la nube. Este es el escenario tipo para compañías que quieren dar servicios de información accesible a sus clientes.

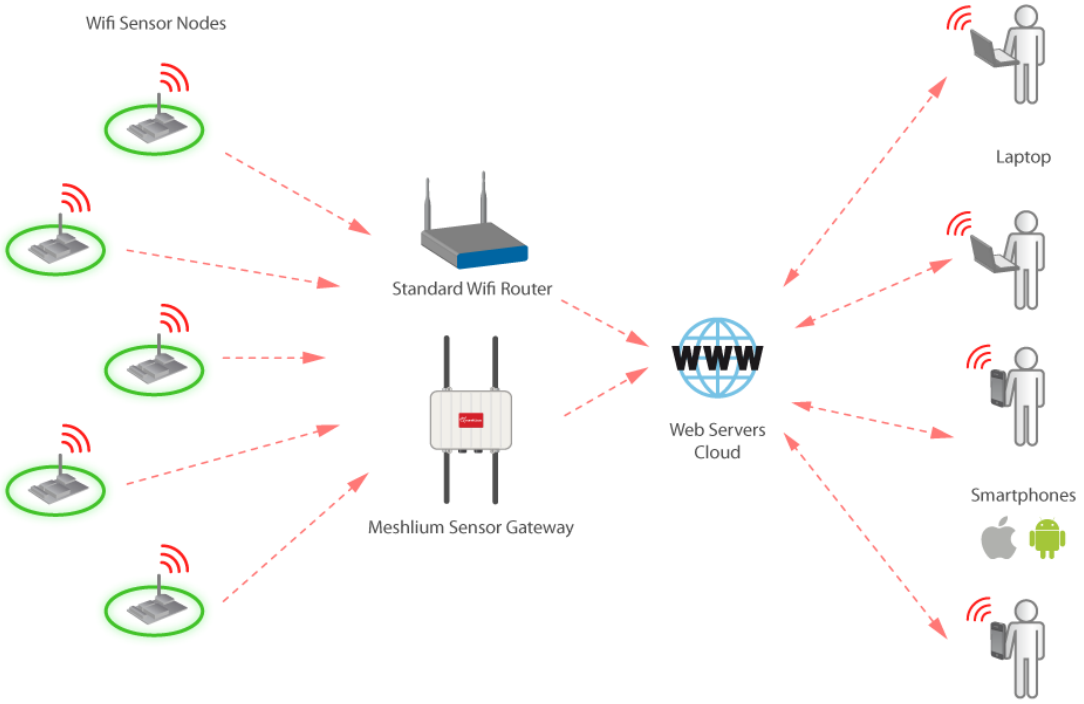

Figura 2.5: Esquema topología con Punto de Acceso e Internet

Como se ha indicado antes, el módulo Wi-Fi puede enlazarse a cualquier router estándar, sin embargo, la conexión puede ser mejorada usando *Meshlium [10]* en lugar de un router estándar.

Meshlium es un *router* multiprotocolo diseñado por Libelium que es especialmente recomendado para escenarios al aire libre, ya que está diseñado para soportar condiciones climáticas adversas. También ofrece otras prestaciones y funcionalidades como el guardar y organizar los datos recibidos en una base de datos.

Figura 2.6: router Meshlium.

2.3.2. Ad-hoc con dispositivo móvil

El siguiente diagrama muestra cómo un móvil inteligente (Smartphone) puede también comunicarse directamente con el módulo Wi-Fi integrado en cada *Waspmote* a través de una red Ad-hoc sin necesidad de un router o puerta de enlace.

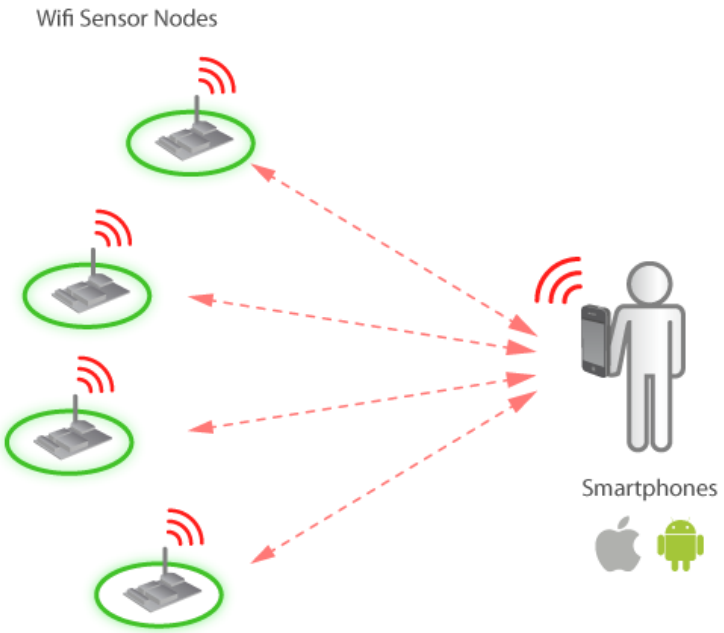

Figura 2.7: Esquema topología Ad-hoc con dispositivo móvil.

2.3.3. Ad-hoc entre Waspmote

La topología Ad-hoc puede ser usada también entre una colección de dispositivos *Waspmote* creando una red punto a punto en la que cada módulo Wi-Fi es enlazado a todos los demás dispositivos Wi-Fi en la red, ya sea mediante comunicación directa o retransmisión. Como en el caso anterior, no existe punto de acceso.

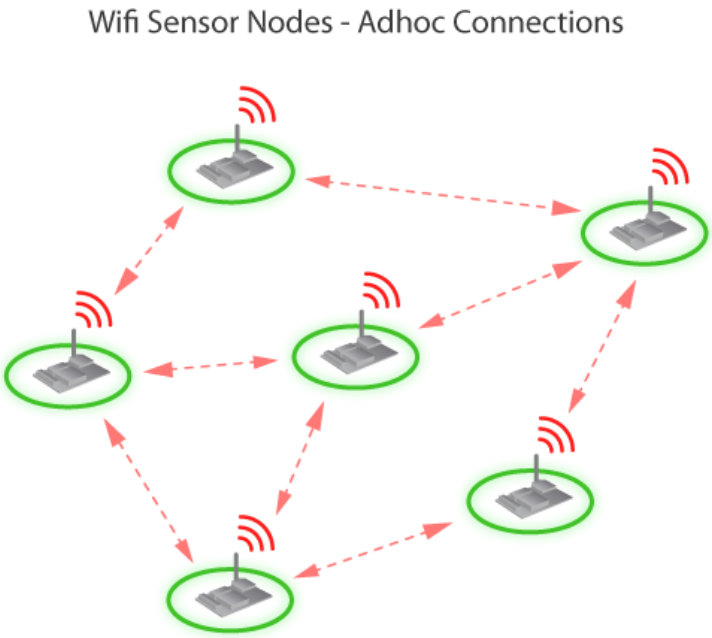

Figura 2.8: Esquema topología Ad-hoc entre Waspmote.

Capítulo 3

Diseño Hardware

3.1. Contexto Hardware

Los desarrollos de este proyecto se realizan en torno al dispositivo *Waspmote,* el cual se explica en detalle en el Anexo A. He aquí un pequeño resumen de las principales características hardware de dicho dispositivo:

- Cuenta con un microprocesador *ATMEGA 1281* [11] programable utilizando un lenguaje de programación basado en C++.

- Dispone de 10 puertos de entrada/salida digitales para poder conectar sensores externos, así como 8 entradas analógicas. De esta forma obtenemos una gran cantidad de posibilidades de interconexión para nuestras redes sensoriales.

- También dispone de dos conexiones de comunicación serie asíncrona (UART) seleccionables y de un puerto síncrono (SPI).

Figura 3.1: Dispositivo Waspmote

3.2. Selección de integrados Wi-Fi

Nuestra tarea ha consistido en seleccionar un integrado que, incorporado al dispositivo *Waspmote*, le dote de capacidad de comunicación Wi-Fi.

El ideal de los radios de comunicación en los dispositivos empotrados es permanecer el mayor tiempo posible en estado de bajo consumo o dormido, y despertarse para realizar una operación sólo cuando sea necesario.

La empresa ha preseleccionado dos integrados Wi-Fi compatibles, en cuanto a conectividad y control con las características del dispositivo *Waspmote*.

En el mercado hemos encontrado dos que satisfacían tales requisitos, *WiFly RN-171 de Roving Networks* [12] y *RS9110N11 de RedPine Signals* [13]. El integrado de *Roving Networks* puede comunicarse mediante puerto UART y el de *RedPine Signals* puede comunicarse tanto por UART como SPI.

Figura 3.2: Integrado Roving Networks

Figura 3.3: Integrado RedPine Signals

Los dos integrados son similares, por lo que se acota la comparativa a potencia de transmisión, rendimiento máximo de comunicación con micro controlador, protocolos disponibles y consumo. En este estudio no se ha considerado el precio de cada integrado al caer fuera del alcance de este proyecto, ya que las consideraciones económicas fueron analizadas previamente por la empresa.

En la tabla 3.1 se observa la comparativa de dichas funcionalidades:

	Roving Networks	**Redpine Signals**
Protocolo Wi-Fi	802.11 b/g	802.11 b/g/n
Potencia de Transmisión	0 dBm a 12 dBm	18 dBm
Rendimiento	UART: 3 Mbps	UART: 2 Mbps SPI: 8 Mbps
Encriptación Wi-Fi	Open, WEP-40, WEP-128, WPA-PSK, WPA2	Open, WEP, WPA, WPA2, WPS
Protocolos de Red	TCP, DHCP, UDP, DNS, ARP, ICMP, FTP, HTML, WPS, Ad-hoc	TCP, DHCP, UDP, ARP, ICMP
Consumo Estado dormido Recibiendo Transmitiendo	 4 µA 36 mA 38 mA	 0.52 mA 24 mA 30 mA

Tabla 3.1: Tabla comparativa de integrados Wi-Fi

En conclusión, el consumo en estado dormido es menor en el integrado de *Roving Networks* (4 µA vs 520 µA), y además es más configurable y soporta más protocolos de red. Por ello, hemos optado por seleccionar el integrado de **Roving Networks WiFly RN-171**.

3.3. Módulo Wi-Fi definitivo

El integrado Wi-Fi elegido tiene conectividad por comunicación serie asíncrona (UART). El dispositivo *Waspmote* dispone de dos conexiones UART, a una se accede directamente y a la otra se accede mediante una placa de expansión (*Expansion Radio Board*) [14].

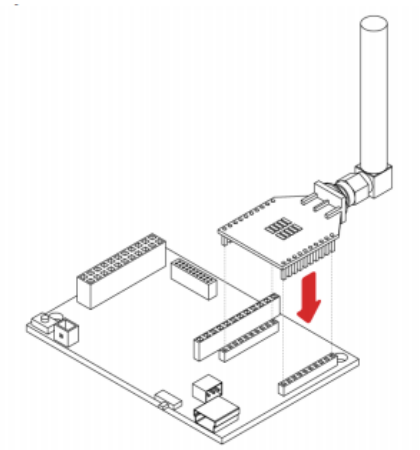

Figura 3.4: Puerto UART principal

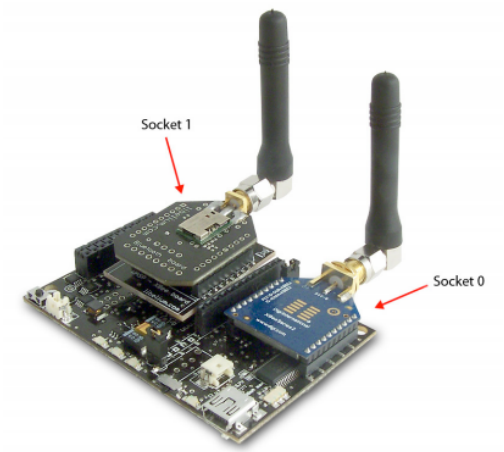

Figura 3.5: Waspmote con Expansion Board

Al tener esta placa de expansión, se pueden utilizar dos sockets para conectar los radios y hacer que funcionen en paralelo. Para acceder a ellos, se selecciona el puerto UART en el código. Dicha placa permite hacer combinaciones usando cualquiera de los seis radios disponibles para *Waspmote*: 802.15.4, ZigBee, Bluetooth, RFID, GPRS, y ahora Wi-Fi.

Para la integración del módulo Wi-Fi, se ha tenido en cuenta los pines necesarios para la alimentación y los necesarios para la comunicación entre el módulo y el *Waspmote*. El integrado *WiFly RN-171* necesita dos pines de alimentación (3,3 V y GND), y otros dos para la comunicación de la UART (TXD, RXD).

Para ello, primero se encaminan dichas conexiones a los pines correspondientes para una conexión efectiva con el dispositivo *Waspmote*. Los detalles se describen en el Anexo B.

Y finalmente, se añade un conector *SMA* [15] a la entrada de antena, con esto se permite la conexión de una gran variedad de antenas dependiendo de la necesidad.

En la figura 3.6 se observa el resultado final del módulo Wi-Fi con dos tipos distintos de antenas.

Figura 3.6: Libelium módulo Wi-Fi definitivo con antenas de 2 dBi y de 5 dBi

Capítulo 4

API Módulo Wi-Fi

4.1. Funcionamiento módulo Wi-Fi

Como ya se ha comentado, la comunicación con el radio Wi-Fi se realiza mediante el puerto UART del *Waspmote*. Controlando este puerto, se puede configurar el radio Wi-Fi, crear una conexión y enviar datos a través de esa conexión.

Este tipo de integrados son configurados mediante órdenes recibidas a través del canal de comunicación UART. Este integrado dispone de dos modos: modo comando y modo transparente. Los mensajes recibidos a través del canal de comunicación son interpretados de forma distinta dependiendo del modo en el que se encuentre el módulo.

- En modo **comando**, se configura el radio Wi-Fi, ya sean los datos de una conexión o cualquier otro comportamiento interno del radio Wi-Fi. Esto se hace mediante órdenes que se reciben por la línea serie con un formato determinado. Por ejemplo, la orden *"SET I D 1<cr>"* activa el protocolo DHCP.

- El modo **transparente** le dota de un comportamiento similar a un repetidor, ya que todos los datos procedentes del radio Wi-Fi se envían por la línea serie al micro controlador y los datos procedentes de la línea serie se envían por el radio Wi-Fi. A este modo se accede cuando ya está configurada una conexión y se quiere mandar un mensaje.

Al alimentar el integrado se activa el modo transparente por defecto, por lo que entraremos en modo comando (mandando *$$$* al integrado sin retorno de carro) en nuestra función de inicialización.

La biblioteca que se ha desarrollado para la programación del módulo Wi-Fi proporciona distintas funcionalidades, de las cuales los siguiente aspectos se abstraen de cara al usuario/programador final:

- Controlar la **velocidad** de los datos recibidos y enviados a través del puerto serie al radio Wi-Fi, vaciando el buffer del puerto serie antes de que se llene y manteniendo retardos para sincronizarlo con el micro controlador de *Waspmote*.

- Cambios de **modo**. El cambio a modo transparente es necesario cuando se realiza una conexión. Y cuando se inicia o reinicia el dispositivo se necesita cambiar a modo comando si queremos cambiar la configuración.

- Pulsos de **tensión y reinicios** del radio necesarios. Para iniciar el radio Wi-Fi hay que enviar un pulso de tensión (alto-bajo-alto), y en los casos de auto conexión, se necesita guardar los datos de la conexión y reiniciar el radio Wi-Fi.

- Agrupar **comandos**. El integrado dispone de multitud de comandos que se han agrupado y se envían de manera secuencial dentro de las funciones para realizar la tarea correspondiente.

4.2. Bibliotecas Waspmote

Para la programación del dispositivo *Waspmote* se ha utilizado un lenguaje de programación basado en C/C++ que fue desarrollado para utilizar el dispositivo *Arduino* [16]. Es un lenguaje de programación basado en *Wiring* [17], open-source programming framework para micro controladores

Esto es posible debido a que *Waspmote* se comunica mediante la transmisión de datos en formato serie, algo que los lenguajes C , C++ soportan. Admite todas las funciones del estándar C y algunas de C++.

El siguiente fragmento de código sería un ejemplo tipo de un programa para *Waspmote*:

```
# define LED_PIN 13
// Método de Inicialización del dispositivo
void setup () {
        // Se activa el pin 13 para salida digital
        pinMode (LED_PIN, OUTPUT);
}

// Método que se itera indefinidamente. Enciende y apaga LED_PIN.
void loop () {
        // Se enciende el led enviando una señal alta
        digitalWrite (LED_PIN, HIGH);
        // Esperamos un segundo (1000 ms)
        delay (1000);
        // Se apaga el led enviando una señal baja
        digitalWrite (LED_PIN, LOW);
        // Espera un segundo (1000 ms)
        delay (1000);
}
```

El orden de ejecución será: Primero se ejecuta *setup()* que configura LED_PIN, y a continuación se realiza un bucle que ejecuta repetidamente el método *loop().*

La estructura de las **bibliotecas** para *Waspmote* se compone de un archivo C++ (.cpp) y una cabecera (.h) donde se definen las funciones. El archivo .cpp contiene el código de las funciones declaradas en la cabecera, y en la cabecera se crea el objeto que es utilizado para llamar a las funciones correspondientes.

El siguiente código *WaspEjemplo* ejemplifica el funcionamiento de las bibliotecas:

```
//   WaspEjemplo.h – Cabecera de Biblioteca de ejemplo.
class WaspEjemplo
{ public:
    función_1(int pin);
    void función_2();
  private:
    int _pin;
};
extern WaspEjemplo Ejemplo;
```

```
//   WaspEjemplo.cpp – Biblioteca de ejemplo
# include "WaspEjemplo.h"
//   Método función_1. Configura pin.
WaspEjemplo:: función_1 (int pin)
{
    pinMode(pin, OUTPUT);
    _pin = pin;
}
//   Método función_2. Alterna el pin con señales HIGH y
//   LOW cada 250 milisegundos
void WaspEjemplo::función_2()
{
    digitalWrite(_pin, HIGH);
    delay(250);
    digitalWrite(_pin, LOW);
    delay(250);
}
```

A continuación se muestra un fragmento de código con dos invocaciones a los métodos función_1 función_2 del objeto Ejemplo, una instancia de la clase *WaspEjemplo*.

```
Ejemplo.función_1(23); Ejemplo.función_2();
```

Una restricción es el consumo de **recursos**, por ello, hay que tener en cuenta que al ser un sistema empotrado, tenemos recursos hardware muy limitados.

Por ello hay que minimizar el uso de memoria, ello se logra:

- Economizando memoria: Usando tipos de datos que consuman poca memoria cuando sea posible, por ejemplo *uint8_t* en lugar de *int.*
- Usando memoria dinámica y liberándola cuando sea posible.
- Controlando el tamaño del buffer de los puertos: nunca se deben llenar y cuando ya hemos tratado un dato, se debe eliminar del buffer.

4.3. Métodos de la biblioteca

La Biblioteca desarrollada dentro de este proyecto se denomina biblioteca ***WaspWIFI***, y tiene un formato tipo (archivos *WaspWIFI.h* y *WaspWIFI.cpp* y el objeto *WIFI*) como el resto de bibliotecas *Waspmote,* el cual se ha explicado en el apartado anterior.

La biblioteca se ha estructurado en dos **niveles**. Por una parte, se han desarrollado ocho métodos de bajo nivel. Estos métodos tienen carácter privado y dan soporte a los métodos públicos de la biblioteca. A un nivel superior están los métodos públicos, a los que me voy a referir a partir de ahora como funciones como carácter general, que facilitan al programador la configuración y uso del módulo Wi-Fi.

Además hay dos **variables** de objeto. Una pública llamada *answer*, en la que se almacena los últimos datos recibidos por el módulo Wi-Fi. De esta manera podemos vaciar el buffer hardware para tratar dicho mensaje.

Y en la parte privada se encuentra la variable *question*, en la cual se almacenan datos para enviar al radio Wi-Fi, (un comando, en modo comando, o datos, en modo transparente).

Los **métodos privados** son:

- *commandMode,* entrar en modo comando.
- *contains*, fragmentar un texto del buffer de línea serie y compararlo con cadenas preestablecidas.
- *readData*, leer datos por línea serie.
- *sendCommand*, enviar un comando al radio Wi-Fi.
- *saveReboot*, guardar configuración actual y reiniciar el integrado para que las nuevas configuraciones tengan efecto.
- *open*, abrir una conexión (TCP, UDP, FTP por ejemplo) y poner el radio Wi-Fi en modo transparente
- *openHTTP*, abrir una conexión HTTP (comando GET a un servidor).
- *parseBroadcast*, comprobar la información de una trama Broadcast.

Los métodos públicos están relacionados con los métodos privados como muestra la figura 4.1:

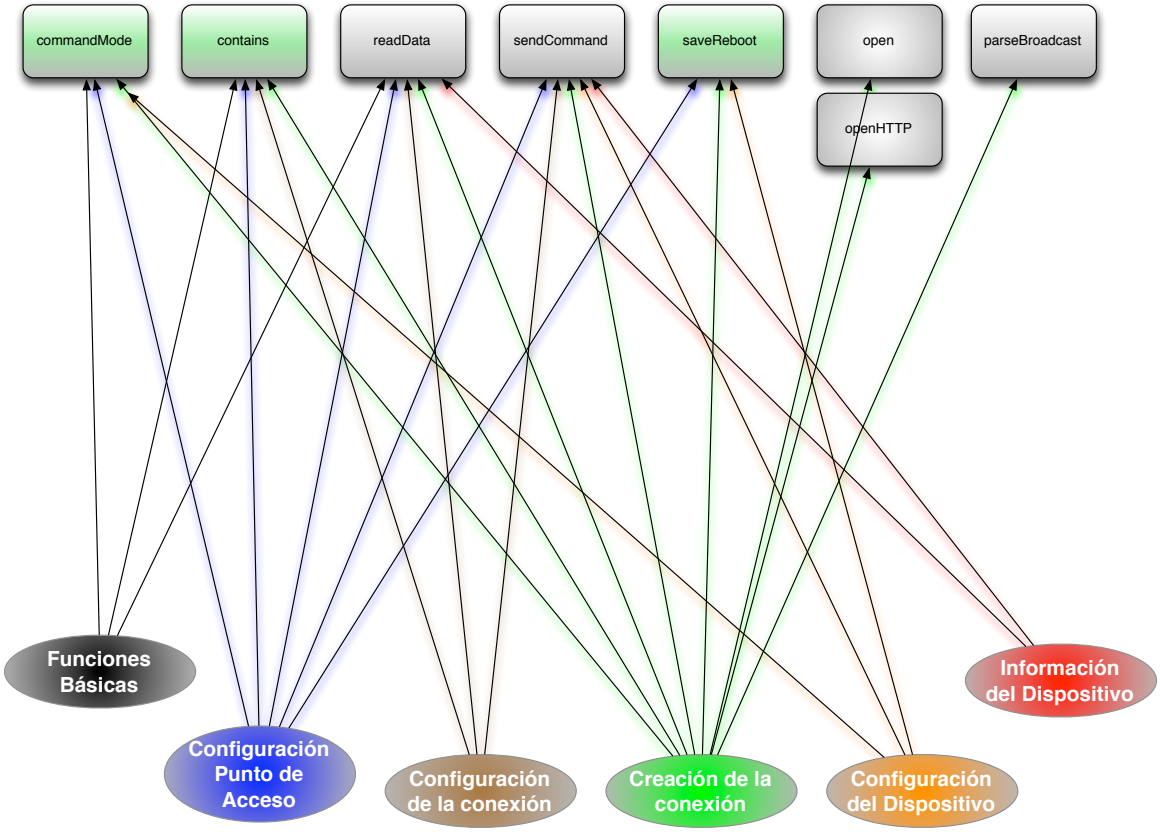

Figura 4.1: Diagrama de la clase WaspWiFi

En la Figura 4.1 se observan dos partes:

- En la parte superior, están los métodos privados. Dichos métodos representan funcionalidades generales del radio Wi-Fi, y están relacionadas con los grupos de funciones de abajo. Observando estas relaciones, se puede intuir el funcionamiento de la clase.

- Y en la parte inferior están las métodos públicos, agrupados dependiendo de su funcionalidad. A dichas funciones se accede con el objeto de la clase en el programa Waspmote, por ejemplo, *WIFI.ON()*.

Las **funciones públicas** desarrolladas se clasifican en:

- *Funciones básicas*: Estas funciones son el constructor de la clase, funciones para inicializar el módulo y para encenderlo y apagarlo.
  Algunos ejemplos ilustrativos: *begin(), ON().*

- *Configuración del Punto de Acceso*: Aquí se configura la topología de red, el tipo de punto de acceso, las características de dicho punto y también se dan funciones para conectar y desconectar de dichos puntos.
  Algunos ejemplos ilustrativos: *setJoinMode(), join(), setAdhocOptions().*

- *Configuración de la conexión*: En este apartado están las funciones encargadas de configurar las conexiones, las direcciones IP, los puertos, máscaras de red, etc.
  Algunos ejemplos ilustrativos: *setDHCP(), setGW(), setIP(), setNetmask(), setLocalPort().*

- *Creación de la conexión*: Aquí están las funciones necesarias para crear la conexión para cada tipo de protocolo (TCP [18], UDP [19], Broadcast-UDP [20], HTML [21], FTP [22]), y también para cerrar la conexión.
  Algunos ejemplos ilustrativos: *setTCPClient(), send(), read(), close(), getURL(), openFTP(), sendAutoBroadcast().*

- *Configuración del dispositivo*: Estas funciones permiten utilizar el dispositivo de manera eficiente, encontraremos cómo reiniciar el dispositivo, ponerlo en modo dormir, despertarlo, ajustar la velocidad de la UART, guardar configuraciones por defecto, etc.
  Algunos ejemplos ilustrativos: *reset(), resetValues(), setBaudRate(), setTXPower(), setSleep().*

- *Información del dispositivo*: Estas funciones devuelven información de la configuración actual del módulo Wi-Fi, la información se transmite por el puerto del USB que tiene el dispositivo Waspmote.
  Algunos ejemplos ilustrativos: *getConnectionInfo(), getStats(), getAPstatus().*

4.3.1. Inicialización

Antes de usar el módulo Wi-Fi, éste necesita ser inicializado, durante este proceso, los parámetros de configuración son enviados al radio Wi-Fi. También se inicializan otras bibliotecas que son utilizadas, como son *USB* para el caso de mostrar información del dispositivo y *SD* en el caso de subir o descargar archivos a través de conexiones FTP utilizando una tarjeta SD acoplada al *Waspmote*.

La inicialización de la biblioteca *WaspWIFI* se hace a través de la función ***begin()***, con la cual el *Waspmote* se prepara para comunicarse con el módulo Wi-Fi, inicializando las bibliotecas necesarias.

Y llamando a ***ON(socket)***, el módulo abre los pines necesarios para la comunicación, y también configura la velocidad de transmisión del puerto UART. Dependiendo del socket seleccionado, se cambia un multiplexor que prepara *Waspmote* para comunicar por el UART principal o utilizando la placa de expansión.

