Algoritmos de Encaminamiento en Gradiente para Redes de Sensores

Fernando Concepción Gutiérrez
Iñaki Goffard Giménez
Felipe Gutiérrez Lébedev

Profesor Director
Juan Carlos Fabero Jiménez

Proyecto de Sistemas Informáticos
Facultad de Informática
Universidad Complutense de Madrid
Curso 2009 - 2010

Índice general

Resumen

Hoy en día es frecuente el uso de sensores móviles que deben poder establecer una red para el envío de datos entre ellos de forma continua.

Para posibilitar esta comunicación es necesario un algoritmo que permita decidir el camino que debe seguir la información cuando no existe una comunicación directa entre el origen y el destino.

Se ha desarrollado un software para controlar la interfaz *Bluetooth* de *SHIMMER*, que es uno de los principales sensores inalámbricos que se podrían utilizar para el algoritmo. Esto ha sido probado en el simulador *MSPsim* después de una serie de adaptaciones que éste ha requerido para su correcto funcionamiento.

Por otra parte, se ha llevado a cabo el diseño y la implementación de dicho algoritmo de encaminamiento dinámico para sensores equipados con tecnología Bluetooth que permite que los nodos trabajen ordenadamente para recibir datos. Dichos datos deben viajar de un nodo origen a un nodo destino a través del camino más corto, es decir, aquel que suponga atravesar el menor número de intermediarios posibles. Para ello se ha definido un formato de trama adecuado y se han introducido detalles interesantes en la implementación, como mecanismos para poder dar servicio a más de un nodo a la vez.

Palabras clave: Bluetooth, encaminamiento, HDLC, gradiente, L2CAP, MSPsim, red de sensores, SHIMMER.

Nowadays it is common to use mobile sensors which must be capable of establishing a network for a seamlessly data sending between them.

In order to make this communication posible an algorithm is needed to decide the path the information will follow when there is no direct communication between the source and destination.

A software has been developped in order to control the *SHIMMER*'s *Bluetooth* interface which is one of the main wireless sensors that could be used for the algorithm. This has been tested in the *MSPsim* simulator after a series of adaptations required for a proper running.

On the other hand, the design and implementation of the said dynamic routing algorithm for Bluetooth technology equipped sensors have been carried out in order to allow the nodes to work tidily for data receiving. The data above must travel form an source node to a destination node through the shortest path, that means, the one ment to go through the lowest number of possible intermediaries. For this a suitable frame format has been defined and some interesting details have been introduced in the implementation as mechanisms for attending more than one station at a time.

Keywords: Bluetooth, routing, HDLC, gradient, L2CAP, MSPsim, network sensors, SHIMMER.

Convenciones Tipográficas

Convenciones usadas en este documento

Para proveer un texto consistente y fácil de leer, se ha decidido establecer y seguir una serie determinada de convenciones tipográficas.

Los siguientes tipos de formato en el texto indican información especial:

- **Negrita Especial**:

  * Nombres de capítulos, secciones y subsecciones.

  Ejemplo: Consultar el **capítulo 4**.

- *Cursiva*

  * Usada para enfatizar la importancia de un punto, para introducir un término o para designar un marcador de línea de comando, que debe reemplazarse por un nombre o valor real.

  Ejemplo: pila oficial de protocolos *BlueZ*

- `Monoespacio`

  * Nombres de comandos, rutas, archivos y directorios.

  Ejemplo: Por ejemplo, `tinyos-1.x/contrib/handhelds/tos/platform/shimmer/hardware.h`

- MAYÚSCULAS

  * Nombres de teclas.