A continuación se muestra un fragmento del código que selecciona el multiplexor dependiendo del valor de socket.

```
if (socket==socket0)
{     // Puerto UART principal
      // Se configuran como salida (OUTPUT) los pines del multiplexor UART.
      pinMode(XBEE_PW, OUTPUT);
      pinMode(MUX_USB_XBEE, OUTPUT);
      // Se alimentan dichos pines para seleccionar la entrada de UART
      digitalWrite(XBEE_PW, HIGH);
      digitalWrite(MUX_USB_XBEE, HIGH);
}
else
{     // Puerto UART de la placa de expansión
```

```
        // Se configuran como salida (OUTPUT) los pines del multiplexor UART.
        pinMode(MUX_PW, OUTPUT);
        pinMode(MUX0, OUTPUT);
        pinMode(MUX1, OUTPUT);
        // Se alimentan dichos pines para seleccionar la entrada de UART
        digitalWrite(MUX_PW, HIGH);
        digitalWrite(MUX0, HIGH);
        digitalWrite(MUX1, HIGH);
}
```

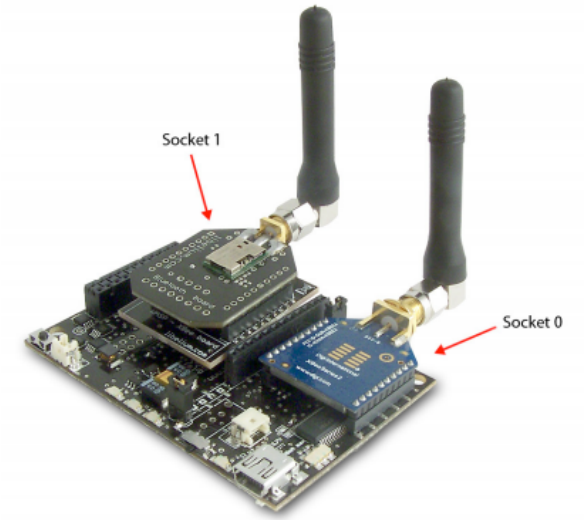

Figura 4.2: Waspmote funcionando con placa de expansión

4.3.2. Configuración Punto de Acceso

Aquí se describe como se conecta el módulo Wi-Fi a una red (Punto de Acceso) para hacer posible las conexiones deseadas.

La función principal en esta sección es ***setJoinMode(TIPO),*** dependiendo de TIPO el módulo se conecta a un punto de acceso de distinta forma. La función devuelve 1 si va todo bien, y 0 si algo ha fallado al conectarse a dicha red.
A continuación se muestra el comportamiento de la función de una forma más detallada dependiendo del argumento:

- Si tipo es ***MANUAL***, el módulo Wi-Fi no se conecta a ninguna red de forma automática. El usuario debe llamar a ***join*** de forma manual indicando el nombre de la ESSID y la contraseña de red si la hubiera cuando quiera conectarse a una red.

- Si tipo es ***AUTO_BEST***, se guarda la configuración actual en la memoria del integrado y reinicia el módulo Wi-Fi, el cual hace un barrido de redes para comunicarse e intenta conectarse con la red de mejor calidad teniendo en cuenta el nivel de encriptación si hubiera alguna contraseña guardada en el dispositivo.

- Si el tipo es **AUTO_STORE**, se guarda la configuración actual en la memoria del integrado y reinicia el módulo Wi-Fi, el cual hace un barrido de redes para comunicarse e intenta comunicarse con una red que tenga el mismo nombre y canal indicados previamente mediante las funciones correspondientes.

- Y finalmente, si el tipo es **CREATE_ADHOC**, el radio Wi-Fi crea una red Ad-hoc con el nombre de la red, la seguridad si la hubiera y el canal especificados previamente. Si dicha red ya existe, se unirá a dicha red mediante modo Ad-hoc. Existe otra función en la que el usuario también puede configurar el ratio y el tiempo de espera sin conexión de la red Ad-hoc ***setAdhocOptions(beacon, probe)***. Dicha conexión también crea la conexión Ad-hoc.

4.3.3. Configuración de Red

En esta sección se describe cómo configurar la red. El módulo Wi-Fi tiene que estar configurado antes de enlazarse a un punto de acceso o crear una red Ad-hoc.

La función principal es **setDHCPoptions**(TIPO), con la que configuramos el modo en el que se resuelve la dirección IP [23], necesaria para realizar cualquier tipo de comunicación con Internet. Los tipos posibles son: DHCP_OFF, DHCP_ON, AUTO_IP, y DHCP_CACHE.

Llamando a *setDHCPoptions(DHCP_ON)* se activa el protocolo DHCP [24]. Llamando a este protocolo, el módulo Wi-Fi recibe una dirección IP válida desde el punto de acceso automáticamente. Con DHCP_CACHE, el módulo Wi-Fi usa una dirección IP antigua válida, si la sesión DHCP no ha expirado.

Con DHCP_OFF se desactiva el protocolo DHCP, con lo que hay que indicar las direcciones de red manualmente. Un posible código sería el siguiente:

```
// 1. Desactiva protocolo DHCP.
WIFI.setDHCP(DHCP_OFF);
// 2. Configura la puerta de enlace
WIFI.setGW(“192.168.4.1”);
// 3. Configura la máscara de red
WIFI.setNetmask(“255.255.255.0”);
// 4. Configura el puerto local
WIFI.setLocalPort(55555);
// 5. Configura la dirección IP
WIFI.setIP(“192.168.1.50”);
```

Y finalmente, llamando a *setDHCPoptions(AUTO_IP),* el módulo Wi-Fi está configurado para resolver direcciones IP en redes Ad-hoc. Una dirección link-local (Auto-IP), es una dirección IP para comunicaciones locales o punto a punto. Dichas direcciones están definidas en el bloque de direcciones 169.254.0.0/16.

4.3.4. Conexión

Este apartado describe brevemente cómo configurar y preparar una conexión desde el módulo Wi-Fi dependiendo del protocolo deseado. Las posibles conexiones son *Ad-hoc, Broadcast-UDP, UDP, TCP, HTTP y FTP*.

Con la función ***setConnectionOptions(TCP|UDP|…)*** se configura el tipo o los tipos de protocolo si vamos a hacer una conexión HTTP, cliente TCP, UDP seguro, servidor TCP o UDP simple. En este caso el argumento es una máscara de 5 bits. De esta manera se puede configurar una conexión que soporte distintos protocolos a la vez utilizando el operador OR (|). La máscara de bits interna es la siguiente:

HTTP	TCP Cliente	UDP Seguro	TCP Cliente & Servidor	UDP
10000	01000	00100	00010	00001

Tabla 4.1: Máscara de bits del selector de protocolo

A la hora de crear la conexión, cada protocolo funciona de manera distinta, ya que hay protocolos orientados a conexión como TCP, y otros que no, como UDP. El protocolo HTTP funciona mandando mensajes de petición y esperando respuesta y en FTP se controla la conexión internamente. Por ello, se han creado funciones distintas para cada tipo de protocolo.

A continuación se explica el funcionamiento de cada protocolo:

- **TCP:** El Protocolo de Control de Transmisión o TCP, es uno de los protocolos fundamentales en Internet. Es un protocolo de comunicación orientado a conexión y fiable del nivel de transporte.

  Por ello, las funciones necesarias son:
  - Abrir la conexión ***setTCPclient***, ***setTCPserver***, indicando la dirección y el puerto.
  - Enviar o recibir ***send, read*** una vez establecida la conexión.
  - Y finalmente cerrar la conexión explícitamente ***close***.

  A continuación se muestra un ejemplo de uso de las funciones implementadas:

```
// Abre la conexión TCP
if (WIFI.setTCPclient("192.168.1.50",3550,2000))
{     // Si todo va bien, enviar y recibir mensajes.
      while(1)
      {     WIFI.send("xxxx"); WIFI.read();
            if (WIFI.answer[0]=='0') break;
      }
      // Cierra la conexión TCP
      WIFI.close();
}
```

En este caso, el módulo Wi-Fi abre conexión TCP, manda un mensaje y espera recibir otro, en el caso que empiece por cero, sale del bucle y cierra la conexión.

- **UDP:** El Protocolo de Datagramas de Usuario o UDP es un protocolo de nivel de transporte basado en el intercambio de datagramas. Permite el envío de datagramas a través de la red sin que se haya establecido previamente una conexión.

  Por ello, las funciones necesarias son:

  - Abrir la conexión ***setUDPclient***, ***setUDPserver***, indicando la dirección y el puerto.
  - Enviar o recibir ***send, read*** una vez establecida la conexión.
  - Y finalmente cerrar la conexión explícitamente ***close***, aunque en este caso debido a la naturaleza del protocolo no hace falta cerrar la conexión, es necesario indicar al módulo Wi-Fi que ya hemos terminado de enviar datagramas UDP al host determinado.

  A continuación se muestra un ejemplo de uso de las funciones implementadas:

```
// Selecciona la conexión UDP
if (WIFI.setUDPclient("192.168.1.50",3550,2000))
{     // Si todo va bien, enviar y recibir mensajes.
      while(1)
      {     WIFI.send("xxxx");
            WIFI.read();
            if (WIFI.answer[0]=='0') break;
      }
      // Sale de la conexión UDP
      WIFI.close();
}
```

  En este caso, el módulo Wi-Fi especifica modo UDP y la dirección del servidor, envía un datagrama UDP, y se pone en espera de recibir otro desde el servidor, en el caso de que este mensaje empiece por cero, sale del bucle y sale del modo mandar mensajes para continuar con otra tarea.

- **HTTP:** El protocolo de transferencia de hipertexto o HTTP es el usado en cada transacción de la *World Wide Web*.

  Por ello, para este protocolo tenemos una función ***getURL*** en la cual se especifica el servidor contra el cual se mandará la petición HTTP, y la petición GET. El servidor puede especificarse con su dirección IP, o con una URL, la cual será resuelta usando protocolo DNS.

  A continuación se muestra un ejemplo de uso de las función implementada:

```
// Llama a funcion HTTP especificando IP y Query
WIFI.getURL(IP,”84.20.10.73”,”GET$/radioWiFitest.php?name=Joaquin”);
// Llama a funcion HTTP especificando URL y Query
WIFI.getURL(DNS,”www.libelium.com”,”GET$/radioWiFitest.php?name=Jok”);
```

- **FTP:** El protocolo de transferencia de Archivos o FTP es un protocolo de red para la transferencia de archivos entre sistemas conectados a una red TCP, basado en la arquitectura cliente-servidor.

  Una transferencia FTP se puede desarrollar en tres etapas, por ello, se desarrollan las funciones:
  - Especificar el nombre y contraseña del servidor, ***openFTP***.
  - Subir un archivo, ***uploadFile***.
  - Descargar un archivo desde el servidor, ***getFile***.

  Las funciones de abrir conexión, cerrar conexión, etc. son abstraídas dentro de las funciones. En el caso de las funciones de subir y descargar archivo, se ha utilizado la API de la tarjeta SD para leer y escribir archivos en la tarjeta.

  A continuación se muestra un ejemplo de uso de las funciones implementadas:

```
// Prepara el servidor FTP especificando nombre de usuario
// y contraseña
WIFI.openFTP(“WiFimodu”,”cro3N2C20h”);
// Sube un archivo de la tarjeta SD al servidor FTP indicando
// nombre del documento, carpeta tarjeta SD y carpeta servidor FTP
WIFI.uploadFile(“FileText”,”Folder”,”public_ftp/Folder”);
// Descarga un archivo del servidor FTP a la tarjeta SD indicando
// nombre del documento, carpeta tarjeta SD y carpeta servidor FTP
WIFI.getFile(“documento”,”Folder”,”public_ftp”);
```

- **Broadcast-UDP:** Esta funcionalidad genera datagramas UDP automáticamente cada cierto tiempo, y puede trabajar en paralelo con cualquier tipo de conexión abierta. Esto es útil debido a lo siguiente:

  - Algunos puntos de acceso desconectan dispositivos si no reciben paquetes transcurrido un tiempo.
  - Programas de auto-conexión o auto-descubrimiento pueden utilizar esta utilidad cuando por ejemplo pasan de modo dormido a modo encendido.

  El formato del paquete es un paquete de 110 bytes de datos en el cual se especifican la dirección MAC, canal, puerto, etc. y un mensaje que se puede personalizar para cada dispositivo.

  A continuación se muestra un ejemplo de configuración de Broadcast-UDP utilizando la función ***sendAutoBroadcast***.

```
// Configura generación de mensajes UDP contra la dirección Broadcast
// y el puerto 55555  cada 7 segundos y mensaje “Mensaje”
WiFi.sendAutoBroadcast(“255.255.255.255”,55555,0x7,”Mensaje”);
```

4.4. Configuración módulo Wi-Fi

El módulo Wi-Fi puede ser configurado para optimizar el rendimiento del mismo en cuanto a velocidad, alcance y consumo.

En el caso de configuraciones, para volver a la configuración por defecto se debe llamar a ***resetValues***.

También se puede reiniciar el módulo manualmente llamando a ***reset*** (necesario para que una configuración determinada surja efecto), o guardar la configuración actual del dispositivo para que no cambie cuando se reinicie llamando a ***storeData***. Dicha configuración actual se guardará en el integrado Wi-Fi y aunque se reinicie el módulo, no cambiará a no ser que se reconfigure o se llame a ***resetValues***.

4.4.1. Funciones de optimización

Para optimizar el funcionamiento del módulo Wi-Fi se han programado una serie de funciones, que permiten definir:

- Velocidad de comunicación del radio Wi-Fi con el micro controlador, ***setBaudRate***, indicando la velocidad en Baudios (bps).
- Ratio de transmisión, ***setTXRate***, indicando el ratio, de 1 a 54 Mbit/sec.
- Potencia de transmisión, ***setTXPower***, indicando la potencia, de 1 a 12dBm.
- Tiempo de espera de una conexión, ***setJoinTime***, en milisegundos.

A continuación se muestra un ejemplo en el que se usan dichas funciones:

```
// Establece la ratio de comunicación de datos con el micro controlador
WIFI.setBaudRate(115200);
// Establece la ratio de transmisión de datos con la red
WIFI.setTXRate(15);
// Establece la potencia de transmisión
WIFI.setTXPower(12);
// Establece el timeout, en milisegundos, de una conexión a una red
WIFI.setJoinTime(100);
```

4.4.2. Funciones de ahorro energético

También es posible poner el módulo Wi-Fi en un estado dormido para ahorrar energía cuando no se esté utilizando. El radio Wi-Fi usa un reloj interno para generar tiempos. Dicho reloj está activo incluso cuando el radio Wi-Fi está en modo dormido. Esto hace posible poner el módulo a dormir y despertarlo utilizando intervalos de tiempo. Durante el tiempo de dormir, cualquier dato que sea enviado al radio Wi-Fi no será procesado.

Esto se puede hacer explícita o automáticamente. El usuario puede programar poner el radio Wi-Fi a dormir o despertar manualmente con las funciones ***sleep*** y ***wake***, o automáticamente con la función ***setSleep***.

Un ejemplo de uso automático se muestra a continuación en la figura 4.15:

```
// Se configuran los datos de la red
WIFI.setESSID("TEST_Wi-Fi");
// Se configura una conexión automática consistente en enviar un
// mensaje UDP con el texto "estoy despierto" cada 7 segundos.
WIFI.setAutoBroadcast("192.168.1.150",2500,0x7,"Estoy despierto");
// Se configura el radio Wi-Fi para estar despierto durante 30 segundos y
// dormido durante 90 segundos
WiFi.setSleep(30,90);
// Se reinicia el módulo Wi-Fi para que los cambios surjan efecto
WiFi.reset();
```

El proceso es el siguiente: El radio despierta, se conectará a la red "TEST_Wi-Fi" y mandará mensajes por UDP a la dirección IP indicada cada 7 segundos. Entonces *Waspmote* duerme el módulo Wi-Fi. Y pasados 90 segundos el módulo despierta.

4.5. Información del estado

Por último, están las funciones que indican el estado actual del radio Wi-Fi, útil para comprobar el correcto funcionamiento del módulo Wi-Fi. La información se envía por el puerto UART 0, que en el *Waspmote* está por defecto ocupado por el puerto USB, por lo que si el *Waspmote* está conectado al ordenador, se ven los resultados del puerto UART 0 en el *serial Monitor* del *Waspmote IDE*.

Esta información puede ser las configuraciones actuales de los distintos protocolos, la fuerza de la señal de una red, o las configuraciones hardware tipo UART, velocidad de transferencia.

El siguiente código muestra el estado de la conexión, del punto de acceso y la estadística de los paquetes enviados y recibidos.

```
// Muestra el estado de la conexión
WIFI.getConnectionInfo();
// Muestra el estado del punto de acceso
WIFI.getAPstatus();
// Muestra estadísticas de paquetes enviados y recibidos
WIFI.getStats();
```

La figura 4.3 muestra el resultado de utilizar el código anterior:

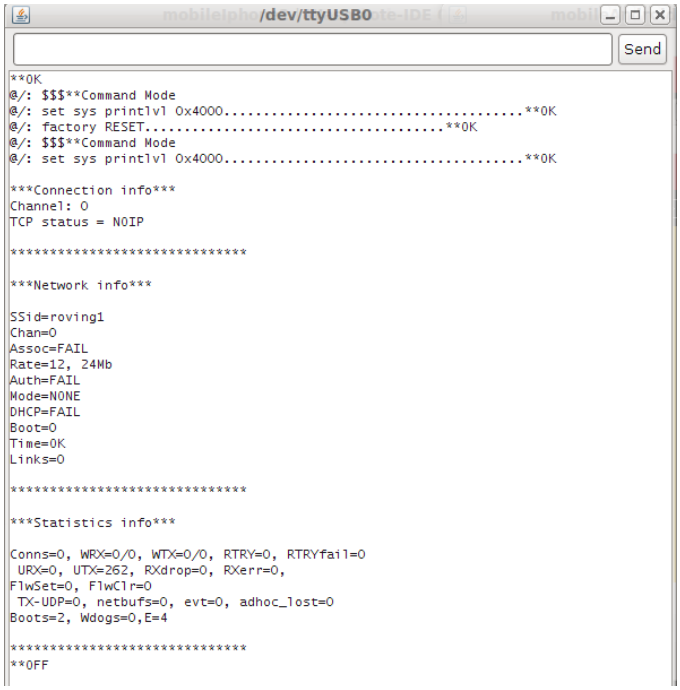

Figura 4.3: Salida de las funciones de estado del dispositivo utilizando Waspmote IDE

Capítulo 5

Pruebas Realizadas

5.1. Funcionamiento

Los módulos Wi-Fi desarrollados usan el estándar IEEE 802.11 b/g como base para la comunicación, añadiendo alguna funcionalidad extra a nivel de API. En esta sección se comprueba el funcionamiento de dichas funcionalidades. En el anexo D se encuentra una descripción completa de las pruebas realizadas.

5.1.1. Puntos de Acceso / router Wi-Fi

Para conectarse a un punto de acceso, es necesario primero hacer un escaneado para ver los puntos de acceso que están disponibles. Para ello en el caso de que esté la red encriptada hay que configurar la correspondiente clave.

ESCANEADO -> SELECCIONAR ENCRIPTACIÓN -> UNIRSE A LA RED

Es interesante conocer cuánto tiempo cuesta conectarnos a un punto de acceso, ya que para poner el radio Wi-Fi en modo dormido automáticamente debemos especificar los tiempos despierto y dormido.

Se ha probado con los puntos de acceso de distintos tipos de router (marcas *Linksys, Cisco y Zyxel*), y con el punto de acceso del dispositivo *Meshlium* (Libelium).

En el caso de encriptación, el módulo Wi-Fi soporta encriptación WEP [25], WPA [26] y WPA2 [27]. El tiempo de conexión al Punto de Acceso difiere dependiendo del tipo de encriptación de la red. A continuación se muestra una tabla con la comparativa de tiempos.

Encriptación	Tiempo (segundos)
Red abierta	3,997 s
WEP	4,077 s
WPA	4,176 s
WPA2	4,293 s

Tabla 5.1: Tabla comparativa de tiempos dependiendo de la encriptación.

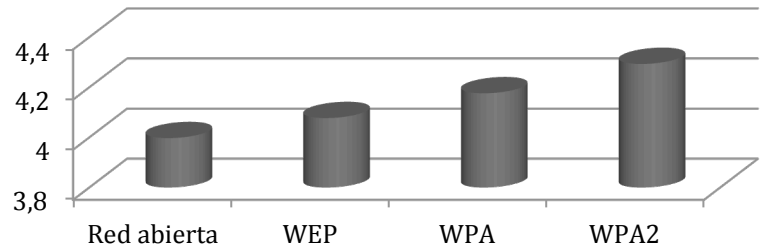

Figura 5.1: Gráfico comparativo de tiempos dependiendo de la encriptación.

El tiempo se contabiliza desde que se llama a la función ***join*** (enlazar al punto de acceso), hasta que se recibe dirección IP y el código está configurado con protocolo DHCP a ON.

5.1.2. Conexión con la nube (Internet)

El módulo Wi-Fi soporta distintos tipos de protocolos, por ello se hacen diferentes pruebas utilizando software adicional desarrollado para ello, con el objeto de comprobar el correcto funcionamiento del módulo desarrollado.

Para comprobar el correcto funcionamiento de los protocolos TCP y UDP se ha desarrollado un software multiplataforma en JAVA que crea sockets TCP y UDP, y permite comunicarse con el módulo Wi-Fi.

Para ello, se indica al módulo Wi-Fi la dirección IP del PC donde se ejecuta el programa y el puerto configurado. Entonces el módulo Wi-Fi se conecta al punto de acceso, en el caso de TCP abre la conexión, en el caso de UDP envía un datagrama, y en el caso de Broadcast-UDP configura la transmisión en broadcast de datagramas UDP.

A continuación se muestran los resultados del Test TCP:

- El radio Wi-Fi abre conexión TCP con el puerto 3550 del PC y envía "Hola desde el módulo Wi-Fi!!". El mensaje *HELLO* es un mensaje por defecto que envía el radio Wi-Fi cuando establece conexión TCP.

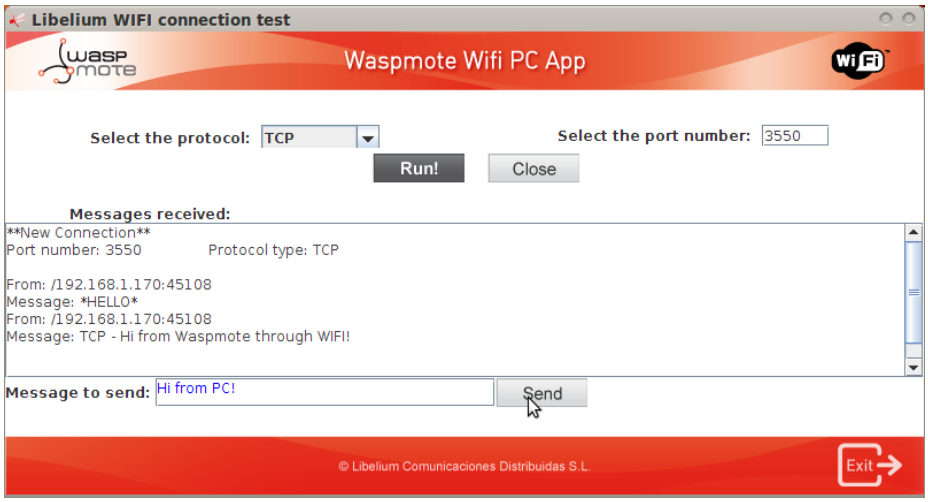

Figura 5.2: Resultados del test PC con protocolo TCP

Para los protocolos UDP, UDP Broadcast, HTTP y FTP también se han realizado pruebas, las cuales aparecen en el Anexo D.

5.1.3. Conexión entre módulos Wi-Fi

Los módulos Wi-Fi se comunican también mediante Wi-Fi Ad-hoc. En este caso cada módulo Wi-Fi crea una red Ad-hoc a la que envía o de la que recibe información.

En este caso, la prueba se ha realizado entre dos módulos Wi-Fi en modo Ad-hoc. El tiempo se contabiliza desde que se inicia el módulo (con la configuración cargada), hasta que se recibe dirección IP, y el código está configurado con protocolo DHCP a AUTO-IP.

Tipo	Tiempo (segundos)
Ad-hoc	9,7 s

Tabla 5.2: Tabla de tiempos conexión Ad-hoc.

5.1.4. Conexión con otro dispositivo Wi-Fi

A la hora de conectar con otro dispositivo Wi-Fi, lo que nos interesa es hacer una conexión punto a punto (Ad-hoc).

En el caso de ***iPhone o iPad***, el dispositivo se puede conectar o desconectar a la red Ad-hoc que crea el módulo Wi-Fi y coger información de la red sin modificar su funcionamiento.

En este caso, el radio Wi-Fi del *Waspmote* crea una red Wi-Fi Ad-hoc a la que comunica o recibe información periódicamente. A continuación el dispositivo iPhone o iPad se conecta a dicha red y a través de la aplicación desarrollada se comunica con el dispositivo Waspmote. Se muestra más información acerca de las aplicaciones en el capítulo 6, software asociado.

La siguiente imagen muestra a un *Waspmote* con módulo Wi-Fi comunicándose con un dispositivo *iPhone*:

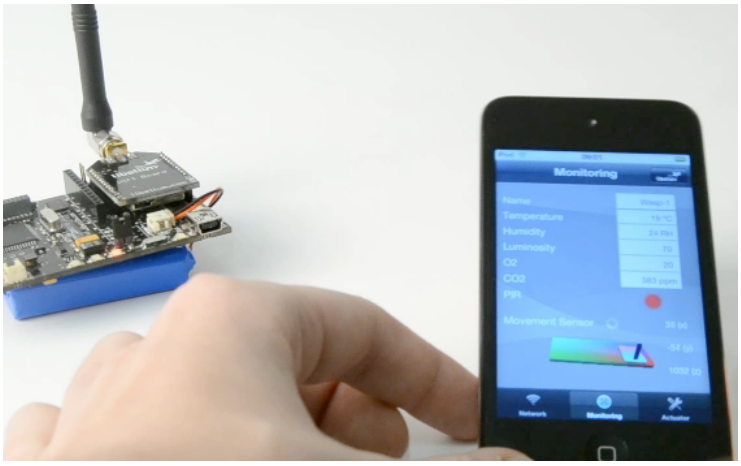

Figura 5.3: Resultados del test iPhone

Sin embargo, los dispositivos móviles con ***Android*** no pueden conectarse todavía a redes Ad-hoc. Pero a partir de *Android* 2.2 dichos dispositivos móviles permiten crear un punto de acceso.

Por ello, primero se configura en el móvil *Android* un punto de acceso. El radio Wi-Fi de *Waspmote* se configura para periódicamente conectarse a la red creada por el móvil y enviar o recibir información. La figura 5.7 muestra un dispositivo con Android y el módulo Wi-Fi comunicándose.

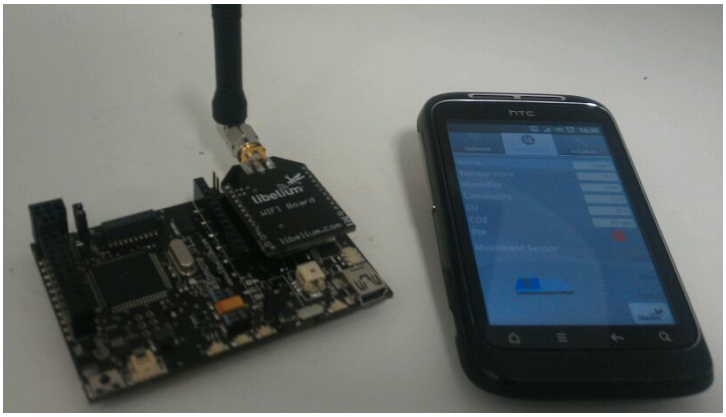

Figura 5.4: Resultados del test Android

5.2. Memoria RAM

El tamaño de memoria RAM empleado es importante, ya que el dispositivo *Waspmote* presenta limitaciones en este sentido (8 KBytes). Para hacer dicha medición, se hace una modificación del *Waspmote IDE* para que cuando compile muestre la memoria en la parte inferior de la pantalla.

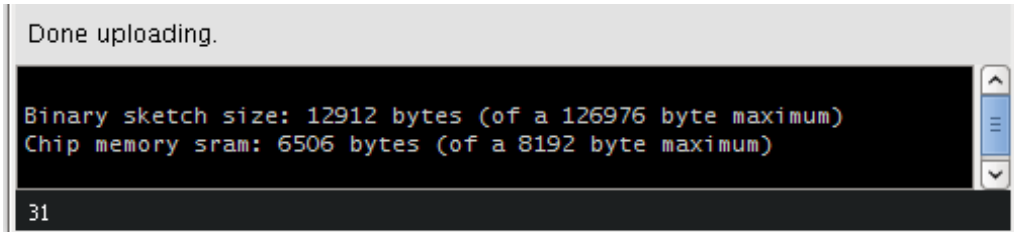

Figura 5.5: Waspmote IDE mostrando memoria RAM en la parte inferior del programa

A continuación se muestra una tabla comparativa de los programas de prueba desarrollados y el uso de memoria RAM en cada caso.

Estado del dispositivo	6,452 Kbyte
Conectar a un punto de acceso	6,464 Kbyte
Comunicación Ad-hoc	6,473 Kbyte
Envío datagramas Broadcast-UDP	6,494 Kbyte
Conexión TCP	6,506 Kbyte
Envío datagramas UDP	6,508 Kbyte
Conexión HTTP	6,528 Kbyte
Conexión FTP	6,925 Kbyte (usa API tarjeta SD)

Programa Android	7,680 Kbyte (usa API GPS)
Programa iPhone 1	7,678 Kbyte (usa API GPS)
Programa iPhone 2	6,492 Kbyte

Tabla 5.3: Tabla comparativa memoria RAM

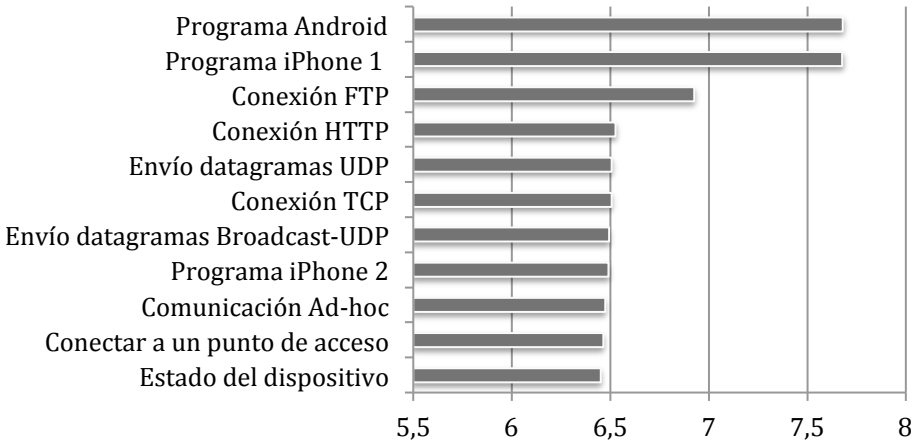

Figura 5.6: Gráfico comparativo memoria RAM

5.3. Ancho de Banda

Como ya se ha comentado, se puede cambiar la velocidad del módulo (bps). Se han hecho mediciones con distintos Baud-Rate [28] trabajando en un entorno real.

A continuación se muestra una tabla con los datos de velocidad de transmisión (del módulo Wi-Fi a un servidor de internet) y de recepción (de un servidor de Internet al módulo Wi-Fi), trabajando con 9600 bps, 19200 bps, y 57600 bps.

Baud-rates	Transmisión	Recepción
9600	142 ms 5,63 kbps	8 ms 100 kbps
19200	87 ms 9,19 kbps	8 ms 100 kbps
57600	51 ms 15,68 kbps	8 ms 100 kbps

Tabla 5.4: Tabla Ancho de Banda

5.4. Consumo

La adaptación realizada al integrado Wi-Fi para adaptarlo a *Waspmote* ha premitido reducir su consumo en 2 mA. En la siguiente tabla se muestra el consumo medio del módulo Wi-Fi en los principales estados del dispositivo.

Estado	Consumo
Apagado	0 µA
Dormido	4 µA
Encendido	33 mA
Recibiendo Datos	36 mA
Transmitiendo Datos	36 mA
Escaneando Puntos de Acceso	34 mA

Tabla 5.5: Tabla consumo

Para medir el consumo se ha utilizado un amperímetro conectado en serie en la alimentación de 3.3V entre el dispositivo *Waspmote* y el módulo Wi-Fi. A continuación se muestra una imagen del test realizado.

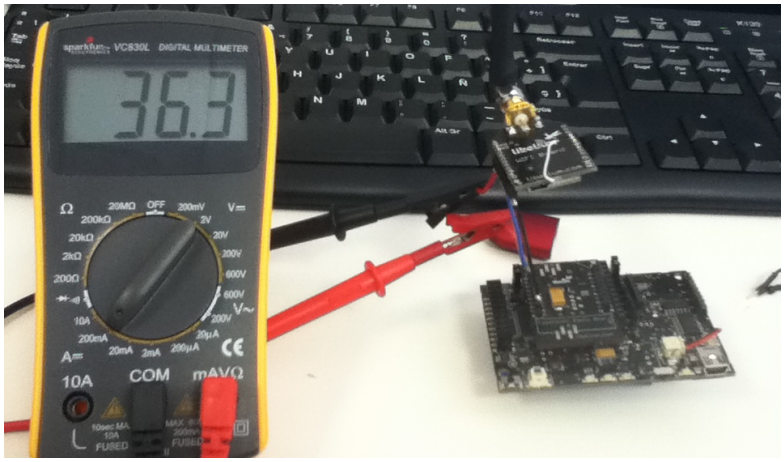

Figura 5.7: Prueba de consumo del módulo Wi-Fi definitivo

5.5. Alcance

La realización de esta prueba es necesaria para determinar la longitud de los enlaces que se pueden alcanzar utilizando los módulos Wi-Fi en un entorno real.

El alcance del dispositivo Wi-Fi depende principalmente de la antena conectada al conector SMA. Se acota la comparativa a los dos tipos de antenas que se utilizan con los módulos de *Waspmote*, 2 dBi y 5 dBi, y poniendo la transmisión al máximo de potencia (12 dBm).

Para la realización de esta prueba se buscó una recta sin obstáculos. Se ha elegido la calle del Poeta Luciano Gracia, una recta cerca de la Escuela de Ingeniería y Arquitectura de la Universidad de Zaragoza.

Los rangos cambian en comunicaciones entre *Waspmote* y conectando a un punto de acceso. También son diferentes para una transmisión perfecta (100% paquetes UDP recibidos) y para visibilidad de un AP (distancia mayor).

La siguiente tabla muestra un resumen de dichas mediciones dependiendo de la antena.

Waspmote + antena 2 dBi	20 m – 100 m
Waspmote + antena 5 dBi	300 m – 500 m

Tabla 5.6: Tabla de distancias

5.6. Rendimiento

Finalmente, se han realizado pruebas "de carga", estas pruebas se utilizan para comprobar el correcto funcionamiento a largo plazo del módulo Wi-Fi. En este caso la prueba es sencilla:

Despertar -> Conectar AP Meshlium -> Mandar trama UDP -> Dormir

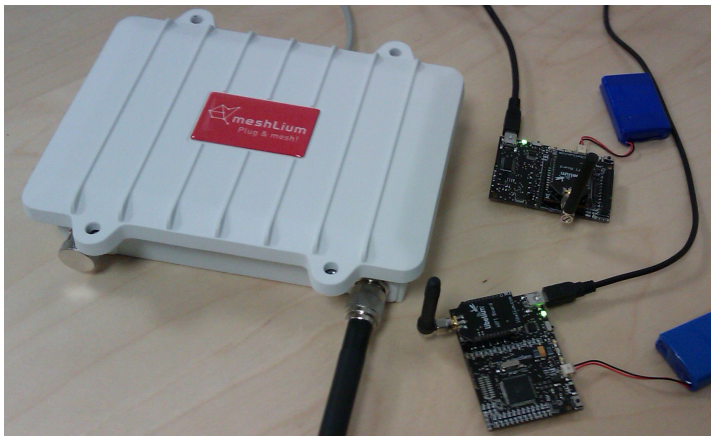

Figura 5.8: Prueba de rendimiento con Waspmote v1.2

Así durante 10 días, durmiendo cada 5 minutos. Se realiza la prueba con la versión *Waspmote* actual (v1.1) y la que saldrá a lo largo de este año (v1.2).

Waspmote v1.1	99,96 % recibidos
Waspmote v1.2	99,93 % recibidos

Tabla 5.7: Tabla de rendimiento

Capítulo 6

Software Asociado

Para conectar el módulo Wi-Fi a otro dispositivo que no sea el propio módulo Wi-Fi se ha desarrollado una serie de software asociado. Dicho software se compone de un programa multiplataforma para comunicar el módulo Wi-Fi con un PC en modo cliente, y aplicaciones para *iOS* [29] y *Android* [30] para comunicar con *iPhone*, *iPad* y móviles *Android* en modo Ad-hoc.

En esta sección se explica lo más importante del software asociado, se dispone de más información en el Anexo E.

6.1. Software PC

En esta sección se describe el programa multiplataforma para PC que se comunica con el módulo Wi-Fi del *Waspmote* mediante UDP, TCP o traduciendo el contenido del Broadcast-UDP.

Dicho programa se puede descargar de la web de *Libelium*:
http://www.libelium.com/development/waspmote

La pantalla del programa está dividida en 3 partes principales:

- La primera parte corresponde a los datos de configuración y a los botones de crear conexión y cerrar conexión.
- En la segunda parte hay dos cuadros de texto, uno que muestra información y los mensajes recibidos, y en la segunda se escribe el mensaje a enviar junto con el botón de enviar.
- Finalmente, en la parte inferior de la pantalla hay un botón para salir del programa.

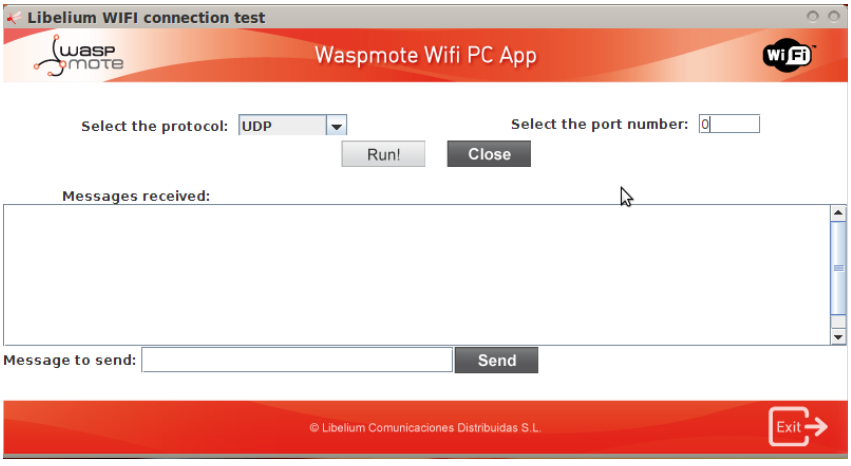

Figura 6.1: Software PC en funcionamiento

El funcionamiento del software es simple, basta con seleccionar el puerto y el tipo de protocolo en el programa e indicar la IP del ordenador en el código que se introduce al *Waspmote*. En cuanto recibamos un primer mensaje del *Waspmote* podremos responderle un mensaje, ya que sabremos su dirección.

En cuanto al **código del software PC**, es un programa escrito en Java bajo una interfaz gráfica utilizando librerías Java-Swing y un socket corriendo en un *Threat* utilizando librerías Java-Net.

6.2. Aplicación *iPhone*

Se han desarrollado aplicaciones para *iOS,* una para recibir datos de módulos Wi-Fi (en concreto posición GPS) y dibujar dichos módulos en un mapa. Otra aplicación permite la comunicación del iPhone al dispositivo *Waspmote*, mandándole mensajes para controlar su funcionamiento.

Finalmente se realiza la aplicación "oficial" ***Waspmote Wifi***, que es la que se va a introducir aquí. Dicha aplicación está disponible en el ***App Store*** [31] de Apple.

Para utilizar la aplicación basta con descargarla a un dispositivo iPod, iPhone o iPad. Conectar el dispositivo a la red que crea módulo Wi-Fi, y lanzar la aplicación.

Figura 6.2: Conexión a la red del módulo Wi-Fi e icono de la iPhone App

Dentro de la aplicación, hay 3 pestañas en la parte inferior de la pantalla que corresponden a cada una de las pantallas de la aplicación.

- La primera pantalla (*Network* – Red), proporciona información acerca de la red creada por el módulo Wi-Fi. Dicha información se compone de Nombre de la red, dirección MAC del módulo Wi-Fi, y la potencia/calidad de dicha red.

- La segunda pantalla (*Monitoring* – Monitorización), muestra la información recibida en tiempo real de sensores acoplados al módulo Wi-Fi tales como temperatura, humedad, luminosidad… y los datos del acelerómetro, con los cuales se dibuja el dispositivo *Waspmote* con la inclinación que tenga en la pantalla.

- Y finalmente, la tercera pantalla (*Actuator* – Actuador), hay tres interruptores con los que se envía distintos mensajes Encendido/Apagado a *Waspmote*, y una barra de desplazamiento que envía valores de 0 a 100%.

Figura 6.3: Las 3 pantallas de la App iPhone

En el vídeo promocional del producto se ve esta aplicación en funcionamiento con un set de luces conectado a *Waspmote*, con el que se controla la intensidad de las mismas. Dicho vídeo se encuentra en: http://www.youtube.com/watch?v=Ybn-K8ZZCPE

6.3. Aplicación *Android*

Para *Android* se han desarrollado las mismas aplicaciones que para *iPhone*, pero adaptadas a los dispositivos *Android*. En este caso, también se ha desarrollado la aplicación "oficial" ***Waspmote Wifi***, la cual se va a introducir en esta sección. Dicha aplicación se puede descargar desde ***Android Market*** [32].

Para utilizar esta aplicación, basta con descargarla del *Android Market* y lanzar una red Ad-hoc desde el dispositivo *Android* a la que *Waspmote* se conecta.

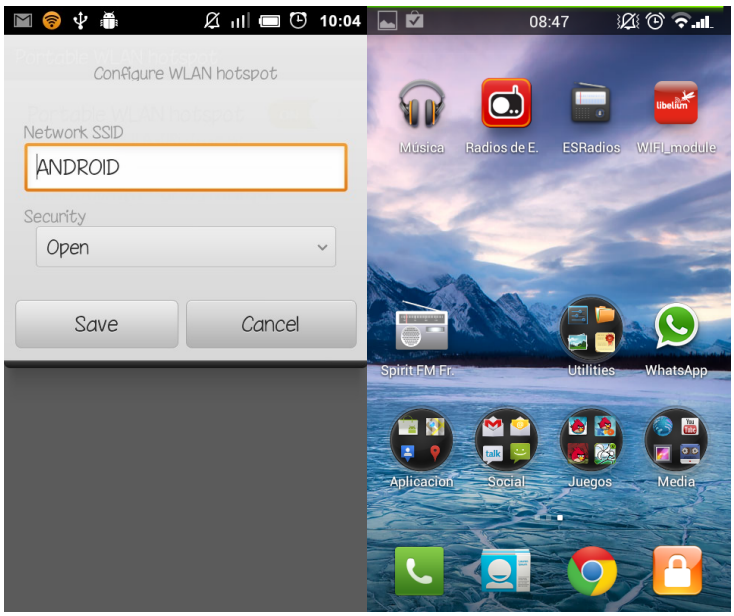

Figura 6.4: Creación de red Wi-Fi e icono de la Android App

Dentro de la aplicación, hay 3 pestañas en la parte superior de la pantalla que corresponden a cada una de las pantallas de la aplicación.

- La primera pantalla (*Network* – Red), proporciona información acerca de la red creada por el módulo Wi-Fi. Dicha información se compone de Nombre de la red, dirección MAC del módulo Wi-Fi, y la potencia/calidad de dicha red.

- La segunda pantalla (*Monitoring* – Monitorización), muestra la información recibida en tiempo real de sensores acoplados al módulo Wi-Fi tales como temperatura, humedad, luminosidad… y los datos del acelerómetro, con los cuales se dibuja el dispositivo *Waspmote* con la inclinación que tenga en la pantalla.

- Y finalmente, la tercera pantalla (*Actuator* – Actuador), hay tres interruptores con los que se manda distintos mensajes Encendido/Apagado a Waspmote, y una barra de desplazamiento que envía valores de 0 a 100%.

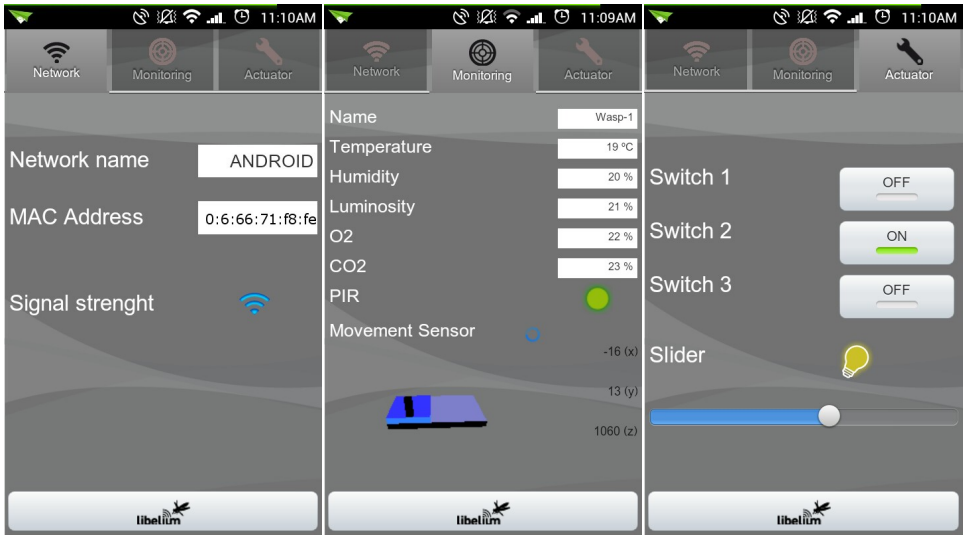

Figura 6.5: Las 3 pantallas de la App Android

Capítulo 7

Conclusiones y Futuro

7.1. Aportaciones del PFC

En este aparatado se realiza un balance del proyecto.

A nivel profesional, se ha trabajado sobre un sistema real que finalmente está operativo, y se está vendiendo con éxito.

Con la selección del integrado Wi-Fi y al tener que programar una API de bajo nivel, he madurado o aprendido en varios aspectos prácticos de la electrónica, como puede ser interpretar el documento esquemático de un integrado, medir el consumo de una placa o algo tan básico como soldar pines a un integrado.

A nivel de software, el desarrollar una API para un dispositivo empotrado con una memoria limitada y con la necesidad de que tenga un consumo limitado, obliga a programar de un modo más optimizado. A nivel de programación he ampliado mis conocimientos sobre C++, Objetive C (*iOS*), Java (*Android*).

A nivel personal, el hecho de desarrollar el proyecto en una empresa como *Libelium*, me ha permitido tener una visión global del funcionamiento de una empresa desde dentro. También su organización y la convivencia con los compañeros, con los que se colabora y en ocasiones se trabaja en paralelo.

Finalmente, he reforzado algunos aspectos que se abordan en la carrera sobre organización de proyectos y realización de diagramas para presentación de resultados. Y como el módulo desarrollado es un módulo Wi-Fi, he reforzado y he aprendido bastantes aspectos sobre funcionamiento interno de protocolos de transmisión tanto en Internet, como punto a punto.

7.2. Cumplimiento de objetivos

El **Objetivo principal** del proyecto, evaluar y desarrollar el API de control de un radio Wi-Fi para su uso en dispositivos empotrados como los *Waspmote*, se ha alcanzado en su totalidad.

Se ha desarrollado la biblioteca para comunicar un dispositivo *Waspmote* con otro dispositivo mediante un módulo Wi-Fi de una manera sencilla y eficiente. Se han desarrollado funciones tanto para comunicación en modo cliente (infraestructura) como en modo Ad-hoc, y en cuanto a comunicación en modo cliente, se han desarrollado funciones utilizando protocolos *TCP, UDP, UDP-Broadcast, HTTP, FTP, DNS*.

Dicha biblioteca tiene en cuenta las limitaciones en cuanto a memoria y consumo del dispositivo empotrado y ofrece herramientas para optimizar el rendimiento y ahorrar energía. Dicha biblioteca es código abierto que podrán modificar los usuarios finales que adquieran el módulo Wi-Fi para su red de nodos sensoriales.

Se ha logrado desarrollar un abanico de software para comprobar el correcto funcionamiento de la biblioteca.

Para el modo cliente (infraestructura) se ha desarrollado software para PC o en el servidor de *Libelium*. Y para las redes Wi-Fi Ad-hoc se ha desarrollado una serie de aplicaciones para *iPhone* y *Android* en las cuales los móviles se comunican con el módulo Wi-Fi de distintas maneras. Finalmente también se ha desarrollado programas para comunicar los módulos Wi-Fi entre ellos vía Wi-Fi Ad-hoc.

El desarrollo del proyecto se realizó de Septiembre de 2011 a Marzo de 2012, con una inversión de 664 horas de trabajo en total. Lo anterior se explica con detalle en el Anexo F.

7.3. Trabajo Futuro

Las líneas futuras de trabajo sobre este proyecto se pueden dividir en:

- Desarrollo con la biblioteca del módulo Wi-Fi de un protocolo que permita comunicar periódicamente datos del *Waspmote* o de sus módulos a una plataforma Cloud en Internet. Esta tarea está siendo desarrollada en la actualidad en la empresa.

- Desarrollo de aplicaciones más complejas para dispositivos móviles para el control y programación de las redes sensoriales como puede ser la programación sobre el aire (*OTA-programming*), lo cual permitiría programar o actualizar el firmware de la red sensorial desde un dispositivo inalámbrico (*iPhone, Android…*) como si fuera una puerta de enlace.

Capítulo 8

Referencias

[1] **WSN - Wireless Sensor Networks**
Network-Growing Scenarios in IEEE 802.15.4 Wireless Sensor Networks. Slobodanka Tomic. Vienna Austria.

[2] **Estándar ZigBee**
http://www.zigbee.org/Standards/Overview.aspx

[3] **Estándar Wi-Fi**
http://standards.ieee.org/getieee802/802.11.html

[4] **Conexión Ad-hoc**
Ad Hoc Mobile Wireless Networks: Protocols and Systems. Chai K Toh. Prentice Hall. 2002.
Wi-Fi Direct: http://es.wikipedia.org/wiki/Wi-Fi_Direct

[5] **Conexión Infraestructurada**
http://en.wikipedia.org/wiki/Wireless_LAN

[6] **Dispositivo Waspmote**
http://www.libelium.com/products/waspmote

[7] **Waspmote IDE**
http://www.libelium.com/development/waspmote/quickstart

[8] **Estándar Bluetooth**
http://standards.ieee.org/about/get/802/802.15.html

[9] **Estándar WiMAX**
http://standards.ieee.org/getieee802/802.16.html

[10] **Meshlium**
http://www.libelium.com/products/meshlium

[11] **ATMEGA 1281**
8-bit Atmel Microcontroller with 64K/128K/256K Bytes In-System Programmable Flash. Atmel. Mayo 2011.
http://www.atmel.com/Images/doc2549.pdf

[12] **WiFly RN-171 de Roving Networks**
User Manual and Command Reference, 802.11 b/g wireless LAN Modules, WiFly GSX. Roving Networks, Inc. Los Gatos, California. Abril 2011.

[13] RS9110N11 de RedPine Signals
RS9110-N-11-22/24/26/28 – Self-Contained 802.11n Modules with Networking Stack, Software Programming Reference Manual. Redpine Signals, Inc. San Jose, California. Septiembre 2010.

[14] Expansion Radio Board
http://www.libelium.com/expansion_radio_board

[15] Conector SMA
http://en.wikipedia.org/wiki/SMA_connector

[16] Dispositivo e IDE Arduino
http://www.arduino.cc

[17] Wiring
http://wiring.org.co/

[18] TCP
Redes de Computadoras. Andrew S. Tanenbaum. Prentice Hall. Mexico. 2003. http://books.google.es/books?id=WWD-4oF9hjEC

[19] UDP
Redes de Computadoras. Andrew S. Tanenbaum. Prentice Hall. Mexico. 2003. http://books.google.es/books?id=WWD-4oF9hjEC

[20] Broadcast-UDP
http://en.wikipedia.org/wiki/Broadcast

[21] HTML
Redes de Computadoras. Andrew S. Tanenbaum. Prentice Hall. Mexico. 2003. http://books.google.es/books?id=WWD-4oF9hjEC

[22] FTP
Redes de Computadoras. Andrew S. Tanenbaum. Prentice Hall. Mexico. 2003. http://books.google.es/books?id=WWD-4oF9hjEC

[23] IP
IP Technology Basics. D.I. Manfred Lindner. Institute of Computer Technology – Vienna University of Technology. 2006

[24] DHCP
Redes de Computadoras. Andrew S. Tanenbaum. Prentice Hall. Mexico. 2003. http://books.google.es/books?id=WWD-4oF9hjEC

[25] WEP
http://es.wikipedia.org/wiki/Wired_Equivalent_Privacy

[26] WPA
http://en.wikipedia.org/wiki/Wi-Fi_Protected_Access

[27] WPA2
http://en.wikipedia.org/wiki/Wi-Fi_Protected_Access

[28] **Baud-Rate (Jean-Maurice-Émile Baudot)**
http://www.compkarori.com/dbase/bu07sh.htm

[29] **iOS 5**
http://www.apple.com/es/ios/

[30] **Android**
http://www.android.com/

[31] **App Store**
http://itunes.apple.com/us/app/waspmote-wifi/id501974230

[32] **Android Market**
https://play.google.com/store/apps/details?id=com.libelium.WIFI_module

II. ANEXOS

Anexo A

Dispositivo Waspmote

Waspmote se basa en una arquitectura modular. La idea es integrar únicamente los módulos que necesitemos en cada dispositivo y ser capaces de cambiarlos y ampliarlos según las necesidades.