  Ejemplo: `ENTER`

Capítulo 1

Introducción

1.1. Objetivos del Proyecto

El objetivo principal de este proyecto es diseñar e implementar un algoritmo de encaminamiento para redes de sensores. Este algoritmo ha de responder a las posibles necesidades de una red de sensores móviles formada por una serie de estaciones base y una serie de clientes que generan y reenvían datos a alguna de esas estaciones base. Por ejemplo, es interesante el uso de una red como esta en un centro hospitalario en el que están ingresados pacientes con problemas cardiacos. Los pacientes llevan un sensor consigo que mide la frecuencia cardiaca y cada cierto tiempo han de enviarla a la red de ordenadores del hospital, red que recoge los datos a través de estaciones base en los extremos de cada planta. Para que los datos generados por los sensores lleguen a las estaciones base, que muchas veces no están directamente a su alcance, es necesario que los sensores formen una red y que gracias al algoritmo de encaminamiento los datos viajen a través de los sensores de forma eficiente y rápida hasta llegar a alguna de las estaciones base.

Para ello tendremos a nuestra disposición herramientas relacionadas con los sensores SHIMMER, que tendremos que aprender a usar. Estas herramientas desarrolladas por la empresa SHIMMER ofrecen la posibilidad de desarrollar software libre cuyo funcionamiento se basa en el sistema operativo *TinyOS*. Al tratarse de programación para una red de sensores se trata de utilizar el lenguaje *nesC* especialmente adaptado para este tipo de hardware. Sin embargo debido a diversas restricciones será necesario programar los algoritmos bajo una distribución de Linux y utilizando el lenguaje *C*, ya que nesC está principalmente basado en C. El núcleo del hardware de SHIMMER es el microprocesador *MSP430* y existen varios simuladores de éste microprocesador como es el *Tmote Sky*. Lamentablemente éste no es del todo compatible con los programas compilados con SHIMMER debido a un mapeo diferente de puertos. Por ello, será necesario adaptar el simulador *MSPsim* compatible con SHIMMER.

En consecuencia será necesario, primero, estudiar la arquitectura de SHIMMER y adaptar el simulador para hacer algún ejemplo para familiarizarse con el entorno. Finalmente habrá que hacer un diseño completo del algoritmo y una implementación a bajo nivel que permita una adaptación sencilla a sensores de este tipo.

1.2. Restricciones iniciales

Una restricción muy importante es la de no tener a nuestra disposición el hardware de SHIMMER para el que haremos los primeros ejemplos. Esto nos obliga a utilizar un simulador basado en *Java* para redes de sensores controlados por procesadores de la familia MSP430: el MSPsim. Existe la dificultad añadida de la necesidad de simular una red de sensores para poder testear el algoritmo. Por otra parte, los simuladores con los que contamos no sirven para probar transmisiones de radio vía Bluetooth o ZigBee.

1.3. Requisitos del algoritmo de encaminamiento

En este punto se pretende *especificar* el comportamiento del sistema a diseñar. Por la última restricción observada en el punto anterior, nos vamos a centrar en un entorno de estaciones fomado por computadores con sistema operativo GNU/Linux y adaptadores Bluetooth. Esto es debido a la imposibilidad de contar directamente con el hardware de SHIMMER. Abordaremos la posibilidad de montar una red entre *computadores linux con bluetooth* de características deseables para nuestro propósito, antes de abordar cualquier posible implementación.

Por otra parte, implementaremos y probaremos en el simulador de SHIMMER, un software que permita leer y escribir a través del puerto serie *USART1*, pues es la interfaz que existe entre el sensor y su chip Bluetooth. A través de este puerto podemos configurar el dispositivo, y envíar y recibir datos. Consultar el **capítulo 4** para ver dicho programa. Está desarrollado en C, y compilado con el compilador cruzado *MSP430GCC*. Este software será muy útil para iniciar la adaptación del algoritmo a los sensores reales de hardware.

1.3.1. Funcionalidades obligatorias

El algoritmo de encaminamiento debe cumplir algunas características:

- Debe estar diseñado de forma que un sensor cualquiera perteneciente a una red pueda enviar datos a otro sensor cercano y perteneciente a la misma red de forma que estos datos lleguen finalmente a una estación base y ésta los procese adecuadamente.
- Debe definir la formación de un camino rápido y óptimo de cualquier sensor de la red al servidor cada vez que se transmiten datos.
- Se debe diseñar un formato de trama adecuado al tipo de comunicación que se va a establecer.
- Las estaciones bases deben recibir la información y ser capaces de interpretarlas.
- El algoritmo debe ser implementado a bajo nivel de forma que sea fácilmente adaptable para plataformas de sensores.