Los módulos disponibles para integrar en Waspmote se clasifican en:

- Módulos ZigBee/802.15.4 (2.4GHz, 868MHz, 900MHz). Baja y alta potencia.
- Módulo GSM - 3G/GPRS (Quadband: 850MHz/900MHz/1800MHz/1900MHz)
- Módulo GPS
- Módulos Sensoriales (Placas de Sensores)
- Módulo de almacenamiento: SD Memory Card

En la siguiente lista podemos ver las especificaciones del dispositivo:

- Microcontrolador: ATmega1281
- Frecuencia: 8MHz
- SRAM: 8KB
- EEPROM: 4KB
- FLASH: 128KB
- SD Card: 2GB
- Peso: 20gr
- Dimensiones: 73.5 x 51 x 13 mm
- Rango de Temperatura: [-20ºC, +65ºC]

La siguiente figura muestra los componentes principales en *Waspmote:*

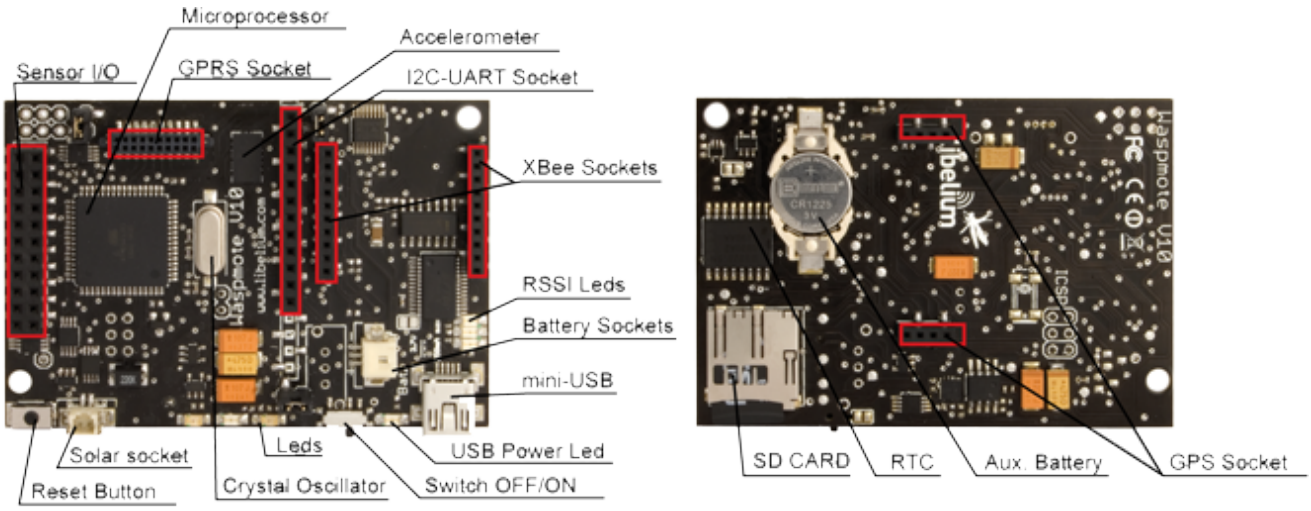

Figura A.1: Componentes principales en Waspmote. Caras superior e inferior.

En las figuras A.2 y A.3 podemos ver diagramas de bloques del dispositivo Waspmote. El diseño del dispositivo Waspmote fue realizado por el equipo de Libelium con anterioridad a la realización de este PFC.

Señales de Datos:

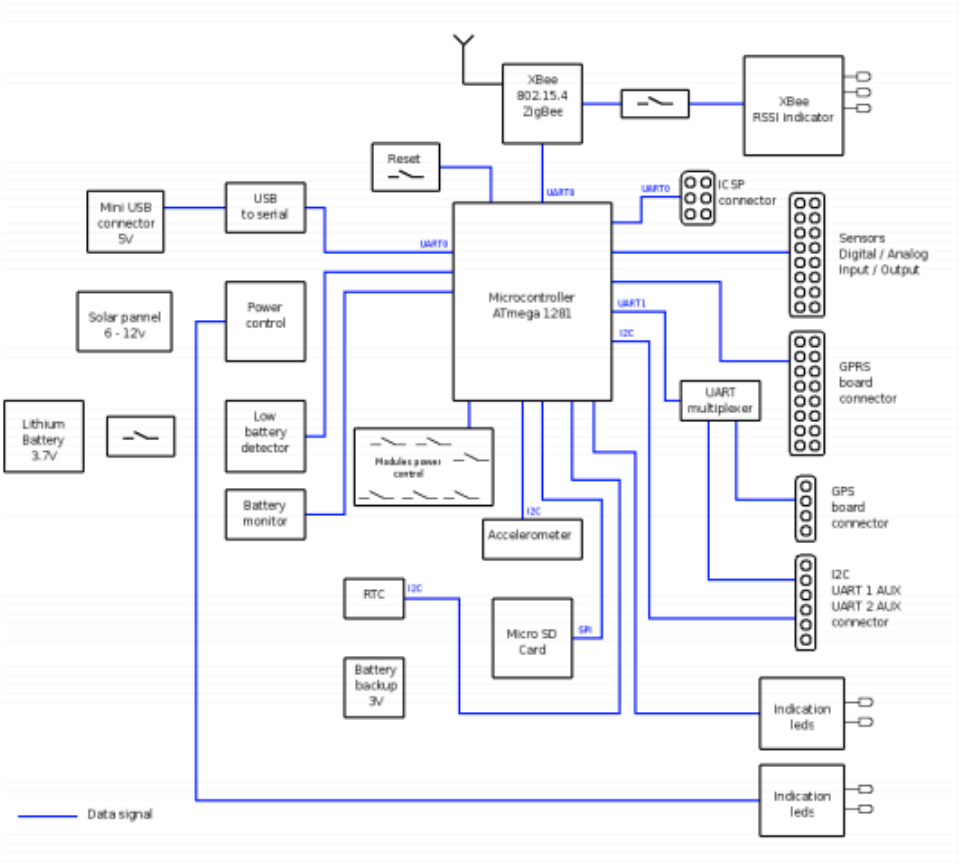

Figura A.2: Señales de datos.

Señales de Alimentación:

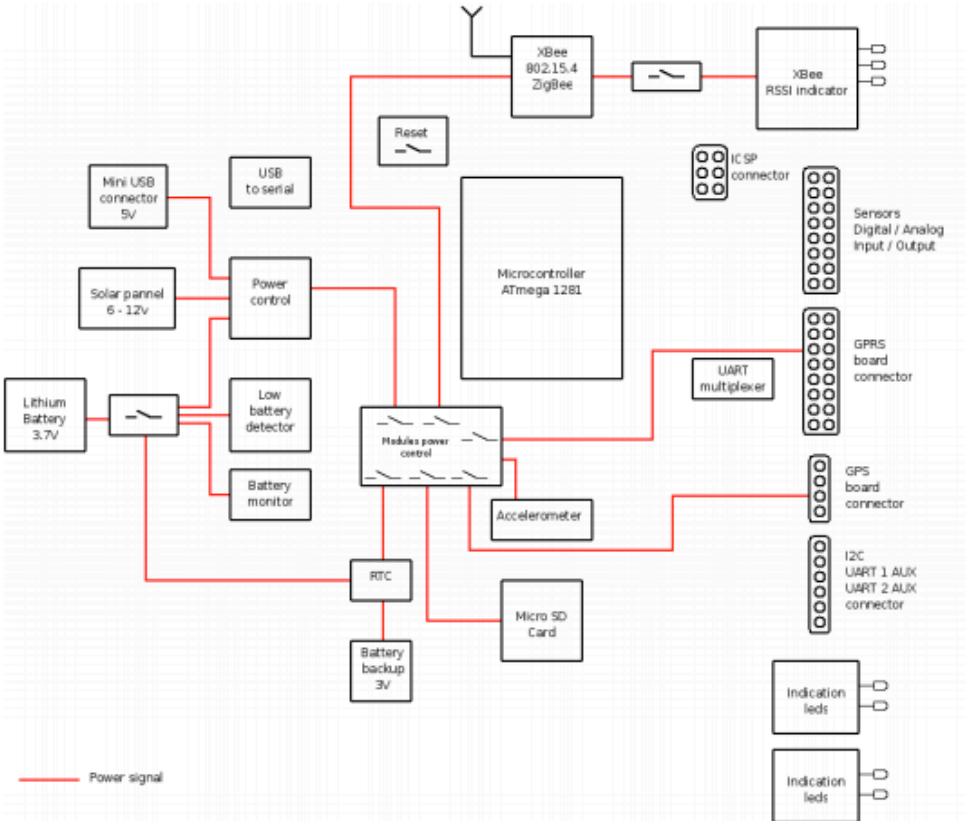

Figura A.3: Señales de alimentación.

El dispositivo está basado en un microcontrolador ATMEGA 1281 que funciona a 8MHz. La ventaja de los microcontroladores ATMEGA es que se pueden utilizar en un entorno amigable para el usuario. Para ello, es necesario cargar un '*bootloader*' antes de empezar a programarlo, que se encargará de cargar el programa a nuestro microcontrolador.

El *bootloader* es el programa que se encarga de preparar al microcontrolador para que acepte el código de cada programa que queremos cargar. De esta forma, la utilización de un dispositivo externo como un programador sólo es necesaria para la carga de dicho *bootloader*. Una vez cargado el *bootloader* se podrá hacer uso de un *IDE (Integrated Development Environment)* para cargar los programas al dispositivo.

En cuanto a consumo, Waspmote tiene 4 modos de funcionamiento:

- ON: modo normal de funcionamiento. El consumo en este estado es de 9mA.

- Sleep: El programa principal se detiene, el microcontrolador pasa a un estado de latencia, del que puede ser despertado por todas las interrupciones asíncronas y por la interrupción síncrona generada por el Watchdog. El intervalo de duración de este estado va de 32ms a 8s. El consumo en este estado es de 62µA.

• Deep Sleep: El programa principal se detiene, el microcontrolador pasa a un estado de latencia del que puede ser despertado por todas las interrupciones asíncronas y por la interrupción síncrona lanzada por el RTC. El intervalo de este ciclo puede ir de 8 segundos a minutos, horas, días. El consumo en este estado es de 62µA.

• Hibernate: El programa principal se detiene, el microcontrolador y todos los módulos de Waspmote quedan completamente desconectados. La única forma de volver a activar el dispositivo es a través de la alarma previamente programada en el RTC (interrupción síncrona).

La placa cuenta con un conector mini-USB para comunicar el microcontrolador con un ordenador, de forma que se pueda programar a través de un entorno amigable. Este conector también nos permite la captura presentación por pantalla de los datos generados por el dispositivo *Waspmote*.

La alimentación puede ser introducida a través del mini-USB, mediante una placa solar o bien a través de una batería de litio. La batería de litio se puede recargar al tener el módulo conectado al mini-USB o a la placa solar. De esta forma, podemos recargar el dispositivo *Waspmote* mientras lo programamos, o incrementar la duración de la vida de su atería al tenerlo conectado a una placa solar.

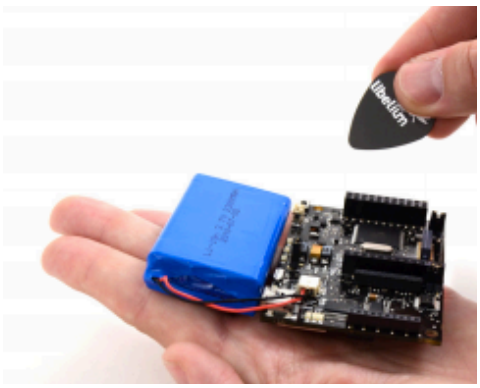

Figura A.4: Dispositivo Waspmote con batería de litio.

El dispositivo *Waspmote* dispone de 10 E/S digitales para poder conectar todo tipo de sensores externos, así como de 8 entradas analógicas. De esta forma obtenemos una gran cantidad de posibilidades de interconexión para nuestras redes sensoriales. En la siguiente figura se muestra como ejemplo el sensor de radiación.

Figura A.5: Sensor de radiación para Waspmote.

El dispositivo *Waspmote* también cuenta con un botón de reset, que permite reiniciar el código cargado en el microcontrolador. El código se reiniciará desde el punto de partida, siendo este código el último que se haya cargado en la placa.

La transmisión de datos se puede realizar de varias maneras. Una de ellas es a través de los módulos de comunicación implementados (el módulo Wi-Fi desarrollado en este PFC, o módulos ZigBee, Bluetooth,… ya implementados). La otra forma de transmitir es mediante 3G.

Figura A.6: módulos ZigBee, Bluetooth y Wi-Fi para Waspmote.

El almacenamiento de los datos capturados se puede realizar en una tarjeta mini-SD. Para ello, se ha diseñado un zócalo de conexión de dicho tipo de tarjetas SD. De esta forma, se pueden almacenar gran cantidad de datos, concretamente se puede trabajar con tarjetas de hasta 4 GBytes.

La sincronía y funciones de reloj se pueden realizar gracias al uso de un reloj interno. Dicho reloj sirve para poder tener la red sincronizada en caso de ser necesario, pudiendo despertar al dispositivo de un estado de bajo consumo.

Para posibles aplicaciones móviles o detectar caídas del dispositivo, se ha dotado al dispositivo *Waspmote* con un acelerómetro. De esta forma, se puede detectar si el dispositivo ha sufrido una caída o ha cambiado de dirección en un movimiento constante.

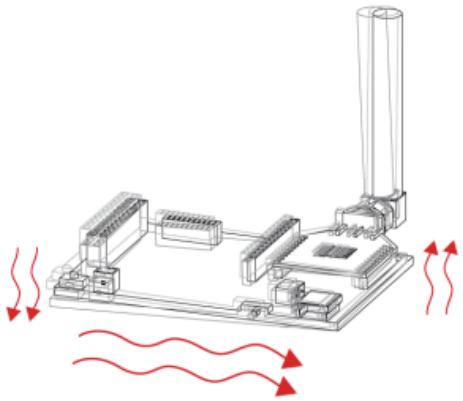

Figura A.7: Uso del acelerómetro frente a vibraciones.

Finalmente, con el fin de dotar a dichos dispositivos de un sistema de localización, se ha diseñado una placa de interconexión para conectar un receptor GPS. De esta forma, podemos obtener la localización en cada momento de los dispositivos *Waspmote*.

Anexo B

Libelium Radio Wi-Fi

En esta sección se explican los principales cambios en el módulo original de *Roving Networks* para realizar el Libelium Radio Wi-Fi final.

En las siguientes páginas se encuentran los documentos esquemáticos tanto del módulo *Roving Networks RN-171* original (Figura B.1), como del módulo Wi-Fi final (Figura B.2). También se encuentran las imágenes de la placa de cobre encargada de redirigir las entradas del integrado a los pines de la placa *Waspmote* (Figura B.3) y el esquema de pines del módulo final (Figura B.4).

Principales cambios:

- Se eliminan leds (GPIO_4, GPIO_5, GPIO_6).
- Las tierras (GND) se unen al mismo conector y se redirigen a un **pin GND**.
- Se conecta un **conector SMA** a la salida de la antena (ANTENNA).
- Las entradas VBATT, VDD_3V3_RF, VDD_3V3 se redirigen a un **pin a +3V3**.
- Las entradas GPIO_11 (UART_RX) y GPIO_10 (UART_TX) se redirigen a dos **pines RX y TX**.
- Finalmente, se sacan a ***agujeros*** las entradas ISP_RX, ISP_TX y RESET_N para tener una entrada/salida secundaria y una entrada de *reset*, esto se hace para futuras actualizaciones/necesidades.

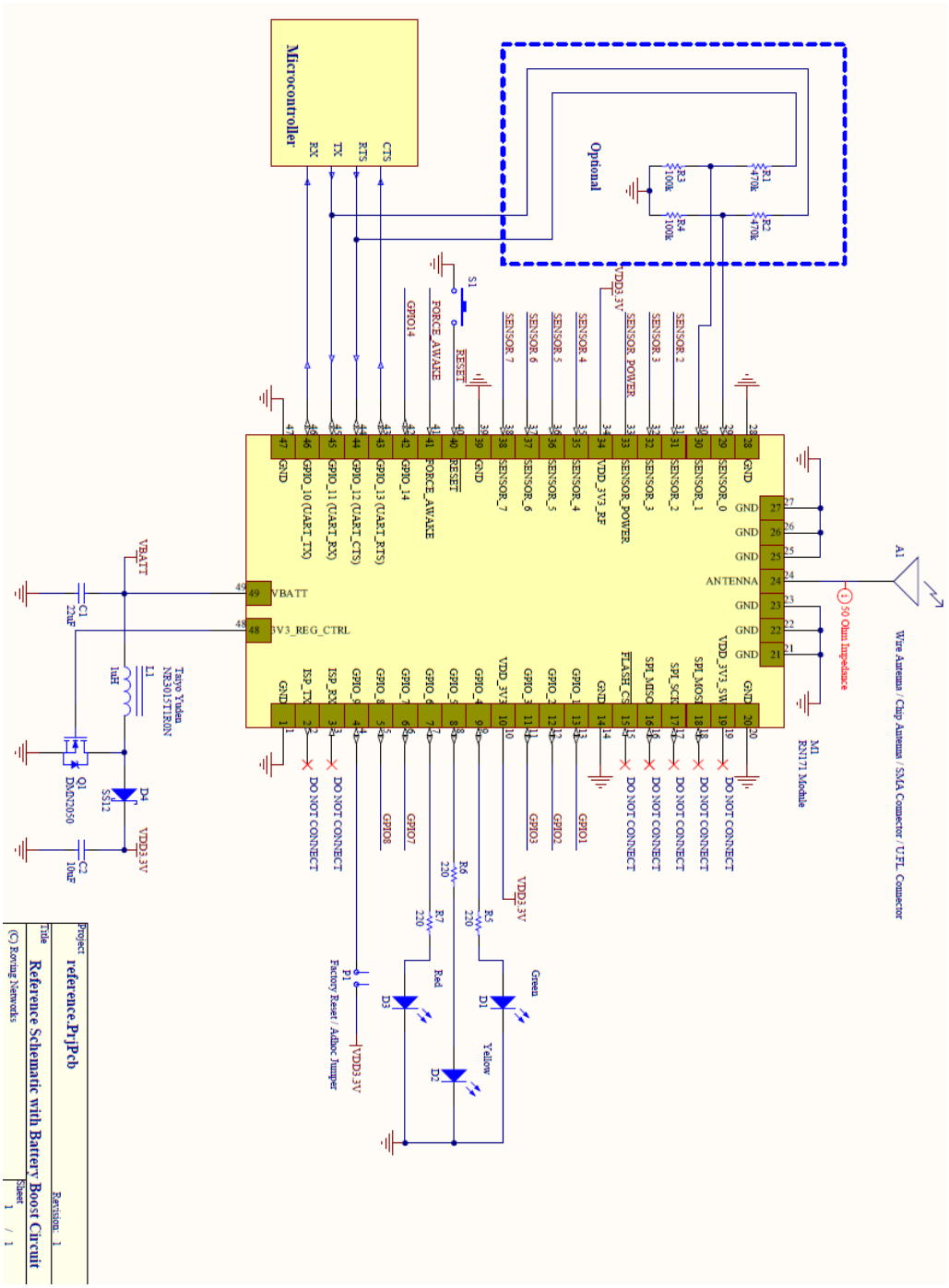

Figura B.1 : Documento esquemático Roving Networks RN-171.

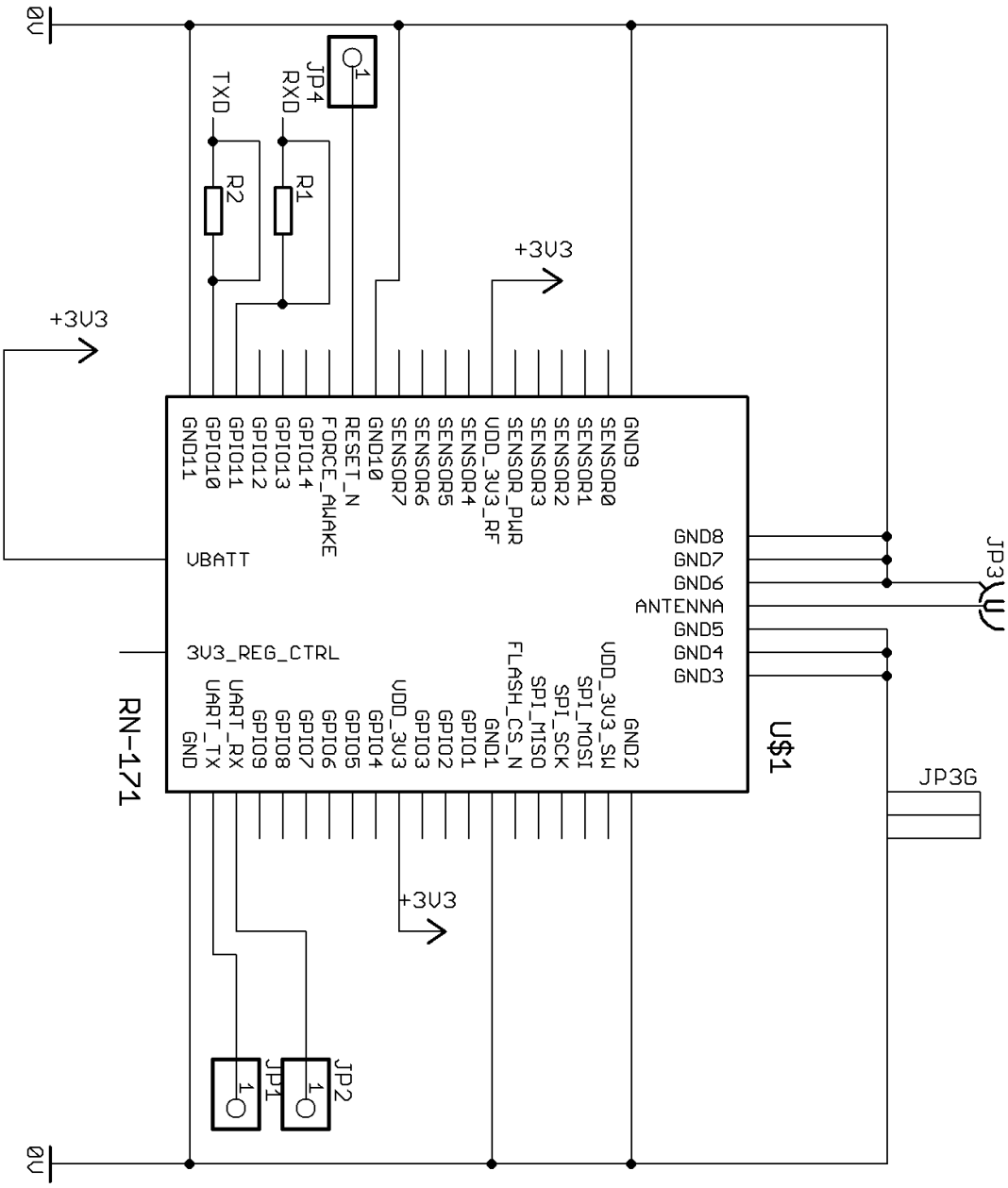

Figura B.2 : Documento esquemático final módulo Wi-Fi.

A continuación se presentan las placas de cobre encargadas de redirigir las entradas del integrado a los pines *Waspmote*:

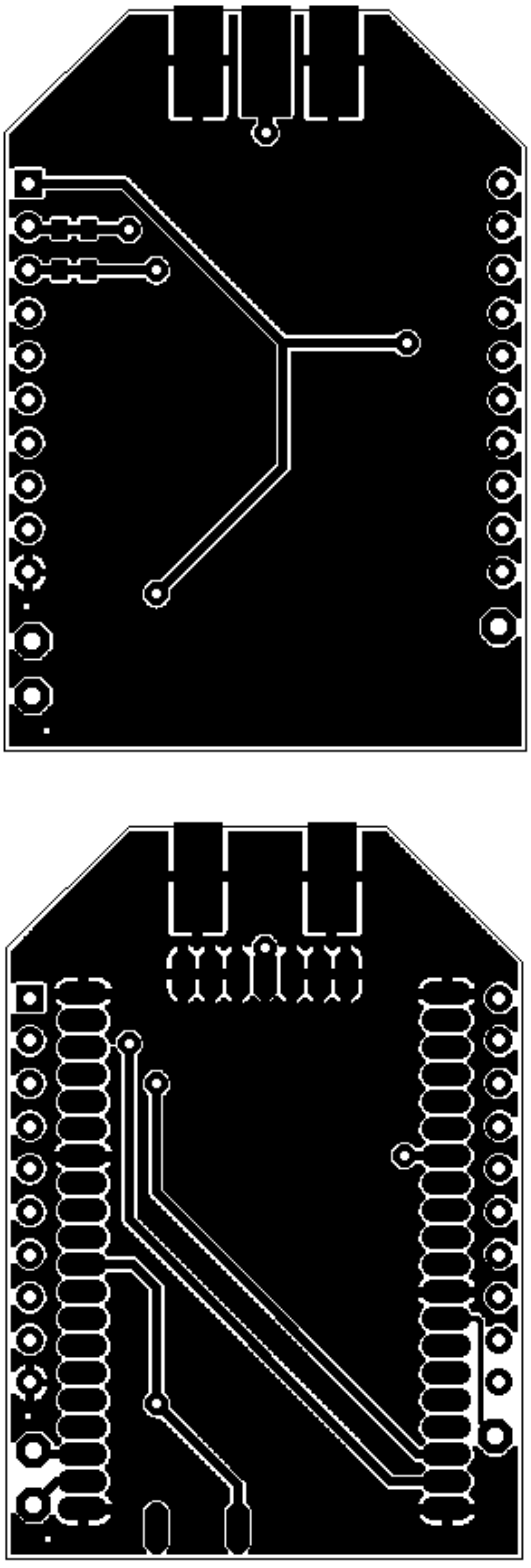

Figura B.3: Placa de cobre (superior e inferior)

Finalmente, se presentan los conectores finales del módulo Wi-Fi a *Waspmote*:

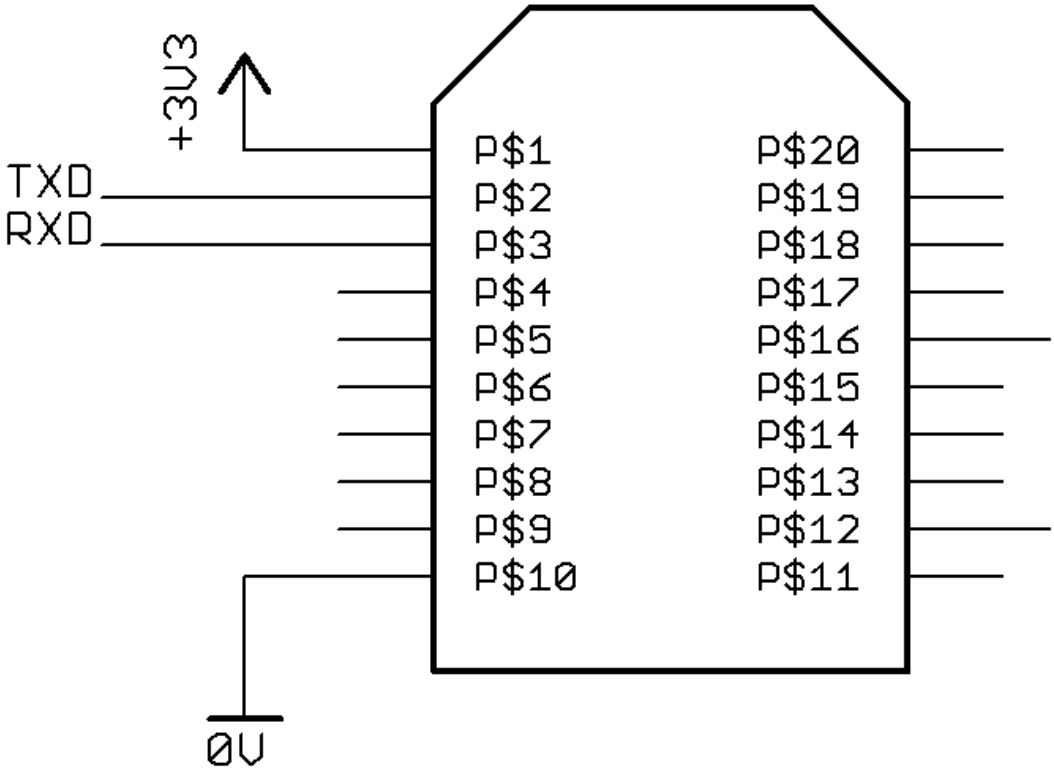

Figura B.4: Pines módulo Wi-Fi

Anexo C

API Waspmote Wi-Fi

En este anexo se describe todo lo relacionado con el código del API Waspmote Wi-Fi que se ha introducido en el capítulo 4 de la memoria (API módulo Wi-Fi).

Primero se presentan los requisitos del sistema, después se describe como se controla el integrado hardware desde el código desarrollado, y finalmente se presenta el código completo comentado.

C.1. Catálogo de Requisitos

Con la información recopilada en las reuniones, se elabora el siguiente Catálogo de Requisitos. Este Catálogo de Requisitos será la base sobre la que funcionará el módulo Wi-Fi.

A continuación se muestra el Catálogo de Requisitos que se compone de requisitos funcionales y no funcionales.

Requisitos Funcionales:

1. Requisitos tipo cualquier módulo *Waspmote*:

- Leer mensajes del integrado Wi-Fi: Ya sean mensajes de OK, FAIL o estado del dispositivo en modo comando, o mensajes recibidos a través de una conexión en modo transparente.

- Escribir mensajes al integrado Wi-Fi: Ya sean comandos en modo comando, o mensajes a enviar a través de una conexión.

- Cambiar integrado modo Comando a modo Transparente.

- Reiniciar integrado Wi-Fi: Necesario para que algunas configuraciones sean efectivas.

2. Requisitos propios Wi-Fi:

- Escanear puntos de acceso: Con diferentes opciones como escanear solo un canal, o con una encriptación determinada.

- Conectarse a un punto de acceso: Seleccionando su nombre (ESSID), posición en la lista de escaneado, o en modo automático el de mejor señal.

- Desconectarse de un punto de acceso.

- Lanzar un comando *ping* a una dirección IP: para saber si una dirección es válida o alcanzable.
- Resolver mediante DNS una dirección web.
- Sincronizar reloj interno con un servidor web.
- Configurar dirección IP estática, mediante protocolo DHCP, o mediante protocolo Auto-IP.
- Configurar puerta de enlace y máscara de red estáticamente.
- Crear una conexión *Ad-hoc*: Para conexiones con otros módulos Wi-Fi sin necesidad de un punto de acceso.
- Crear una conexión servidor *UDP*.
- Crear una conexión cliente *UDP*.
- Crear una conexión servidor *TCP*.
- Crear una conexión cliente *TCP*.
- Mandar datagramas *Broadcast UDP* cada cierto tiempo.
- Abrir una conexión *HTTP* con un comando predeterminado.
- Subir un fichero mediante *FTP*: Desde una tarjeta SD en *Waspmote* a un servidor en Internet.
- Descargar un fichero mediante *FTP*: Desde un servidor en Internet a una tarjeta SD conectada a *Waspmote*.
- Poner el módulo en modo bajo consumo.
- Despertar el módulo de modo bajo consumo.
- Mostrar información sobre el estado del módulo.

Requisitos No Funcionales:

- Encriptación: WEP, WPA, WPAMIX o WPA2.
- Rango canales: 1 a 13.
- Rango baud-rate: 2400 a 921600.
- Consumo módulo < 40 mA.
- Modo bajo consumo.
- Potencia transmisión configurable: 0 dBm a 12 dBm.

- Rango de distancia: Hasta 300 m.
- Tiempo de conexión menor de 10 s.
- Capacidad tarjeta micro-SD: 2 GB.
- Capacidad memoria RAM Waspmote: KBytes.

C.2. Diagrama de Control.

Como ya se ha introducido en el capítulo 4º, el integrado Wi-Fi dispone de dos modos, modo transparente y modo comando, uno para enviar información a través de una conexión configurada y otro para configurar dicha conexión.

El siguiente diagrama muestra el proceso de mandar un comando al integrado Wi-Fi una vez estamos en modo comando.

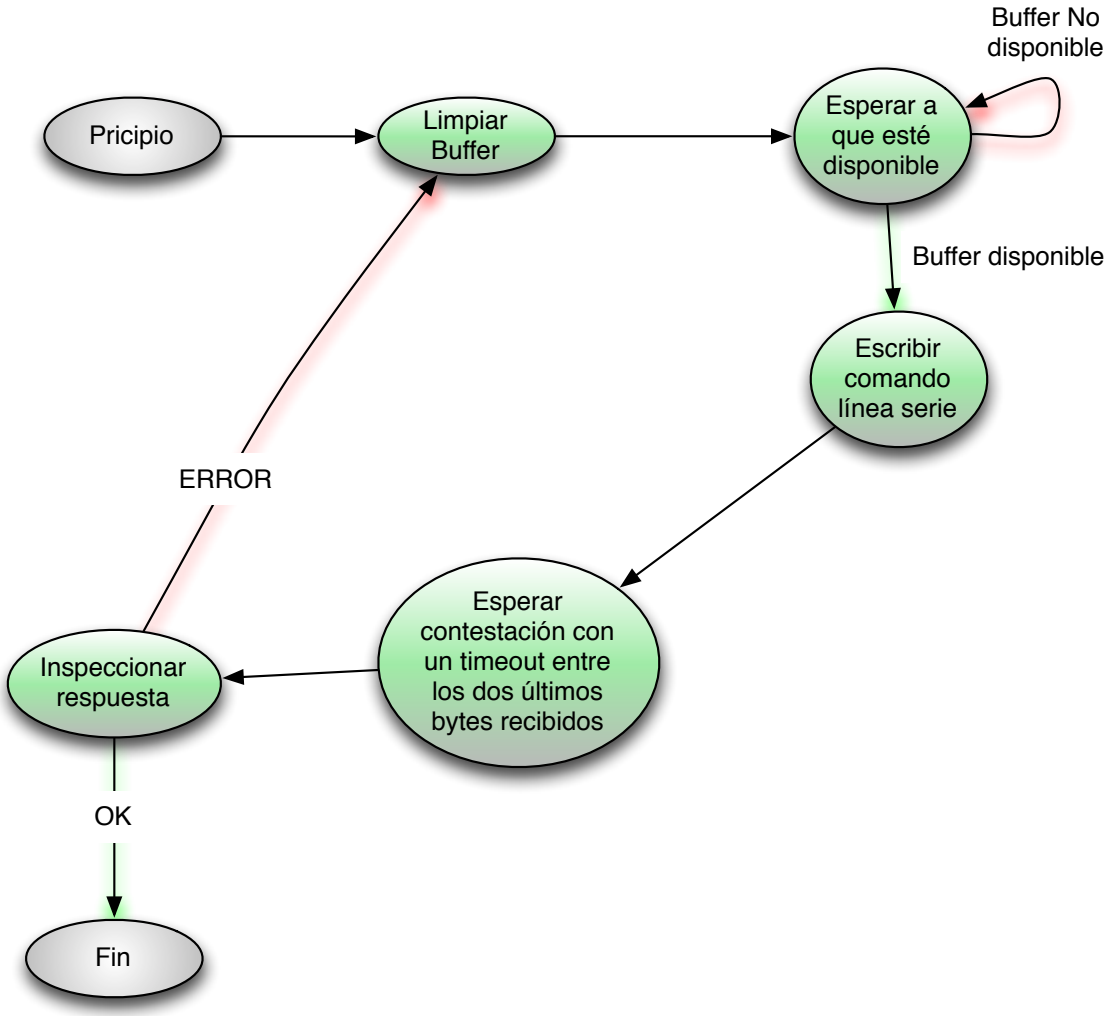

Figura C.1: Diagrama control enviar un comando al módulo Wi-Fi.

A continuación se describe el control del módulo Wi-Fi en el caso de descargar un archivo de un servidor FTP a la tarjeta SD.

Estos son los pasos a seguir con la API realizada:

- Inicializar el módulo: *WIFI.begin*
- Seleccionar protocolos a utilizar: *WIFI.setConnectionOptions*
- Configurar conexión FTP: *WIFI.setFTP, WIFI.openFTP*
- Conectar a un Punto de Acceso: *WIFI.join*
- Descargar fichero: *WIFI.getFile*

A continuación se muestra un diagrama con las funciones a bajo nivel de dicho proceso:

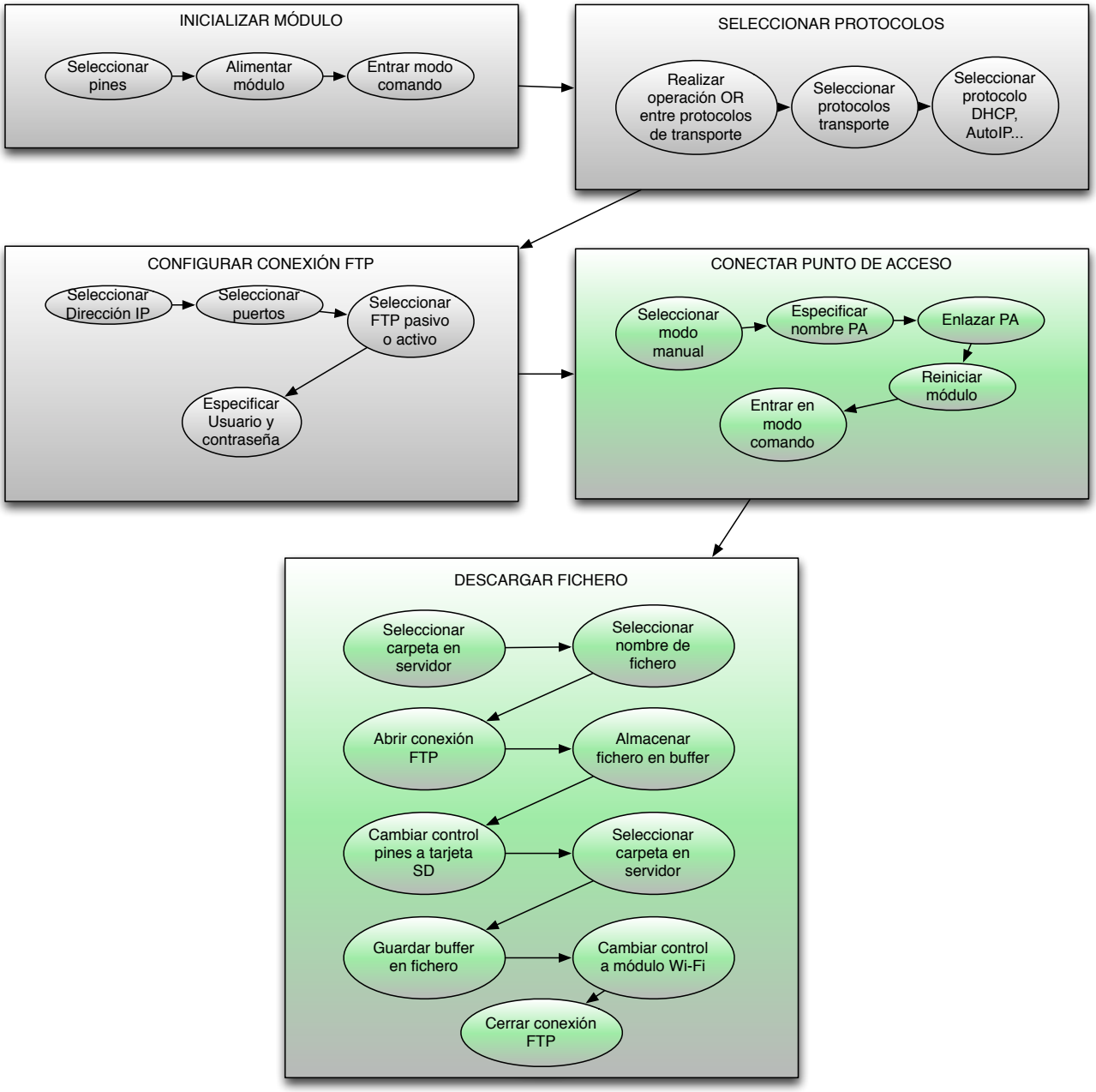

Figura C.2: Diagrama descargar fichero mediante protocolo FTP.

C.3. Código de ejemplo.

En este apartado se presentan programas de ejemplo que cubren casi todas las funcionalidades del módulo Wi-Fi de una forma general.

Ejemplos Básicos

```
/*
 *  ------Waspmote WIFI Status Commands Example--------
 *
 *  Explanation: This example shows how to get the status
 *  of the different WIFI module features.
 *
 *  Copyright (C) 2012 Libelium Comunicaciones Distribuidas S.L.
 *  http://www.libelium.com
 *
 *  Implementation:         Joaquin Ruiz
 */

void setup(){
  // Initialize the WIFI API and the connections with Waspmote
  WIFI.begin();
  // Then switch on the WIFI module on the desired socket.
  WIFI.ON(socket0);
  // If we don't know what configuration had the module, reset it.
  WIFI.resetValues();
}

void loop(){
  // Displays connection status.
  WIFI.getConnectionInfo();

  // Displays Access Point status.
  WIFI.getAPstatus();

  // Displays singal strenght information.
  WIFI.getRSSI();

  // Displays the statistics of the sent and received packets.
  WIFI.getStats();

  // Diplays the seconds since last powerup or reboot.
  WIFI.getUpTime();

  // Diplays adhoc settings.
  WIFI.getAdhocSettings();

  // Displays broadcast settings.
  WIFI.getBroadcastSettings();

  // Displays communications settings.
  WIFI.getComSettings();

  // Displays DNS settings.
  WIFI.getDNSsettings();

  // Displays FTP settings.
  WIFI.getFTPsettings();

  // Displays IP settings.
  WIFI.getIP();

  // Displays the MAC address.
  WIFI.getMAC();
```

```
  // Displays option settings.
  WIFI.getOptionSettings();

  // Displays system settings.
  WIFI.getSystemSettings();

  // Displays time server information.
  WIFI.getTime();

  // Displays Access Point settings.
  WIFI.getWLANsettings();

  // Displays UART settings.
  WIFI.getUARTsettings();

  // Displays the version.
  WIFI.getVersion();

  // Switches off the WIFI module.
  WIFI.OFF();
}
```

```
/*
 *  ------Waspmote WIFI wake sleep test--------
 *
 *  Explanation: This example shows how to configure the Wifi module
 *  to sleep and to wake up periodically.
 *
 *  Copyright (C) 2012 Libelium Comunicaciones Distribuidas S.L.
 *  http://www.libelium.com
 *
 *  Implementation:         Joaquin Ruiz
 */

void setup(){
  // First, initialize the WIFI API and the connections with the waspmote
  WIFI.begin();
  // Then switch on the WIFI module on the desired socket.
  WIFI.ON(socket0);
  // If we don't know what configuration had the module, reset it.
  WIFI.resetValues();

  // 1. Configure the transport protocol (UDP, TCP, FTP, HTTP...)
  WIFI.setConnectionOptions(HTTP);
  // 2. Configure the way the modules will resolve the IP address.
  WIFI.setDHCPoptions(DHCP_ON);
}

void loop(){
  // Configure something default to do when waking up.
  WIFI.setESSID("libelium_AP");
  WIFI.sendAutoBroadcast("192.168.1.150", 2500,0x7,"WAKED!");

  // Configures the Wifi module to sleep 90 seconds and then wake up for 30 seconds
  WIFI.setSleep(90,30);

  while(1){}
}
```

Conectar Punto de Acceso

```
/*
 *  ------Waspmote WIFI Connect AP modes Example--------
 *
 *  Explanation: This example shows the different ways to join to an
 *  AP (MANUAL, AUTO_STORE, AUTO_BEST, CREATE_ADHOC).
 *
 *  Copyright (C) 2012 Libelium Comunicaciones Distribuidas S.L.
 *  http://www.libelium.com
 *
 *  Implementation:         Joaquin Ruiz
 */

void setup(){
  // First, initialize the WIFI API and the connections with the waspmote
  WIFI.begin();
  // Then switch on the WIFI module on the desired socket.
  WIFI.ON(socket0);
  // If we don't know what configuration had the module, reset it.
  WIFI.resetValues();

  // 1. Configure the transport protocol (UDP, TCP, FTP, HTTP...)
  WIFI.setConnectionOptions(CLIENT_SERVER|CLIENT|UDP|HTTP);
  // 2. Configure the way the modules will resolve the IP address.
  WIFI.setDHCPoptions(DHCP_ON);
}

void loop(){
// *** MANUAL-MODE ***
  // 3. Configure how to connect the AP.
  WIFI.setJoinMode(MANUAL);
  // 3.1 If it is manual, call join giving the name of the AP.
  if (WIFI.join("libelium_AP"))
    // 4. Call the function that needs a connection.
    WIFI.resolve("www.libelium.com");

// *** AUTO-BEST ***
  // 3.0 Configure the Authentication mode of the auto-join.
  WIFI.setAutojoinAuth(OPEN);
  // 3. Configure how to connect the AP.
  if (WIFI.setJoinMode(AUTO_BEST))
    // 4. Call the function that needs a connection.
    WIFI.resolve("www.libelium.com");

// *** AUTO-STORE ***
  // 3.0 Configure the number of channel where we want to join
  WIFI.setChannel(6);
  // 3.1 Configure the authentication mode of the auto-join.
  WIFI.setAutojoinAuth(OPEN);
  // 3.2 Sets the name of the AP we want to join.
  WIFI.setESSID("libelium_AP");
  // 3. Configure how to connect the AP.
  if (WIFI.setJoinMode(AUTO_STOR))
    // 4. Call the function that needs a connection.
    WIFI.resolve("www.libelium.com");

// *** CREATE ADHOC ***
  // 2. Configure the way the modules will resolve the IP address.
  WIFI.setDHCPoptions(AUTO_IP);

  // 3.0 Configure the number of channel where we want to create the ADHOC network.
  WIFI.setChannel(6);
```

```
  // 3.1 Configure the authentication mode of the auto-join.
  WIFI.setAutojoinAuth(ADHOC);
  // 3.2 Sets the name of the ADHOC network we want to join.
  WIFI.setESSID("libelium_ADHOC");
  // 3. Configure how to connect the AP.
  if (WIFI.setJoinMode(CREATE_ADHOC))
    // 4. Whatever we want to do
    while(1){}
}
```

```
/*
 *  ------Waspmote WIFI Connect encrypted AP Example--------
 *
 *  Explanation: This example shows how to join to an encrypted AP with
 *  different types of encryption (WEP, WPA1, WPAMIX, WPA2).
 *
 *  Copyright (C) 2012 Libelium Comunicaciones Distribuidas S.L.
 *  http://www.libelium.com
 *
 *  Implementation:         Joaquin Ruiz
 */

void setup(){
  // Initialize the WIFI API and the connections with Waspmote
  WIFI.begin();
  // Then switch on the WIFI module on the desired socket.
  WIFI.ON(socket0);
  // If we don't know what configuration had the module, reset it.
  WIFI.resetValues();

  // 1. Configure the transport protocol (UDP, TCP, FTP, HTTP...)
  WIFI.setConnectionOptions(CLIENT_SERVER|CLIENT|UDP|HTTP);
  // 2. Configure the way the modules will resolve the IP address.
  WIFI.setDHCPoptions(DHCP_ON);
}

void loop(){
// *** Wired Equivalent Privacy (WEP) ***
  // 3.1 Sets WEP encryptation // 13 Characters
  WIFI.setAuthKey(WEP, "1234567890123");
  // 3. Call join the AP
  if (WIFI.join("libelium_AP"))
  { // Resolve DNS address
    WIFI.resolve("www.libelium.com");
  }
  // Leaves AP
  WIFI.leave();

// *** Wifi Protected Access (WPA) ***
  // 3.1 Sets WPA1 encryptation // 1-64 Characters
  WIFI.setAuthKey(WPA1, "password");
  // 3. Call join the AP
  if (WIFI.join("libelium_AP"))
  { // Resolve DNS address
    WIFI.resolve("www.libelium.com");
  }
  // Leaves AP
  WIFI.leave();

// *** Wifi Protected Access Mix (WPA MIX) ***
  // 3.1 Sets Mixed WPA1 & WPA2-PSK encryptation // 1-64 Character
  WIFI.setAuthKey(WPAMIX, "password");
  // 3. Call join the AP
```

```
  if (WIFI.join("libelium_AP"))
  { // Resolve DNS address
    WIFI.resolve("www.libelium.com");
  }
  // Leaves AP
  WIFI.leave();

// *** Wifi Protected Access 2 (WPA 2) ***
  // 3.1 Sets WPA2-PSK encryptation // 1-64 Character
  WIFI.setAuthKey(WPA2, "password");
  // 3. Call join the AP
  if (WIFI.join("libelium_AP"))
  { // Resolve DNS address
    WIFI.resolve("www.libelium.com");
  }
  // Leaves AP
  WIFI.leave();
}
```

Modo Ad-hoc

```
/*
 *  ------Waspmote WIFI AdHoc communication Example--------
 *
 *  Explanation: This example shows the different ways to
 *  connect with another device in Adhoc mode.
 *
 *  Copyright (C) 2012 Libelium Comunicaciones Distribuidas S.L.
 *  http://www.libelium.com
 *
 *  Implementation:         Joaquin Ruiz
 */

 void setup(){
  // First, initialize the WIFI API and the connections with the waspmote
  WIFI.begin();
  // Then switch on the WIFI module on the desired socket.
  WIFI.ON(socket0);
  // If we don't know what configuration had the module, reset it.
  WIFI.resetValues();

  // 1. Configure the transport protocol (UDP, TCP, FTP, HTTP...)
  WIFI.setConnectionOptions(UDP);
  // 2. Configure the way the modules will resolve the IP address.
  WIFI.setDHCPoptions(AUTO_IP);
  // 3. Configure how to connect the AP.
  WIFI.setAutojoinAuth(ADHOC);
  // 3.1 Sets the name of the ADhoc network.
  WIFI.setESSID("libelium_ADHOC");
  // 3.2 Sets the channel of the ADhoc network
  WIFI.setChannel(6);
}

void loop(){
  // 3. call function to create/join the Adhoc network.
  if(WIFI.setJoinMode(CREATE_ADHOC)){
    while(1){
      // Send messages to another device in same adhoc.
      WIFI.send("Hi, I'm Waspmote nº x");
      // Receive messages from the other in same adhoc.
      WIFI.read(NOBLO);
    }
  }
}
```

Conexiones TCP, UDP

```
/*
 *  ------Waspmote WIFI UDP Server Example--------
 *
 *  Explanation: This example shows how to create an UDP Server
 *  connection.
 *
 *  Copyright (C) 2012 Libelium Comunicaciones Distribuidas S.L.
 *  http://www.libelium.com
 *
 *  Implementation:         Joaquin Ruiz
 */

void setup(){
  // First, initialize the WIFI API and the connections with the waspmote
  WIFI.begin();
  // Then switch on the WIFI modules.
  WIFI.ON(socket1);
  // If we don't know what configuration had the modules, reset it.
  WIFI.resetValues();

  // 1. Configure the transport protocol (UDP, TCP, FTP, HTTP...)
  WIFI.setConnectionOptions(UDP);
  // 2. Configure the way the modules will resolve the IP address.
  WIFI.setDHCPoptions(DHCP_ON);
}

void loop(){
  // 3. Configure how to connect the AP.
  WIFI.setJoinMode(MANUAL);
  // 3.1 If it is manual, call join giving the name of the AP.
  if (WIFI.join("libelium_AP"))
  {
    // 4. Call the function to create UDP connection in port 2500.
    if (WIFI.setUDPserver(2500))
    { // 2. Now we can use read functions to receive UDP messages.
  while(1){
    WIFI.read(NOBLO);
          // If the datagram received begins with '0' ends the connection
    if (WIFI.answer[0]=='0') break;
  }
    }
    // 3.Exits from UDP sending data mode.
    Wifi.close();
  }
```

```
/*
 *  ------Waspmote WIFI UDP Broadcast Example--------
 *
 *  Explanation: This example shows how to create periodically UDP Broadcast
 *  messages.
 *
 *  Copyright (C) 2012 Libelium Comunicaciones Distribuidas S.L.
 *  http://www.libelium.com
 *
 *  Implementation:         Joaquin Ruiz
 */

void setup(){
  // First, initialize the WIFI API and the connections with the waspmote
  WIFI.begin();
  // Then switch on the WIFI module on the desired socket.
  WIFI.ON(socket0);
  // If we don't know what configuration had the module, reset it.
  WIFI.resetValues();

  // 1. Configure the transport protocol (UDP, TCP, FTP,HTTP...)
  WIFI.setConnectionOptions(UDP);
  // 2. Configure the way the modules will resolve the IP address.
  WIFI.setDHCPoptions(DHCP_ON);
}

void loop(){
  // 3. Configure how to connect the AP
  WIFI.setJoinMode(MANUAL);
  // 3.1 If it is manual, call join giving the name of the AP
  if (WIFI.join("libelium_AP"))
  {
    // 4. Call the function to create the broadcast UDP messages to IP address
    // "192.168.1.150:55555" , each 7 seconds with the text "BROADCAST_TEST"
    if (WIFI.sendAutoBroadcast("192.168.1.150",55555,0x7,"BROADCAST_TEST"))
    {
      // 5. Now the waspmote will send udp broadcast messages each 7 seconds.
      while(1){}
    }
  }
}
```

```
/*
 *  ------Waspmote WIFI TCP Client Example--------
 *
 *  Explanation: This example shows how to create a TCP client
 *  connection.
 *
 *  Copyright (C) 2012 Libelium Comunicaciones Distribuidas S.L.
 *  http://www.libelium.com
 *
 *  Implementation:         Joaquin Ruiz
 */

void setup(){
  // First, initialize the WIFI API and the connections with the waspmote.
  WIFI.begin();
  // Then switch on the WIFI module.
  WIFI.ON(socket0);
  // If we don't know what configuration has the modules, reset it.
  WIFI.resetValues();

  // 1. Configure the transport protocol (UDP, TCP, FTP, HTTP...)
  WIFI.setConnectionOptions(CLIENT);
  // 2. Configure the way the modules will resolve the IP address.
  WIFI.setDHCPoptions(DHCP_ON);
}

void loop(){
  // 3. Configure how to connect the AP (encryptation and name of the AP).
  WIFI.setAutojoinAuth(OPEN);
  WIFI.setESSID("libelium_AP");
  // 3.1 Sets Store-AP join mode.
  if (WIFI.setJoinMode(AUTO_STOR))
  {
    Utils.setLED(LED1,LED_ON);
    // 4. Call the function to create a TCP connection to IP address
    // "192.168.1.216:3550" from local port 2000
    if (WIFI.setTCPclient("192.168.1.216", 3550,2000))
    {
      // 5. Now the connection is open, and we can use send and read functions
      // to control the connection.
      while(1){
        // Sends to the TCP connection
        WIFI.send("TCP - Hi from Waspmote through Wifi!");
        // Reads am answer from the TCP connection
        WIFI.read(BLO);
      }
      // Closes the TCP connection.
      WIFI.close();
    }
  }
```