1.3.2. Escenarios de Calidad

Nuestro algoritmo debe cumplir con unos requisitos marcados por la naturaleza del entorno en que va a funcionar. Los escenarios de calidad son situaciones que permiten identificar qué módulos hay que implementar y qué decisiones tomar para dar soporte a estos requisitos. Volvemos a recordar aquí que el objetivo final es poder hacer que datos generados por algún sensor de la red, atravesando el menor número de enlaces posible, lleguen a la estación base. Ésta es un dispositivo con mayores recursos de capacidad de cómputo, almacenamiento, interacción con usuarios, etc... Esta estación es fija, y será identificada con el número más bajo de la red (por ejemplo 1). Para más información ver el **apartado 5.2**. Por tanto, de ahora en adelante, por *estación más cercana*, nos referimos a aquella que esté dentro del radio de alcance de un dispositivo, y que tenga el número más bajo. Nuestro algoritmo va a funcionar en un entorno que impone los siguientes requisitos:

- Movilidad: Se trata de una red de sensores inalámbricos (aunque simulemos su comportamiento a través de computadores de propósito general), que pueden ser móviles.
- Adaptabilidad: Por restricciones de diseño, debemos pensar en que nuestros algoritmos sean fácilmente ejecutables en diversas plataformas.
- Disponibilidad: El algoritmo debe ser lo más robusto posible, para garantizar la comunicación de paquetes en la red.

- Fiablilidad: Si en algún momento fuese imposible encontrar un camino hasta la estación base, la estación con el número más bajo debería implementar un sistema que permitiera almacenar de manera temporal cierta cantidad de datos, a la espera de una nueva posibilidad de hacer que éstos lleguen a la estación base.
- Versatilidad: El algoritmo debe permitir conmutar con rapidez entre las distintas conexiones que se implementen para soportar la red.

A continuación pasamos a ver los escenarios para cada uno de estos requisitos:

Movilidad

Pueden darse tres escenarios distintos, para las estaciones (sensores o computadores) que compongan la red. Estos escenarios plantean un modelo cliente - servidor , ya que se trata de comunicaciones en red.

1. Estación que genera y/o reenvía datos: por un lado debe trabajar como un cliente a la hora de conectarse con la estación más cercana y retransmitir los datos que otra estación le entregue. Esa otra estación le habrá seleccionado a su vez como estación más cercana. Por otro lado, deberá actuar como servidor para estar a la escucha de las nuevas tramas que le lleguen, y que tendrá que reenviar siempre que sea posible. Por último, esta estación puede ser ella misma fuente de nuevos datos, por lo que deberá proporcionarse un reparto de tiempo entre una función que le permita actuar como puente, y otra que le haga funcionar como cliente. A esta estación le llamaremos *intermediario* en nuestra implementación.
2. Estación receptora de datos: en este caso se presenta la situación propia del dispositivo que actúa como estación base. Se deberá implementar un servidor que esté a la escucha y acepte conexiones para recibir datos. En este caso le llamaremos *servidor*.

Adaptabilidad

Aquí se plantea el escenario de un *cambio en la plataforma de ejecución*. Este escenario viene provocado por un cambio en el hardware que se tenga disponible: ya se dispone de sensores SHIMMER, y vamos a ejecutar nuestro algoritmo en una red formada por éstos. Este escenario nos plantea la decisión de desarrollar el algoritmo en el lenguaje C, ya que podemos compilar directamente para la plataforma SHIMMER utilizando un compilador cruzado. Habrá que realizar sin embargo todas aquellas modificaciones que vengan impuestas por un posible cambio en la interfaz entre el *host* y su dispositivo Bluetooth, como se explica al principio del **apartado 5.3**