Conexiones HTTP, FTP

```
/*
 *  ------Waspmote WIFI HTTP Example--------
 *
 *  Explanation: This example shows how to send a HTTP get request
 *  message.
 *
 *  Copyright (C) 2012 Libelium Comunicaciones Distribuidas S.L.
 *  http://www.libelium.com
 *
 *  Implementation:         Joaquin Ruiz
 */

void setup(){
  // First, initialize the WIFI API and the connections with the waspmote
  WIFI.begin();
  // Then switch on the WIFI module on the desired socket.
  WIFI.ON(socket0);
  // If we don't know what configuration had the module, reset it.
  WIFI.resetValues();

  // 1. Configure the transport protocol (UDP, TCP, FTP, HTTP...)
  WIFI.setConnectionOptions(HTTP);
  // 2. Configure the way the modules will resolve the IP address.
  WIFI.setDHCPoptions(DHCP_ON);
}

void loop(){
  // 3. Configure how to connect the AP
  WIFI.setJoinMode(MANUAL);
  // 3.1 If it is manual, call join giving the name of the AP
  if (WIFI.join("libelium_AP"))
  {
    // 4. Call the function that send the HTTP get/post request.
    // Specifying the DNS address
    WIFI.getURL(DNS,"www.libelium.com","GET$/radioWIFItest.php?name=Joaquin");
    // Specifying the IP address
    WIFI.getURL(IP,"84.20.10.73","GET$/radioWIFItest.php?name=Joaquin");
  }
}
```

```
/*
 *  ------Waspmote WIFI FTP Example--------
 *
 *  Explanation: This example shows how to upload and download files
 *  to and from a server using FTP protocol.
 *
 *  Copyright (C) 2012 Libelium Comunicaciones Distribuidas S.L.
 *  http://www.libelium.com
 *
 *  Implementation:         Joaquin Ruiz
 */

void setup(){
  // First, initialize the WIFI API and the connections with the waspmote
  WIFI.begin();
  // Then switch on the WIFI module on the desired socket.
  WIFI.ON(socket0);
  // If we don't know what configuration had the module, reset it.
  WIFI.resetValues();

  // 1. Configure the transport protocol (UDP, TCP, FTP, HTTP...)
  WIFI.setConnectionOptions(CLIENT_SERVER|UDP|HTTP);
  // 2. Configure the way the modules will resolve the IP address.
  WIFI.setDHCPoptions(DHCP_ON);

  // 3.0 Sets the server IP address, ports and FTP mode
  WIFI.setFTP("80.80.80.80",21,FTP_PASIVE,20);
  // 3.1 Sets the server account with the username and password
  WIFI.openFTP("user","password");
}

void loop(){
  // 4. Configure how to connect the AP
  WIFI.setJoinMode(MANUAL);
  // 4.1 If it is manual, call join giving the name of the AP
  if(WIFI.join("libelium_AP")){
    // 5. Upload a file to a FTP server.
    while(WIFI.uploadFile("File_in_SDCard.txt","SDCard_Folder","public_ftp/Folder")!=1){
}
    // 5. Get a file from the SD card.
    while(WIFI.getFile("file_in_FTPServer.txt","SDCard_Folder","public_ftp")!=1){}
  }
  // Exit and power off the module.
  WIFI.OFF();
}
```

Conectar con Aplicaciones realizadas.

```
/*
 *  ------Waspmote WIFI PC Example--------
 *
 *  Explanation: This example shows how to create a TCP
 *  connection to comunicate with the Waspmote Wifi PC program.
 *
 *  Copyright (C) 2012 Libelium Comunicaciones Distribuidas S.L.
 *  http://www.libelium.com
 *
 *  Implementation:         Joaquin Ruiz
 */

void setup(){
  // First, initialize the WIFI API and the connections with the waspmote
  WIFI.begin();
  // Then switch on the WIFI module on the desired socket.
  WIFI.ON(socket0);
  // If we don't know what configuration had the module, reset it.
  WIFI.resetValues();

  // 1. Configure the transport protocol (UDP, TCP, FTP, HTTP...)
  WIFI.setConnectionOptions(CLIENT|UDP);
  // 2. Configure the way the modules will resolve the IP address.
  WIFI.setDHCPoptions(DHCP_ON);
}

void loop(){
  // 3.0 Specify the Access Point.
  WIFI.setESSID("libelium_AP");
  // 3.1 Call join in stored mode.
  if (WIFI.setJoinMode(AUTO_STOR))
  {
    // 4. Call the function to create the TCP connection from port 2000
    // to program PC address "192.168.1.216:3550"
    if (WIFI.setUDPclient("192.168.1.216", 3550,2000))
    {
      // 5. Now the connection is open, and we can use send and read functions
      // to control the connection.
      while(1){
        // Sends a message to PC.
        WIFI.send("Hi from Waspmote through WIFI!\0");
        // Reads a message from the PC BLO means block until a message is received.
        WIFI.read(BLO);
      }
      // Closes the TCP connection.
      WIFI.close();
    }
  }
```

```
/*
 *  ------Waspmote WIFI IOS smartphone application Example--------
 *
 *  Explanation: This example shows the way to communicate with
 *  the Waspmote Wifi Demo iPhone app.
 *
 *  Copyright (C) 2012 Libelium Comunicaciones Distribuidas S.L.
 *  http://www.libelium.com
 *
 *  Implementation:         Joaquin Ruiz
 */

// Specifies the message that is sent to the WiFi module.
char tosend[128];

void setup(){
  // Initialize the accelerometer
  ACC.begin();
  ACC.setMode(ACC_ON);

  // Initialize the WIFI API and the connections with Waspmote
  WIFI.begin();
  // Then switch on the WIFI module on the desired socket.
  WIFI.ON(socket0);
  // If we don't know what configuration had the module, reset it.
  WIFI.resetValues();

  // 1. Configure the transport protocol (UDP, TCP, FTP, HTTP...)
  WIFI.setConnectionOptions(CLIENT_SERVER|UDP);
  // 2. Configure the way the modules will resolve the IP address.
  WIFI.setDHCPoptions(AUTO_IP);
  // 3. Configure how to connect the AP.
  WIFI.setAutojoinAuth(ADHOC);
  // 3.1 Sets the name of the ADhoc network.
  WIFI.setESSID("iPHONE_ADHOC");
  // 3.2 Sets the channel of the ADhoc network
  WIFI.setChannel(6);
}

void loop(){
  // 3.3 Call function to create/join the Adhoc network.
  if(WIFI.setJoinMode(CREATE_ADHOC)){
    // Switches on green led.
    Utils.setLED(LED0, LED_ON);
    // 4. Creates UDP connection.
    if (WIFI.setUDPclient("255.255.255.255",12345,2000)){
      // 5. Now we can use send and read functions to send and
      // receive UDP messages.
      while(1){
        // Sends data to the IOS smartphone
        sprintf(tosend,"Wasp-1;19;24;70;20;383;0;%d;%d;%d",
                ACC.getX(),ACC.getY(),ACC.getZ());
        WIFI.send(tosend);
        // Reads data to the IOS smartphone
        WIFI.read(NOBLO);
        // Waspmote acts depending on the answer.
        uint8_t period = (WIFI.answer[6] - 48) * 10 + (WIFI.answer[7] - 48);
        if (period<11){
          // Switches on a lamp with intensity= period % (...)
        }
        delay(100);
      }
    }
  }
}
```

```
/*
 *  ------Waspmote WIFI Android smartphone application Example--------
 *
 *  Explanation: This example shows the way to communicate with
 *  the Waspmote Wifi Demo Android app.
 *
 *  Copyright (C) 2012 Libelium Comunicaciones Distribuidas S.L.
 *  http://www.libelium.com
 *
 *  Implementation:         Joaquin Ruiz
 */

// Specifies the message that is sent to the WiFi module.
char tosend[128];

void setup(){
  // Initialize the accelerometer
  ACC.begin();
  ACC.setMode(ACC_ON);
  // First, initialize the WIFI API and the connections with the waspmote
  WIFI.begin();
  // Then switch on the WIFI module on the desired socket.
  WIFI.ON(socket0);
  // If we don't know what configuration had the module, reset it.
  WIFI.resetValues();

  // 1. Configure the transport protocol (UDP, TCP, FTP, HTTP...)
  WIFI.setConnectionOptions(UDP);
  // 2. Configure the way the modules will resolve the IP address.
  WIFI.setDHCPoptions(DHCP_ON);
}

void loop(){
  // 3. Configure how to connect the AP.
  WIFI.setJoinMode(MANUAL);
  while(1){
    // 3.1 Call join giving the name of the AP.
    while(!WIFI.join("ANDROID")){}
    // Switches on green led to show us it's connected.
    Utils.setLED(LED0, LED_ON);
    // 4. Creates UDP connection.
    if (WIFI.setUDPclient("255.255.255.255",12345,2000))
    {
      // 5. Now we can use send and read functions to send and
      // receive UDP messages.
      while(1){
        // 6. Send data to the IOS smartphone
        sprintf(tosend,"Wasp-1;19;20;21;22;23;24;%d;%d;%d;",
                ACC.getX(),ACC.getY(),ACC.getZ());
        WIFI.send(tosend);
        // Reads data to the IOS smartphone
        WIFI.read(NOBLO);
        // Waspmote acts depending on the answer.
        uint8_t period = (WIFI.answer[6] - 48) * 10 + (WIFI.answer[7] - 48);
        if (periodo<11){
          // Switches on a lamp with intensity= period %
        }
      }
    }
  }
}
```

Anexo D

Pruebas Realizadas

En este anexo se describe todo lo relacionado con las pruebas realizadas que se realizan para comprobar el correcto funcionamiento del módulo Wi-Fi. También se describe el código usado para las pruebas.

D.1. Puntos de Acceso

Las pruebas relacionadas con los puntos de acceso se realizan sobre tres modelos diferentes de router, y Meshlium. Y también se comprueban los 3 tipos de encriptación que el módulo Wi-Fi soporta.

Los tipos de router son:

Figura D.1: Linksys WRT54xx

Figura D.2: Cisco Wrv210

Figura D.3: Zixel 660H

Figura D.4: Meshlium Extreme

En cuanto a las encriptaciones, el módulo WIFI soporta encriptación WEP, WPA y WPA2. En el caso de la encriptación WEP, funciona con una clave de 128 bits, es decir, lo que se introduce en la función correspondiente como 13 caracteres ASCII, y el integrado WIFi recibirá la clave en formato hexadecimal. Y en el caso de WPA y WPA2, la encriptación se considera frase en vez de clave, y tiene longitud de 1 a 64 bytes. Puede ser alfanumérica. En el caso de 64 caracteres, se asume que es la representación hexadecimal de la frase PSK de 32 caracteres.

Finalmente, el código utilizado en la prueba es el siguiente:

```
/*
 *  ------Waspmote WIFI Test AP routers Example--------
 *
 *
 *  Copyright (C) 2012 Libelium Comunicaciones Distribuidas S.L.
 *  http://www.libelium.com
 *
 *  Implementation:         Joaquin Ruiz
 */

void setup(){
  // First, initialize the WIFI API and the connections with the waspmote
  WIFI.begin();
  // Then switch on the WIFI module on the desired socket.
  WIFI.ON(socket0);
  // If we don't know what configuration had the module, reset it.
  WIFI.resetValues();

  // 1. Configure the transport protocol (UDP, TCP, FTP, HTTP...)
  WIFI.setConnectionOptions(CLIENT_SERVER|CLIENT|UDP|HTTP);
  // 2. Configure the way the modules will resolve the IP address.
  WIFI.setDHCPoptions(DHCP_ON);
}

void loop(){
// *** MANUAL-MODE ***
  // 3. Configure how to connect the AP.
  WIFI.setJoinMode(MANUAL);
  // 3.1 If it is manual, call join giving the name of the AP.
  if (WIFI.join("libelium_AP"))
    // 4. Call the function that needs a connection.
    WIFI.resolve("www.libelium.com");

// *** AUTO-BEST ***
  // 3.0 Configure the Authentication mode of the auto-join.
  WIFI.setAutojoinAuth(OPEN);
  // 3. Configure how to connect the AP.
  if (WIFI.setJoinMode(AUTO_BEST))
    // 4. Call the function that needs a connection.
    WIFI.resolve("www.libelium.com");

// *** AUTO-STORE ***
  // 3.0 Configure the number of channel where we want to join
  WIFI.setChannel(6);
  // 3.1 Configure the authentication mode of the auto-join.
  WIFI.setAutojoinAuth(OPEN);
  // 3.2 Sets the name of the AP we want to join.
  WIFI.setESSID("libelium_AP");
  // 3. Configure how to connect the AP.
  if (WIFI.setJoinMode(AUTO_STOR))
    // 4. Call the function that needs a connection.
    WIFI.resolve("www.libelium.com");
}
```

En este caso, Waspmote se conecta a un punto de acceso y hace una petición DNS mediante los 3 modos que se han incluido en la API (Manual, Automático y Automático con caché). Mediante el comando ***millis()*** se puede sacar el tiempo del procesador y calcular la diferencia para sacar el tiempo final.

Se hacen diez pruebas para cada tipo de router y encriptación y se saca la media.

D.2. Conexión con Internet

Como se introduce en la memoria, la prueba de conexión con Internet se realiza de diferentes maneras para cada tipo de protocolo.

Mediante el programa Java desarrollado para TCP, UDP y UDP Broadcast. Un archivo PHP subido a la web de la compañía para HTTP, y finalmente, un servidor FTP para el protocolo del mismo nombre.

TCP, UDP, UDP Broadcast.

La comprobación de estos protocolos se realiza mediante el programa Java desarrollado. Se indica al módulo WIFi la dirección IP del PC donde está el programa corriendo y el puerto configurado, entonces el módulo WIFi se conecta al punto de acceso y abre la conexión en el caso de TCP, manda un datagrama en el caso de UDP, o configura Broadcast-UDP.

A continuación se muestran los resultados del Test:

El radio WIFI abre conexión TCP con el puerto 3550 del PC y manda "Hi From Waspmote through WIFI!". El mensaje *HELLO* es un mensaje por defecto que manda el radio WIFi cuando establece conexión TCP.

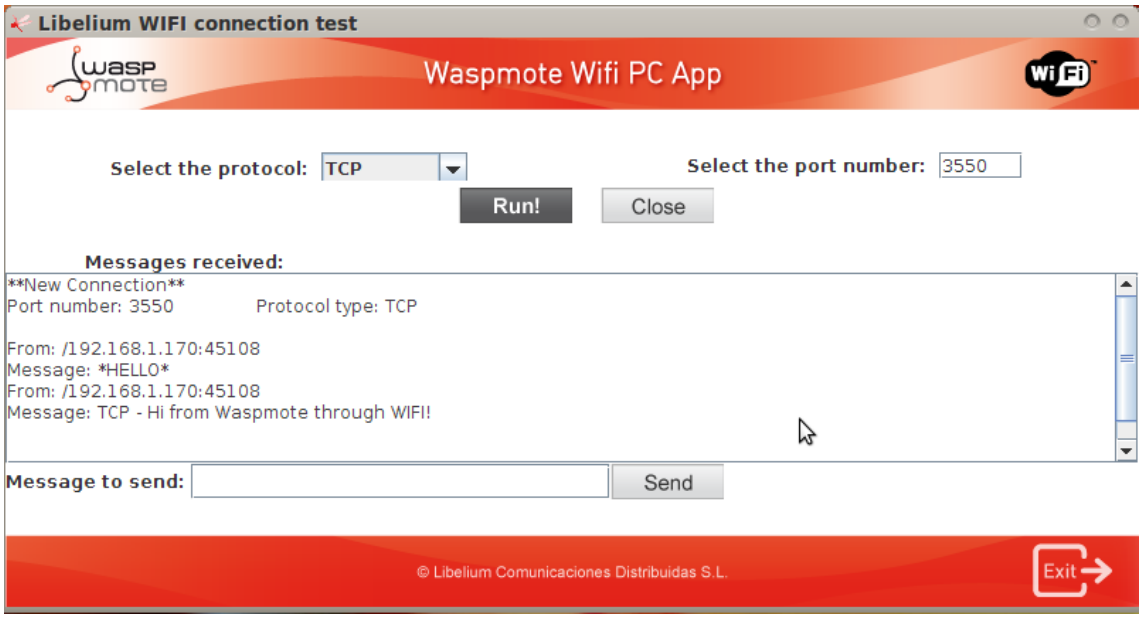

Figura D.5: Resultados del test PC con protocolo TCP

El radio WIFI envía un datagrama UDP al puerto 2500 del PC con el contenido "UDP hola desde el módulo WIFI!!"

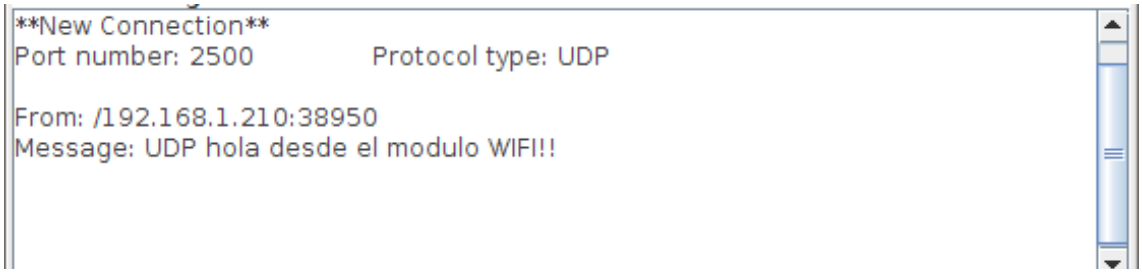

Figura D.6: Resultados del test PC con protocolo UDP

El radio WIFI envía datagramas UDP al puerto 55555 del PC con el contenido por defecto del Broadcast-UDP (110 Bytes de datos) y el programa presenta de forma ordenada dichos datos.

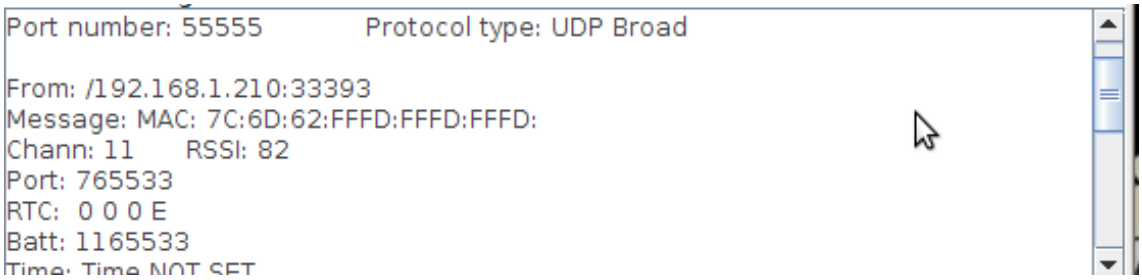

Figura D.7: Resultados del test PC con protocolo UDP-Broadcast

HTTP

Para comprobar el funcionamiento del protocolo HTTP, se desarrolla un archivo PHP sencillo, que se aloja en el servidor de www.libelium.com y desde el módulo WIFi se hace una petición GET por protocolo HTTP seleccionando la IP del servidor, y otra petición igual por protocolo HTTP y protocolo DNS seleccionando el nombre del servidor.

A continuación se muestra el resultado de llamar desde Waspmote a:

```
getURL(DNS,www.libelium.com,"GET$/radioWIFItest.php?name=Joaquin");
```

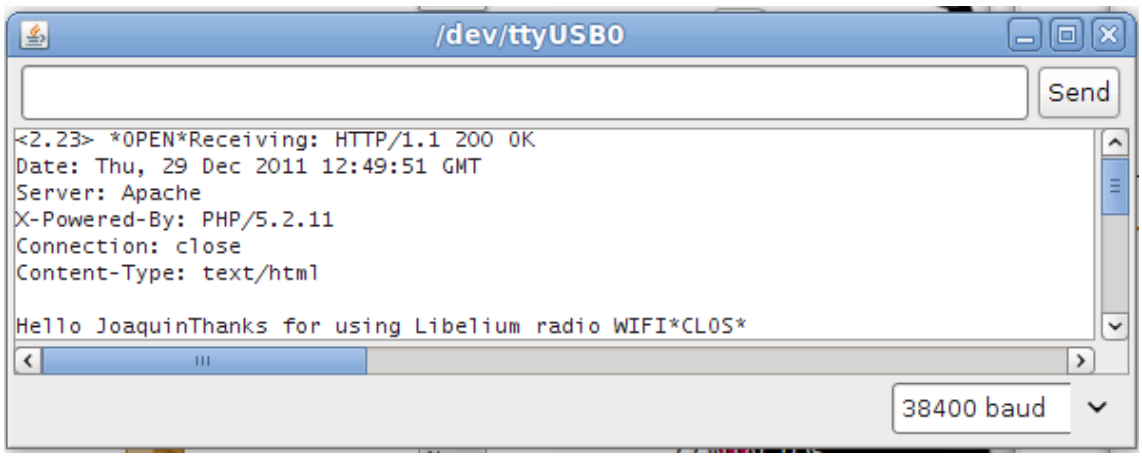

Figura D.8: Resultados del test HTTP

FTP

Finalmente, para comprobar el funcionamiento del protocolo FTP, se suben unos archivos sencillos de texto al servidor FTP y sabiendo el nombre de usuario, la contraseña y las carpetas en los que está alojado se configura el módulo WIFi y la tarjeta SD. A continuación se muestra parte del código utilizado para esta prueba.

```
// Establece la dirección FTP, puerto, etc.
WIFI.setFTP("62.75.203.94",21,FTP_PASIVE,20);
// Prepara el servidor escribiendo el nombre de usuario y contraseña
WIFI.openFTP("wifimodu","cro3N2C20h");
// Configura como conectarse a la red
WIFI.setJoinMode(MANUAL);
// Al ser manual, llama explícitamente a conectar a la red.
if(WIFI.join("TEST_WIFI")){
  // Sube archivo
  WIFI.uploadFile("FileText.txt","Folder","public_ftp/Folder")!=1){}
  // Descarga archivo
  WIFI.getFile("documento","/","public_ftp")!=1){}
}
```

Y se comprueba que efectivamente, "Filetext.txt" está en la carpeta *public_ftp/Folder* del servidor y "documento" está en la tarjeta SD.

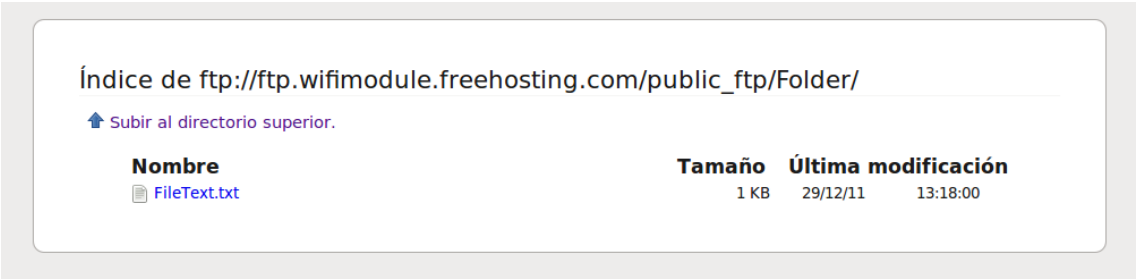

Figura D.9: Resultados del test FTP (servidor)

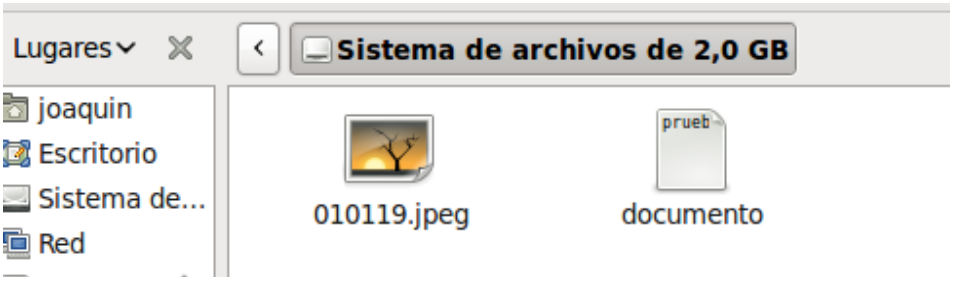

Figura D.10: Resultados del test FTP (tarjeta SD)

D.3. Conexión entre módulos Wi-Fi

El código utilizado para esta prueba es el siguiente:

```
// Configura la IP en modo WIFI Ad-hoc
WIFI.setDHCPoptions(AUTO_IP);
// Configura seguridad básica Ad-hoc
WIFI.setAutojoinAuth(ADHOC);
// Indica el nombre y canal de la red Ad-hoc
WIFI.setESSID("ADHOC_JOKI"); WIFI.setChannel(6);
// Llama a crear/unirse a la red Ad-hoc
if(WIFI.setJoinMode(CREATE_ADHOC)){
  // Manda o recibe mensajes de la red Ad-hoc
  while(1){ WIFI.send("¡Hola mundo!"); WIFI.read(); }
}
```

En este caso, se han hecho 10 pruebas a distintas distancias (dentro de un rango 5 a 20 m), y se saca la conclusión de que el tiempo de conexión a la red Ad-hoc por parte de un módulo WIFi con Auto-IP es de unos 10 segundos.

D.4. Conexión con otro dispositivo Wi-Fi

Aquí se introducen las primeras aplicaciones para Smartphone que se desarrollaron para comprobar el correcto funcionamiento de la conexión con otro dispositivo Wi-Fi.

Antes de la aplicación final, se desarrollaron:

- Aplicación GPS para comprobar la correcta comunicación entre Waspmote y dispositivos iPhone y Android.
- Aplicación Actuador para comprobar la correcta comunicación entre iPhone y Waspmote.

Finalmente se desarrolló la aplicación que figura en la web de Libelium, que comprende ambos aspectos.

iPhone (iOS)

A continuación se muestra una captura del iPhone conectado a la red WIFi Ad-hoc creada por un radio WIFi del Waspmote. Esta **primera aplicación** muestra el nombre de los dispositivos conectados y el mensaje que manda cada uno, en este caso la batería y posición GPS.

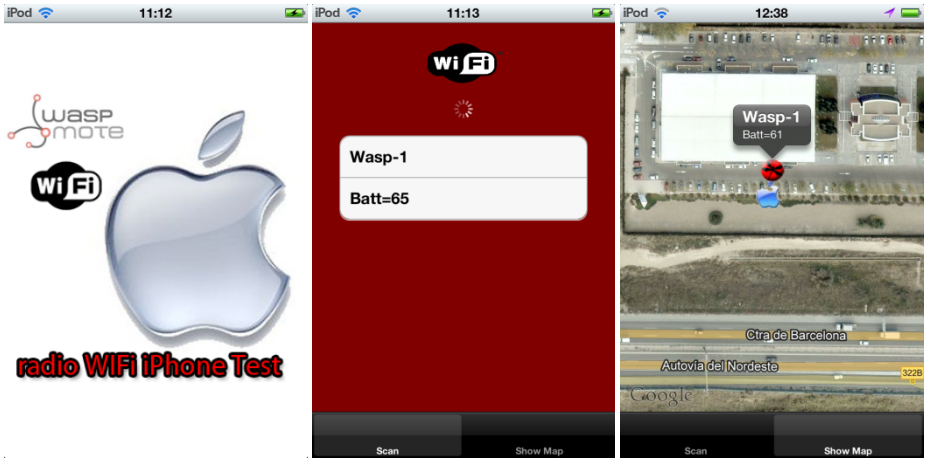

Figura D.11: Resultados del primer test iPhone

En este primer caso el *iPhone/iPad* actúa como cliente pasivo que solo recibe información, y no manda ni envía nada a la red WIFi Ad-hoc.

Por ello, se desarrolla una **segunda aplicación** para *iPhone/iPad* que controla un dispositivo Waspmote mandándole mensajes a través de datagramas UDP una vez conectado a la red WIFi Ad-hoc que crea el radio WIFi del Waspmote.

El funcionamiento es el siguiente: se configura Broadcast-UDP en el radio WIFi, y así cuando se quiera comunicar algo a un dispositivo de la red sensorial, la aplicación iPhone se pone en modo recibir paquete UDP y cuando le llega el datagrama coge la dirección origen envía un datagrama a dicho dispositivo con el mensaje indicado.

A continuación se muestra una captura de la aplicación, dispone de dos *switches* para encender y apagar 2 leds del Waspmote, y un botón para activar un *zumbador* que se ha conectado a un interruptor digital del Waspmote.

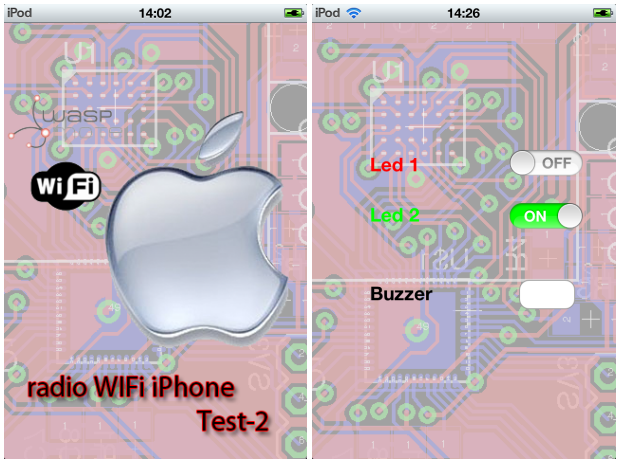

Figura D.12: Resultados del segundo test iPhone

Android

A continuación se muestra una captura del Android conectado al Waspmote a través de la red creada por el móvil. En este caso se muestra el nombre de los dispositivos conectados a la red del móvil Android y el mensaje que manda cada uno, en este caso la batería y posición GPS.

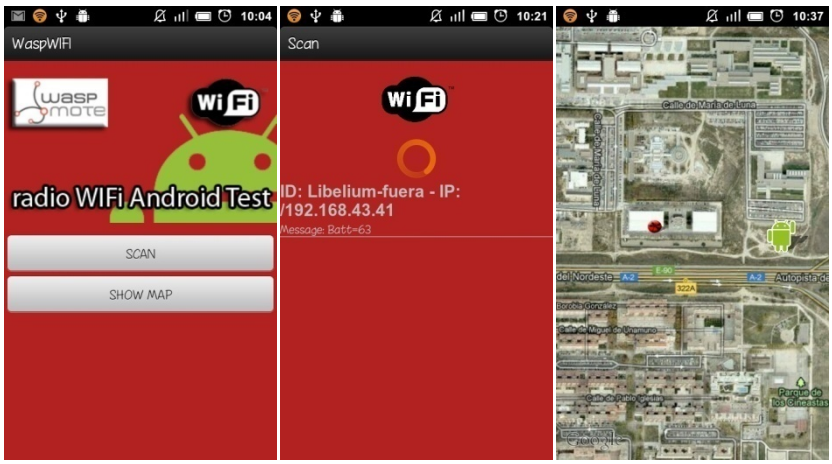

Figura D.13: Resultados del test Android

D.5. Alcance

El alcance del dispositivo WIFi depende principalmente de la antena conectada al conector SMA. Cualquier antena con conector SMA puede ser acoplada al módulo WIFi, por ello se ha acotado la comparativa a dos tipos de antenas, 2 dBi y 5 dBi.

Para la realización de esta prueba se buscó una localización idónea, optimizando la zona *Fresnel* entre los *Waspmote* haciendo dichas pruebas en calles y carreteras rectas. Para ello se utilizó *google maps* como herramienta que calcula distancias entre varios puntos.

Para la localización se eligió la calle del Poeta Luciano Gracia, es una recta que va desde el puente de la autopista hasta la rotonda de Juslibol, cerca de la Escuela de Ingeniería y Arquitectura.

Figura D.14: Localización elegida para la prueba de alcance del módulo WIFi

Los rangos de distancia difieren desde una transmisión perfecta, 100% mensajes recibidos, hasta donde es visible el punto de acceso creado por un dispositivo para comunicación Ad-hoc.

A continuación se muestran los resultados obtenidos en los diferentes casos:

- **Modo Infraestructura**
  En este caso, se conecta el módulo WIFi al punto de acceso creado por el *Meshlium* y hace un ping a un servidor de internet.

Dispositivos	Distancia
Antena 2 dBi con *Meshlium*	23 mts.
Antena 5 dBi con *Meshlium*	~120 mts.

Tabla D.1: Tabla distancia modo Infraestructura

- **Modo Ad-hoc (Visibilidad Punto Acceso)**
  En este caso, un dispositivo crea una red Ad-hoc, y el otro hace escaneados de puntos de acceso.

Dispositivos	Distancia
2 dBi con 2 dBi	200 mts.
2 dBi con 5 dBi	600 mts.
5 dBi con 5 dBi	800 mts.

Tabla D.2: Tabla distancia modo Ad-hoc (visibilidad)

- **Modo Ad-hoc (Conexión 100% correcta)**
  En este caso, se mira el tiempo que tarda hacer un envío de datos mediante una red Ad-hoc compartida por ambos dispositivos. Entonces se van separando dichos dispositivos, y cuando el tiempo transcurrido es mayor a ese tiempo medido nos indica que algún paquete ha sido perdido.

Dispositivos	Distancia
2 dBi con 2 dBi	20 mts
2 dBi con 5 dBi	160 mts
5 dBi con 5 dBi	350 mts

Tabla D.3: Tabla distancia modo Ad-hoc (100% datos recibidos)

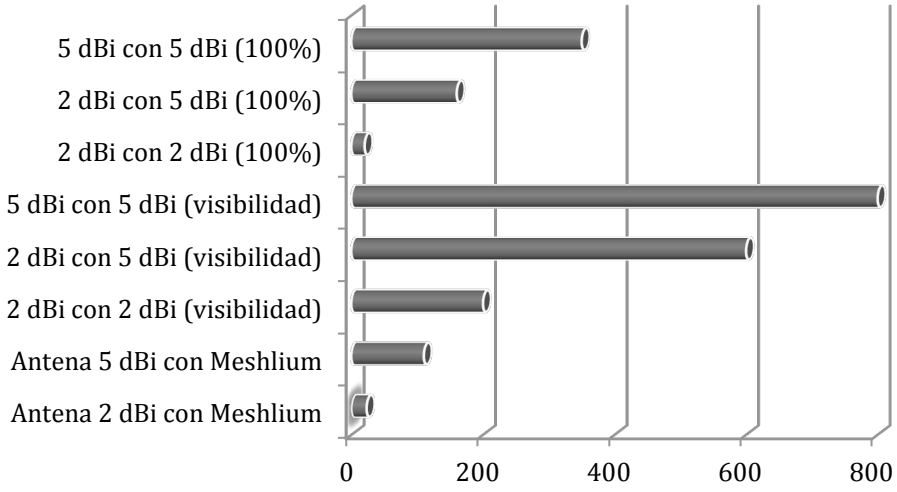

Figura D.15: Gráfico comparativo de distancias (en metros)

Anexo E

Software Desarrollado

En este anexo se describe todo lo relacionado con el software desarrollado para la comunicación con el módulo Wi-Fi de una forma sencilla.

Para conectar el módulo Wi-Fi a otro dispositivo que no sea el propio módulo Wi-Fi se ha desarrollado una serie de software asociado para poder comprobar todas las opciones de conexión que ofrece la tecnología Wi-Fi.

E.1. Software PC

En esta sección se describe el programa multiplataforma que se comunica con el módulo Wi-Fi del *Waspmote* mediante UDP, TCP o traduciendo el contenido del Broadcast-UDP a través de una red infraestructurada.

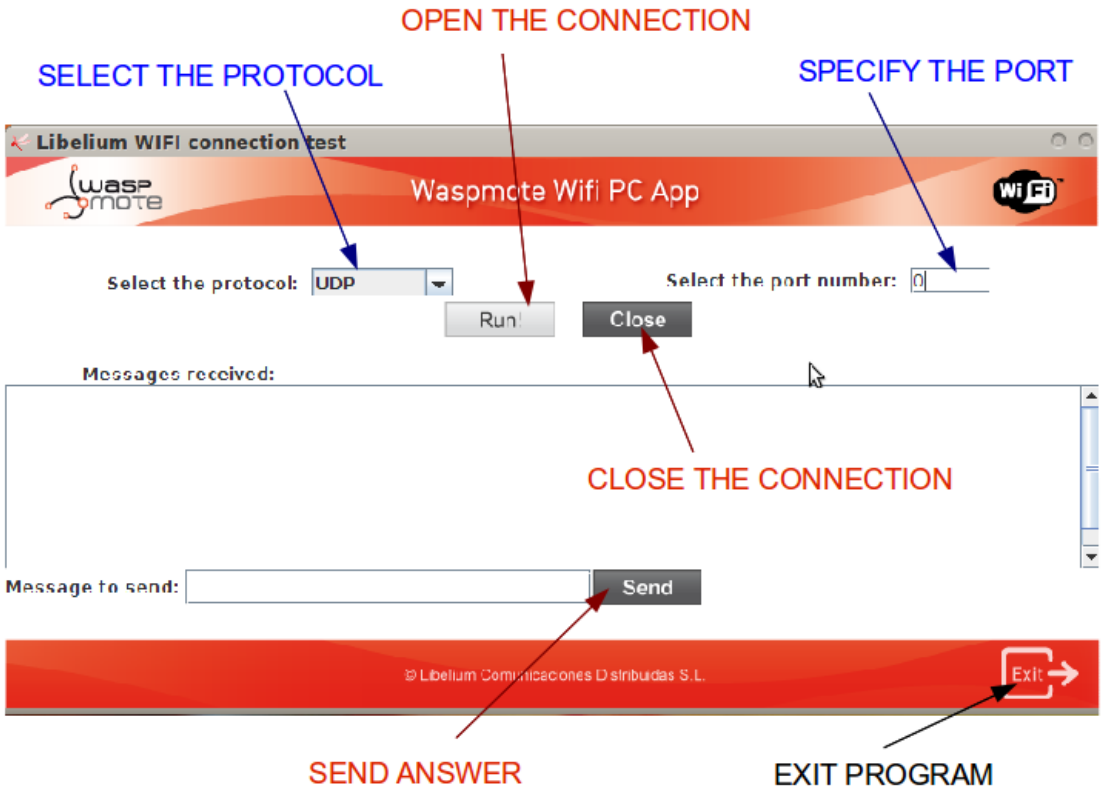

Figura E.1: Descripción componentes del programa PC

El funcionamiento del software es simple.

Primero hay que seleccionar el puerto y el tipo de protocolo en el programa e indicar la IP del ordenador en el código que se introduce al *Waspmote.*

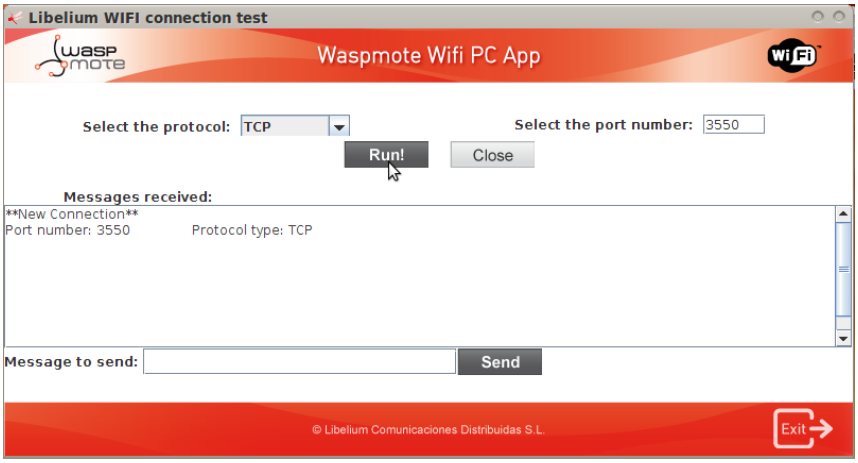

Figura E.2: Pantallazo 1 programa PC

A continuación se recibe un mensaje desde Waspmote.

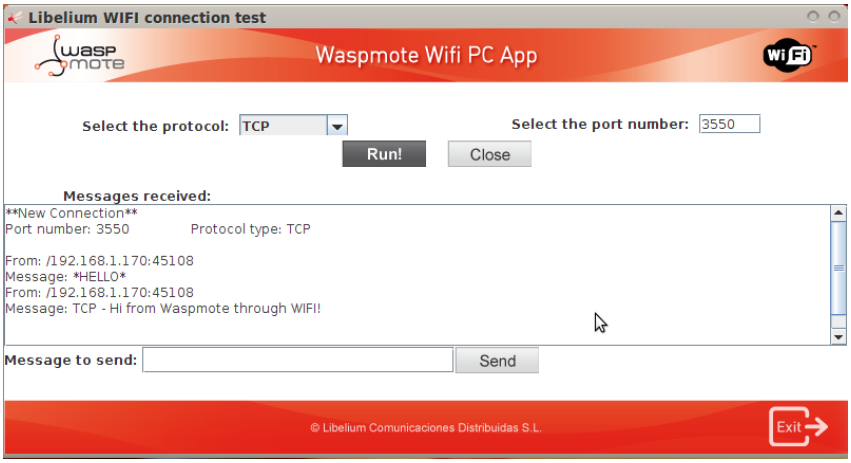

Figura E.3: Pantallazo 2 programa PC

Una vez recibido el mensaje, se puede contestar a Waspmote.

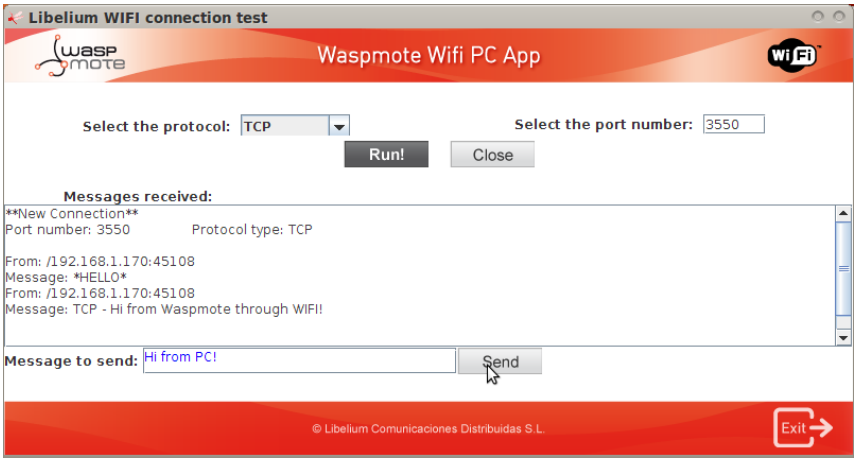

Figura E.4: Pantallazo 3 programa PC

En cuanto al **código del software PC**, es un programa escrito en Java bajo una interfaz gráfica utilizando librerías Java-Swing y un socket corriendo en un *Threat* utilizando librerías Java-Net.

Y finalmente, se muestra un resumen del código que se carga en el **Waspmote**. En este caso es el código para el protocolo TCP.

```
void loop(){
  // Se conecta al punto de acceso correspondiente para tener
  // conexión a la red.
  if (WIFI.join("PRUEBA_WIFI"))
  {
    // Crea la conexión TCP
    if (WIFI.setTCPclient("192.168.1.210",3550,2000))
    {
      // Manda y recibe mensajes
      while(1){
        WIFI.send("TCP hola desde el modulo WIFI!!");
        WIFI.read();
      }
      // Ciera la conexión TCP.
      WIFI.close();
    }
  }
}
```

Finalmente, se han desarrollado ejecutables para Linux, Mac OS y Windows.

Figura E.5: Ejecutables programa PC

E.2. Aplicación iPhone

Se han desarrollado aplicaciones iPhone para conectar el dispositivo móvil con *Waspmote* sin necesidad de un router intermedio, creando una red Ad-hoc entre ellos.

En este apartado se explica la última aplicación desarrollada, que es la más completa.

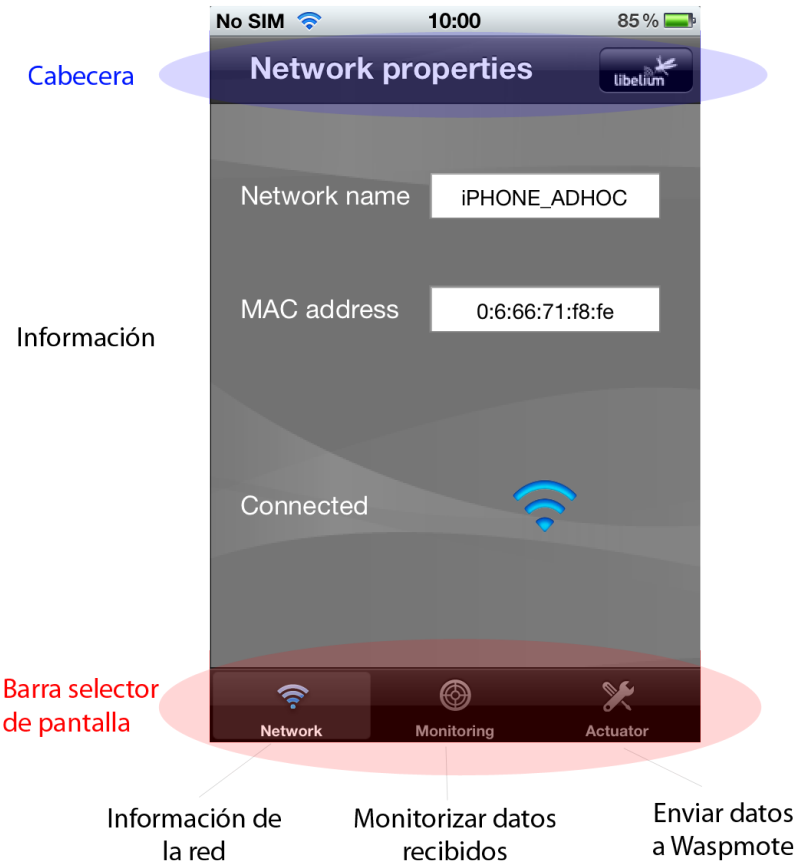

Figura E.6: Pantalla aplicación iPhone.

La aplicación está estructurada de la siguiente manera:

- Un controlador (AppDelegate) controla la barra inferior con la que se puede desplazar por las 3 pantallas desarrolladas. También lanza una clase (EchoUDP) que se comunica mediante mensajes UDP con el módulo Wi-Fi.
- La primera pantalla (FirstView) muestra los datos de la conexión.
- La segunda pantalla (SecondView) muestra los datos recibidos a través de la conexión UDP. También lanza un programa Open-GL.
- La tercera pantalla (ThirdView) especifica los mensajes que se lanzan a través de la conexión UDP.

La comunicación entre dichos controladores se realiza mediante ficheros de comunicación en la memoria caché del dispositivo iPhone/iPad.
La instalación de la aplicación es muy sencilla. Se puede descargar la aplicación desde App Store a iTunes o directamente al dispositivo iPhone/iPod/iPad.

Una vez instalada, la app aparece en la pantalla de su iPhone.

Figura E.7: Icono aplicación iPhone.

A continuación se va a mostrar un pequeño tutorial de uso de la aplicación.

Primero hay que conectar el dispositivo iPhone a uno de los nodos Waspmote. Para ello, se selecciona módulo Wi-Fi en Ajustes->Wi-Fi y seleccionar la red deseada (en este caso, iPHONE_ADHOC).

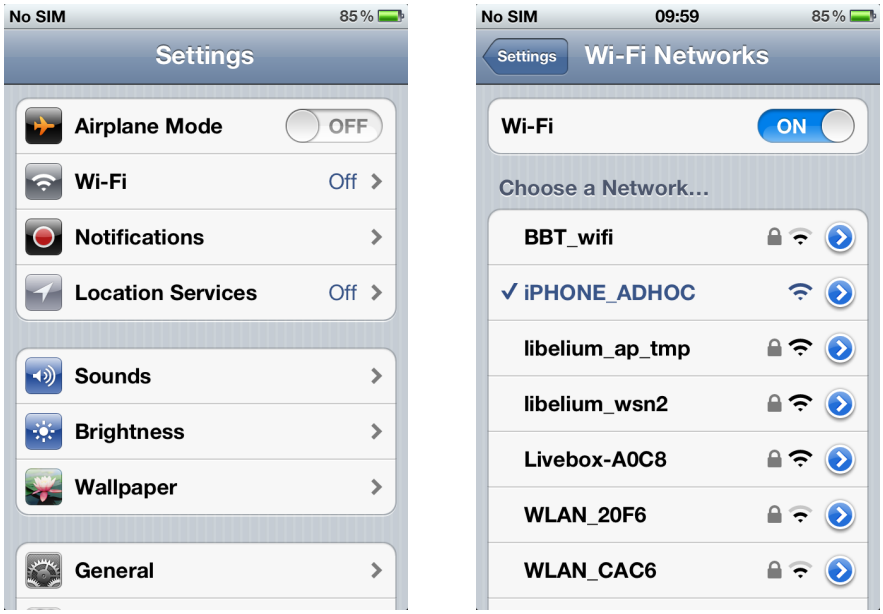

Figura E.8: Conexión a la red Waspmote desde iPhone.

Una vez conectado, se verá un icono azul en la parte superior de la pantalla.

Figura E.9: Icono Wi-Fi azul en iPhone.

Dentro de la app, la primera pestaña "Network" (Red) muestra la información de la conexión.

- Nombre de la red Ad-hoc.
- Dirección MAC del nodo que actúa como puerta de enlace.
- Estado de la conexión.

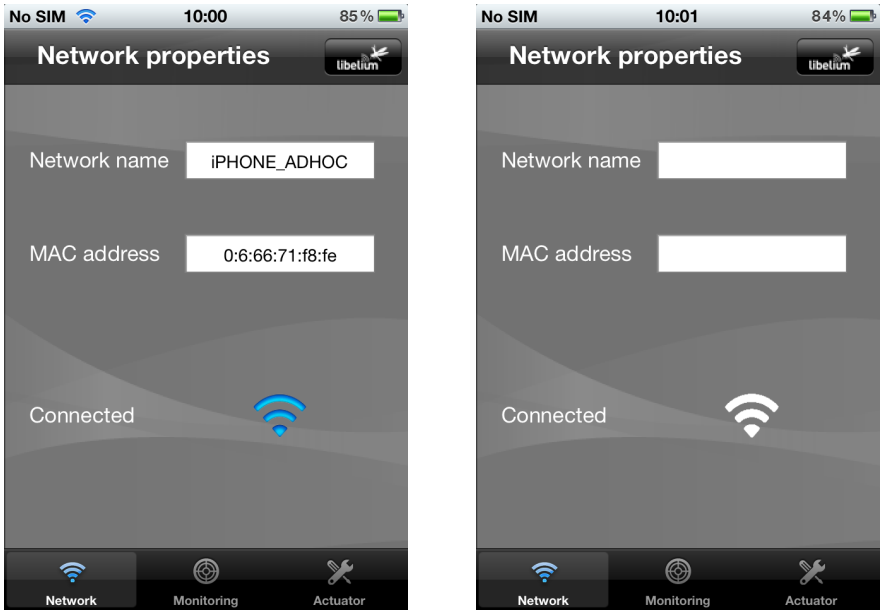

Figura E.10: Pantalla red de la aplicación iPhone.

La pantalla "Monitoring" (Monitorización) muestra la información que los nodos están mandando de los sensores que contienen. En este caso se muestra: temperatura, humedad, oxígeno, dióxido de carbono, sensor de presencia, y un modelo en 3 dimensiones de *Waspmote* que se mueve con la información que proporciona el acelerómetro de *Waspmote*.

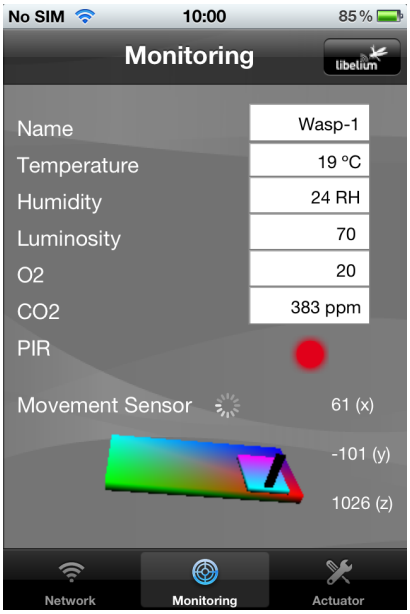

Figura E.11: Pantalla monitorización de la aplicación iPhone.

Finalmente, en la pantalla "Actuator" (Actuador), hay tres interruptores que mandan mensajes con ON u OFF, y una barra desplazadora que manda un mensaje con el valor exacto. En la página web de Libelium hay un vídeo que muestra cómo la aplicación trabaja con un set de luces.

Figura E.12: Pantalla actuador de la aplicación iPhone.

A continuación se muestra el código del dispositivo Waspmote que se ha utilizado en este ejemplo:

```
/*
 *  ------Waspmote WIFI IOS smartphone application Example--------
 *
 *  Explanation: This example shows the way to communicate with
 *  the Waspmote Wifi Demo iPhone app.
 *
 *  Copyright (C) 2012 Libelium Comunicaciones Distribuidas S.L.
 *  http://www.libelium.com
 *  Implementation:         Joaquin Ruiz
 */

// Specifies the message that is sent to the WiFi module.
char tosend[128];

void setup(){
  // Initialize the accelerometer
  ACC.begin();
  ACC.setMode(ACC_ON);

  // Initialize the WIFI API and the connections with Waspmote
  WIFI.begin();
  // Then switch on the WIFI module on the desired socket.
  WIFI.ON(socket0);
  // If we don't know what configuration had the module, reset it.
  WIFI.resetValues();

  // 1. Configure the transport protocol (UDP, TCP, FTP, HTTP...)
  WIFI.setConnectionOptions(CLIENT_SERVER|UDP);
  // 2. Configure the way the modules will resolve the IP address.
  WIFI.setDHCPoptions(AUTO_IP);
  // 3. Configure how to connect the AP.
  WIFI.setAutojoinAuth(ADHOC);
  // 3.1 Sets the name of the ADhoc network.
  WIFI.setESSID("iPHONE_ADHOC");
  // 3.2 Sets the channel of the ADhoc network
  WIFI.setChannel(6);
}

void loop(){
  // 3.3 Call function to create/join the Adhoc network.
  if(WIFI.setJoinMode(CREATE_ADHOC)){
    // Switches on green led.
    Utils.setLED(LED0, LED_ON);
    // 4. Creates UDP connection.
    if (WIFI.setUDPclient("255.255.255.255",12345,2000)){
      // 5. Now we can use send and read functions to send and
      // receive UDP messages.
      while(1){
        // Sends data to the IOS smartphone
        sprintf(tosend,"Wasp-1;19;24;70;20;383;0;%d;%d;%d",
                ACC.getX(),ACC.getY(),ACC.getZ());
        WIFI.send(tosend);
        // Reads data to the IOS smartphone
```

```
        WIFI.read(NOBLO);
        // Waspmote acts depending on the answer.
        uint8_t period = (WIFI.answer[6] -
 48) * 10 + (WIFI.answer[7] - 48);
        if (period<11){
          // Switches on a lamp with intensity= period % (...)
        }
        delay(100);
      }
    }
  }
}
```

Estructura mensaje comunicación iPhone -> Waspmote:

0/1 (Interruptor 1) ; 0/1 (Interruptor 2) ; 0/1 (Interruptor 3) ; %% (barra despl)

Estructura mensaje comunicación Waspmote -> iPhone:

Nombre ; Temperatura ; Humedad ; Luminosidad ; Oxígeno ; Dióxido de carbono ; Sensor de presencia ; Acelerómetro (x) ; Acelerómetro (y) ; Acelerómetro (z)

E.3. Aplicación Android

Se han desarrollado aplicaciones Android para conectar el dispositivo móvil con *Waspmote* sin necesidad de un router intermedio, creando una red Ad-hoc entre ellos.

En este apartado se explica la última aplicación desarrollada, que es la más completa.

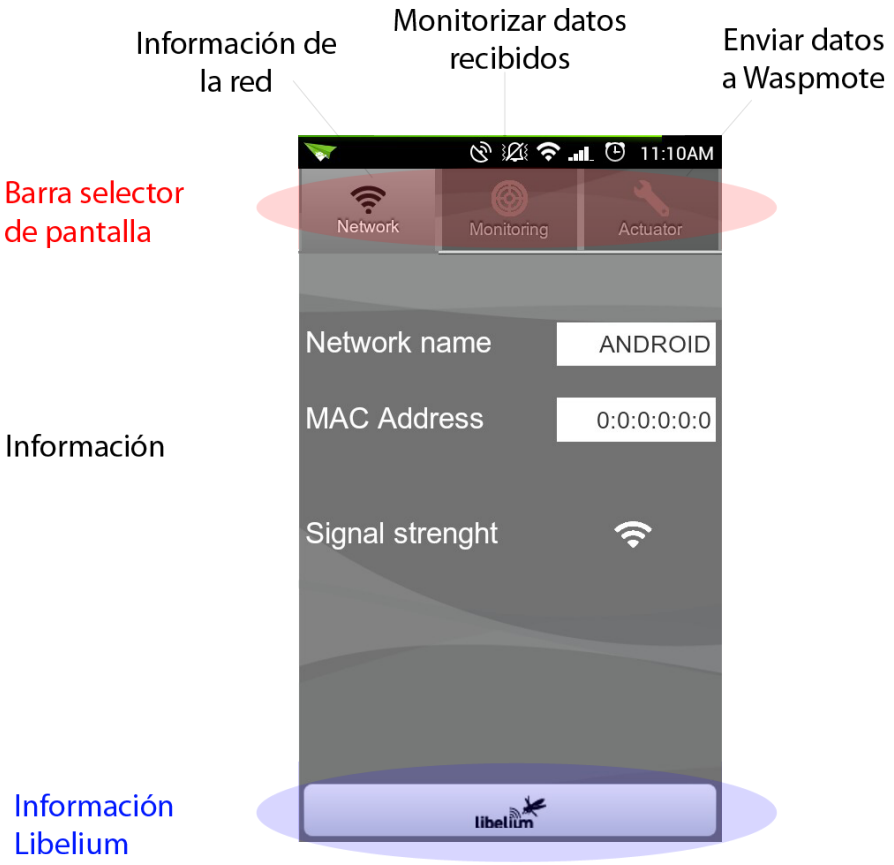

Figura E.13: Pantalla aplicación Android.

La aplicación está estructurada de la siguiente manera:

- Un controlador (WiFiActivity) controla la barra superior con la que se puede desplazar por las 3 pantallas desarrolladas. También lanza un hilo (Threat UDP) que se comunica mediante mensajes UDP con el módulo Wi-Fi.

- La primera pantalla (NetworkActivity) muestra los datos de la conexión.

- La segunda pantalla (MonitorActivity) muestra los datos recibidos a través de la conexión UDP. También lanza un programa Open-GL.

- La tercera pantalla (ControlActivity) especifica los mensajes que se lanzan a través de la conexión UDP.

La instalación de la aplicación es muy sencilla. Se puede descargar la aplicación desde Android Market directamente al dispositivo Android.

Una vez instalada, la app aparece en la pantalla de su Android.

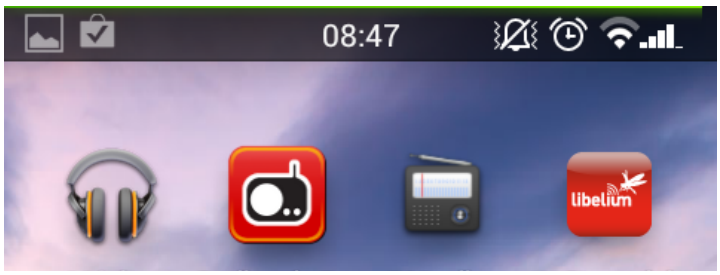

Figura E.14: Icono aplicación Android.

A continuación se va a mostrar un pequeño tutorial de uso de la aplicación.

Primero se tienen hay que conectar el dispositivo Android a uno de los nodos Waspmote. Para ello, se selecciona Ajustes->Wi-Fi->Mi Zona Wi-Fi y crear un punto de acceso al que se conectará el dispositivo Waspmote.

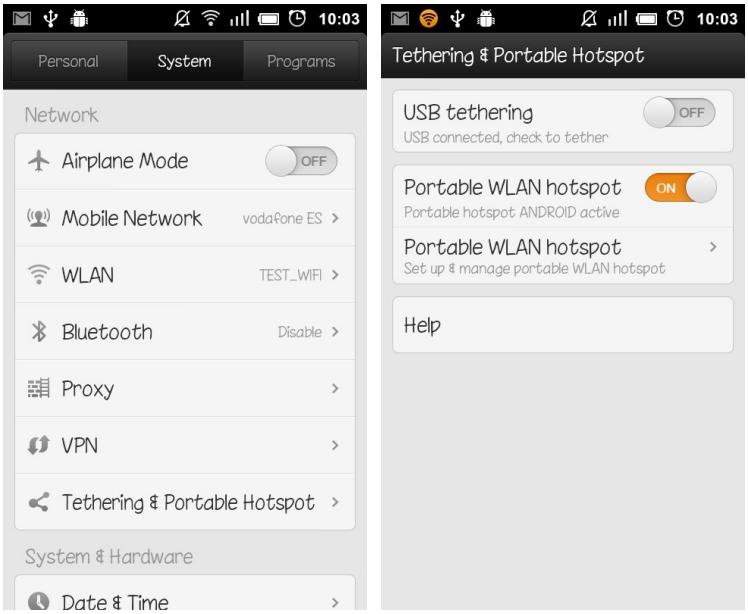

Figura E.15: Creación de la red Ad-hoc desde Android.

A continuación hay que configurar el punto de acceso WLAN (nombre Android, seguridad: Abierta). Estos ajustes se pueden cambiar si se cambia también el código de Waspmote.

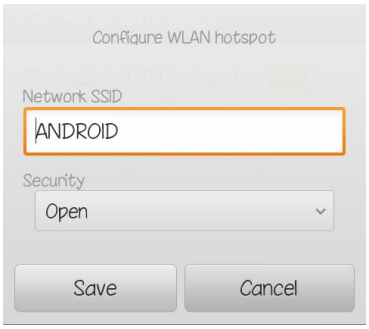

Figura E.16: Configuración de la red Ad-hoc desde Android.

Dentro de la app, la primera pestaña "Network" (Red) muestra la información de la conexión.

- Nombre de la red Ad-hoc.
- Dirección MAC del nodo que actúa como puerta de enlace.
- Estado de la conexión.

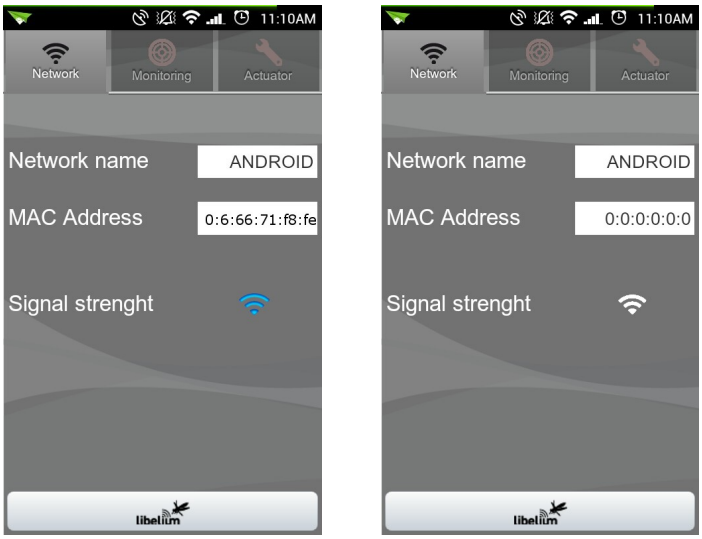

Figura E.17: Pantalla red de la aplicación Android.

La pantalla "Monitoring" (Monitorización) muestra la información que los nodos están mandando de los sensores que contienen. En este caso se muestra: temperatura, humedad, oxígeno, dióxido de carbono, sensor de presencia, y un modelo en 3 dimensiones de *Waspmote* que se mueve con la información que proporciona el acelerómetro de *Waspmote*.

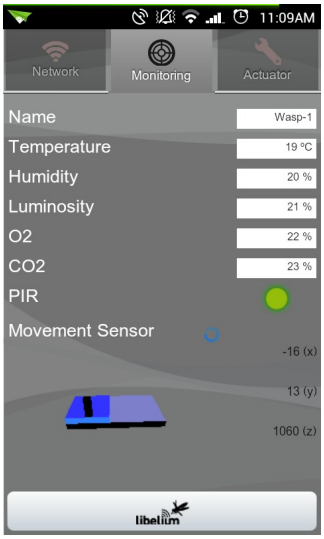

Figura E.18: Pantalla monitorización de la aplicación Android.

Finalmente, en la pantalla "Actuator" (Actuador), hay tres interruptores que mandan mensajes con ON u OFF, y una barra desplazadora que manda un mensaje con el valor exacto. En la página web de Libelium hay un vídeo que muestra cómo la aplicación trabaja con un set de luces.

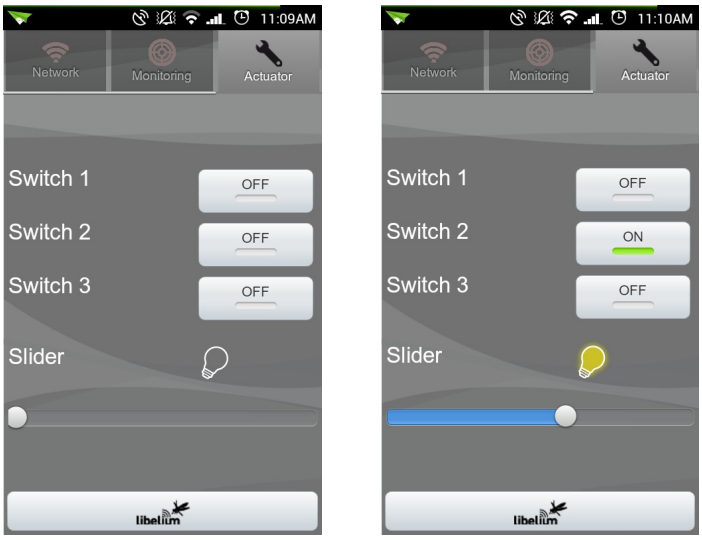

Figura E.19: Pantalla actuador de la aplicación Android.

A continuación se muestra el código del dispositivo Waspmote que se ha utilizado en este ejemplo:

```
/*
 *  ------Waspmote WIFI Android smartphone application Example--------

 *
 *  Explanation: This example shows the way to communicate with
 *  the Waspmote Wifi Demo Android app.
 *
 *  Copyright (C) 2012 Libelium Comunicaciones Distribuidas S.L.
 *  http://www.libelium.com
 *  Implementation:         Joaquin Ruiz
 */

// Specifies the message that is sent to the WiFi module.
char tosend[128];

void setup(){
  // Initialize the accelerometer
  ACC.begin();
  ACC.setMode(ACC_ON);
  // First, initialize the WIFI API and the connections with the waspm
ote
  WIFI.begin();
  // Then switch on the WIFI module on the desired socket.
  WIFI.ON(socket0);
  // If we don't know what configuration had the module, reset it.
  WIFI.resetValues();

  // 1. Configure the transport protocol (UDP, TCP, FTP, HTTP...)
  WIFI.setConnectionOptions(UDP);
  // 2. Configure the way the modules will resolve the IP address.
  WIFI.setDHCPoptions(DHCP_ON);
}

void loop(){
  // 3. Configure how to connect the AP.
  WIFI.setJoinMode(MANUAL);
  while(1){
    // 3.1 Call join giving the name of the AP.
    while(!WIFI.join("ANDROID")){}
    // Switches on green led to show us it's connected.
    Utils.setLED(LED0, LED_ON);
    // 4. Creates UDP connection.
    if (WIFI.setUDPclient("255.255.255.255",12345,2000))
    {
      // 5. Now we can use send and read functions to send and
      // receive UDP messages.
      while(1){
        // 6. Send data to the IOS smartphone
        sprintf(tosend,"Wasp-1;19;20;21;22;23;24;%d;%d;%d;",
                ACC.getX(),ACC.getY(),ACC.getZ());
        WIFI.send(tosend);
        // Reads data to the IOS smartphone
        WIFI.read(NOBLO);
        // Waspmote acts depending on the answer.
        uint8_t period = (WIFI.answer[6] -
 48) * 10 + (WIFI.answer[7] - 48);
        if (periodo<11){
          // Switches on a lamp with intensity= period %
        }
      }
    }
```

```
    }
}
```

Estructura mensaje comunicación Android -> Waspmote:

0/1 (Interruptor 1) ; 0/1 (Interruptor 2) ; 0/1 (Interruptor 3) ; %% (barra despl)

Estructura mensaje comunicación Waspmote -> Android:

Nombre ; Temperatura ; Humedad ; Luminosidad ; Oxígeno ; Dióxido de carbono ; Sensor de presencia ; Acelerómetro (x) ; Acelerómetro (y) ; Acelerómetro (z)

Anexo F

Fases de Desarrollo

En este anexo se describen las fases de desarrollo de la realización de este PFC. Primero se presenta la distribución semanal, y a continuación la distribución por control de esfuerzos.

F.1. Distribución Semanal.

Se presentan por semanas las fases completas del desarrollo del proyecto, desde la primera semana de Septiembre hasta la segunda semana de Marzo:

Mes 1: Septiembre.

Semana 5 Septiembre.
Toma de Contacto con Waspmote: Lectura de manuales y aprendizaje de uso de los entornos de programación Waspmote IDE, Arduino.

Semana 12 Septiembre.
Se sigue la toma de contacto. Se empiezan las pruebas con los dos tipos de integrados Wi-Fi, de los que hay que elegir uno. Esto sirve par conocer los comandos de los integrados y cómo se conectan dichos módulos para comunicarse con ellos.

Semana 19 Septiembre.
Se sigue el estudio de los integrados Wi-Fi, y se empiezan a escribir funciones para dichos integrados.

Semana 26 Septiembre.
Se empieza una primera versión de la API con las funciones para crear conexiones TCP y UDP.

Mes 2: Octubre.

Semana 3 Octubre.
Se sigue trabajando con la primera versión de la API, optimizando el control de los buffer de comunicaciones.

Semana 10 Octubre.
Se controla en la primera versión de la API la memoria RAM utilizada, ya que al ir añadiendo funcionalidades se estaba superando.

Semana 17 Octubre.
Se realiza el software PC en JAVA para controlar el funcionamiento de las funciones TCP y UDP ya realizadas en la API.

Semana 24 Octubre.
Con la ayuda de los compañeros, se realiza el módulo Wi-Fi definitivo basado en el integrado de *Roving Networks*.

Semana 31 Octubre.
Se empieza en pensar cómo desarrollar aplicaciones para *iPhone* y *Android*.

Mes 3: Noviembre.

Semana 7 Noviembre.
Desarrollo de una primera aplicación para *Android*, y se solucionan problemas relacionados con la longitud de mensajes recibidos y enviados en la API.

Semana 14 Noviembre.
Desarrollo de una primera aplicación para *iPhone* (*iOS*), en este caso añado más funcionalidades que las que en principio se hicieron en *Android*, como dibujar la posición de los dispositivos *Waspmote* en un mapa.

Semana 21 Noviembre.
Se adapta lo realizado en la aplicación iPhone para *Android*, y se completa la API con las funciones HTTP, FTP.

Semana 28 Noviembre.
Se sigue trabajando en la API completándola con nuevas funciones (Ad-hoc, información, modo bajo consumo).

Mes 4: Diciembre.

Semana 5 Diciembre.
Se termina la primera versión de la API.

Semana 12 Diciembre.
Se documenta la primera versión de la API, y el programa PC desarrollado.

Semana 19 Diciembre y 26 Diciembre.
Se realizan las pruebas de alcance y consumo

Mes 5: Enero.

Semana 2 Enero.
Se realizan las pruebas del módulo Wi-Fi con la API final.

Semana 9 Enero.
Se corrigen ciertos aspectos de la API gracias a los resultados de las pruebas, y se desarrolla la aplicación para iPhone final, basándonos en las ya desarrolladas..

Semana 16 Enero.
Se termina la aplicación *iPhone*, y se desarrolla la aplicación para *Android* final.

Semana 23 Enero.
Se termina la aplicación *Android* final y se suben a *App Store* y *Android Market*.

Semana 30 Enero.
Se realiza la documentación para la empresa del módulo Wi-Fi.

Mes 6: Febrero.

Semana 6 Febrero.
Se ultima la documentación del módulo Wi-Fi para la compañía y se empieza a realizar el vídeo promocional.

Semana 13 Febrero.
Se termina el vídeo promocional y se sube la información a la web de la compañía. Lanzamiento del producto: 15 Febrero.

Semana 20 Febrero.
Se empieza la realización de la memoria y los anexos. (trabajando en paralelo con otros aspectos como solucionar errores, temas de las aplicaciones)

Semana 27 Febrero.
Se sigue con la realización de la memoria y los anexos. (trabajando en paralelo con otros aspectos como solucionar errores, temas de las aplicaciones)

Mes 7: Marzo.

Semana 5 Marzo.
Se sigue con la realización de la memoria y los anexos. (trabajando en paralelo con otros aspectos como solucionar errores, temas de las aplicaciones)

Semana 12 Marzo.
Se termina la realización de la memoria y los anexos.

En la tabla F.1 se muestra la distribución de esfuerzos y en la figura F.1 se muestra esta distribución de forma gráfica.

De Septiembre a Diciembre, se trabajó 30 horas semanales, mientras que a partir de Enero se trabaja 40 horas semanales. Las horas abajo descritas son las horas reales en el trabajo del Proyecto, sin contar las horas dedicadas a otros aspectos en la empresa.

Mes	Horas
Septiembre	90 h
Octubre	114 h
Noviembre	90 h
Diciembre	60 h
Enero	90 h
Febrero	160 h
Marzo	60 h
Total	**664 horas**

Tabla F.1: Horas empleadas por mes.

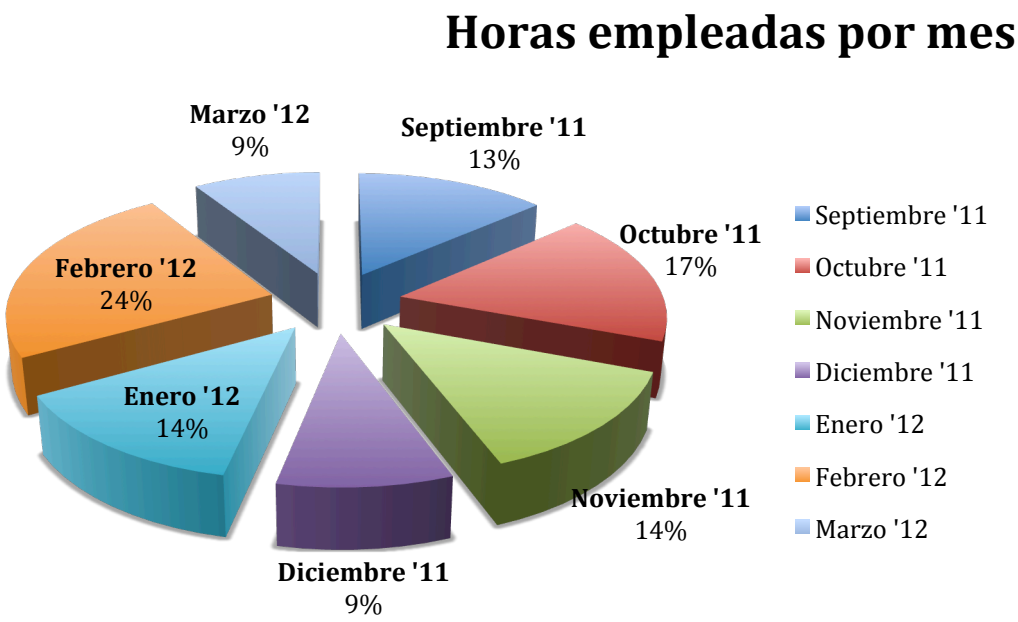

Figura F.1: Gráfico distribución de esfuerzos por meses.

F.2. Control de Esfuerzos.

Las tareas realizadas se han agrupado en categorías que se describen a continuación:

Formación: Horas invertidas en el aprendizaje del funcionamiento de los módulos *Waspmote*, el entorno *Waspmote IDE*. También entran aquí las horas dedicadas a comprender el funcionamiento de los integrados Wi-Fi, y sus comandos.

Reunión: Horas dedicadas a reuniones con el director de proyecto y con los compañeros de Libelium. Los temas de las reuniones han sido para ver hasta donde se llegaba con el API del módulo Wi-Fi (protocolos a implementar…), la elección del integrado final, y finalmente como debe ser el aspecto del software realizado.

Análisis y diseño: Horas dedicadas para analizar las necesidades reales tanto del API como del software realizado (aplicaciones, software PC…).

Implementación: Horas invertidas en la realización del API y del software asociado.

Pruebas: Horas dedicadas a la comprobación del correcto funcionamiento sobretodo del módulo Wi-Fi y de la API desarrollada, aunque también se realizaron pruebas del software desarrollado.

Corrección de Errores: Horas dedicadas a la corrección de los errores encontrados en cada una de las pruebas realizadas.

Documentación: Horas dedicadas a la realización de documentación tanto interna (manuales de usuario, guía técnica, vídeos), como de esta memoria.

A continuación se muestra en la tabla F.2 el control de esfuerzos por tareas, y en la figura F.2de forma gráfica.

Tarea	Horas
Formación	75 h
Reunión	15 h
Análisis y Diseño	90 h
Implementación	280 h
Pruebas	80 h
Corrección de errores	40 h
Documentación	84 h
Total	**664 horas**

Tabla F.2: Horas empleadas por tarea.

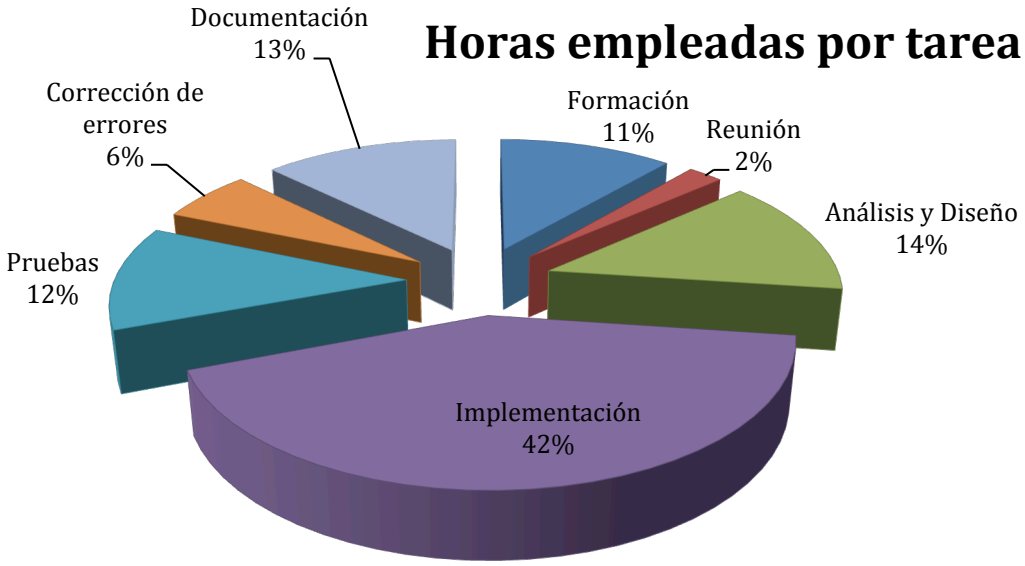

Figura F.2: Distribución horas empleadas por tarea.