Disponibilidad

El escenario que aquí se plantea, es el de la *desaparición o aparición de estaciones*. Como solución a este escenario, se implementará un mecanismo que permita conocer las direcciones MAC de más de una estación objeto de destino de una comunicación directa, o bien de una retransmisión. Así, si la estación que un cliente está utilizando como puente desaparece, intentará continuar con otra de las de su lista de alcance. Será preciso implementar también, como parte de la solución, un mecanismo de exploración periódica que mantenga información actualizada de qué estaciones están al alcance. Con estas dos respuestas se pretende cuidar que el algoritmo pueda estar efectuando trabajo útil el mayor tiempo posible.

Fiabilidad

En este caso, ante la posibilidad de encontrarnos con la situación de no poder entregar los datos a la estación base, en cada dispositivo se deberá implementar una función que permita guardar cierta cantidad de datos, pues cualquiera es susceptible de ser la última estación a la que la que el algoritmo es capaz de llegar. En un computador Linux, no habría problema en utilizar el disco duro. En el caso de sensores SHIMMER, éstos poseen una tarjeta de memoria flash *microSD* (ver **apartado 2.1**) que permitiría esta solución.

Capítulo 2

Arquitectura del Sistema

2.1. Sobre la Tecnología de SHIMMER

Las siglas S. H. I. M. M. E. R. se corresponden con *Sensing Health with Intelligence, Modularity, Mobility and Experimental Reusability*, es decir, Sondeando la Salud con Inteligencia, Modularidad, Movilidad y Reutilización Experimental.

SHIMMER es una pequeña plataforma inalámbrica que puede grabar y transmitir datos fisiológicos y cinemáticos en tiempo real. Esta plataforma supera las restricciones de tamaño, peso y consumo de los sistemas de comunicación inalámbrica actuales. Diseñado como un sensor portátil, SHIMMER cuenta con un determinado número de sensores que permiten capturar un amplio rango de eventos: Se puede observar una foto de SHIMMER en la **figura 2.1**.

- *Acelerómetro*: El acelerómetro de 3 ejes mide la aceleración experimentada relativa a la caída libre. Es capaz de detectar la magnitud y la dirección de la aceleración de forma vectorial y puede usarse para la orientación sensorial, las vibraciones y los impactos.
- *Giróscopo*: El giróscopo de doble eje mide la orientación basándose en el principio de la conservación del momento angular.
- *ECG (Electrocardiograma)*: Interpreta la actividad eléctrica del corazón capturada durante un tiempo determinado y registrada externamente mediante electrodos sobre la piel.
- *EMG (Electromiograma)*: Evalúa y registra la señal de activación de los músculos.
- *Sensor IRP (Infrarrojo Pasivo)*: Se trata de un sensor de detección de movimiento.
- Sensor de Inclinación y Vibración.

Este desarrollo ha permitido investigaciones y análisis en un amplio rango de áreas incluyendo análisis de movimiento, sensaciones cinemáticas, detección de caídas, análisis de la marcha y actividades de la vida cotidiana. Existe también un conector externo que permite la conexión de sensores diseñados aparte, dando a entender que las posibilidades de detección inalámbrica y monitorización son ilimitadas. Con la creciente demanda en servicios sanitarios así como en calidad, eficiencia y soporte a lo largo de todas las industrias, los desarrolladores deben encontrar medidas innovadoras y económicamente razonables para este tipo de servicios. Cada vez es mayor el enfoque hacía Redes de Sensores Inalámbricos (RSI) como patrón para soluciones de apuntalamiento inteligente. Esto ha sido evidente en la industria sanitaria como bases para el desarrollo de tecnologías que permitan la vida asistida, la gestión de las enfermedades, la monitorización y las soluciones sanitarias remotas. Todo ello viene a contribuir a la calidad del cuidado, los servicios y a la reducción de los costes y los problemas de organización del sistema sanitario. Dicho enfoque se ha visto a lo largo de todas las

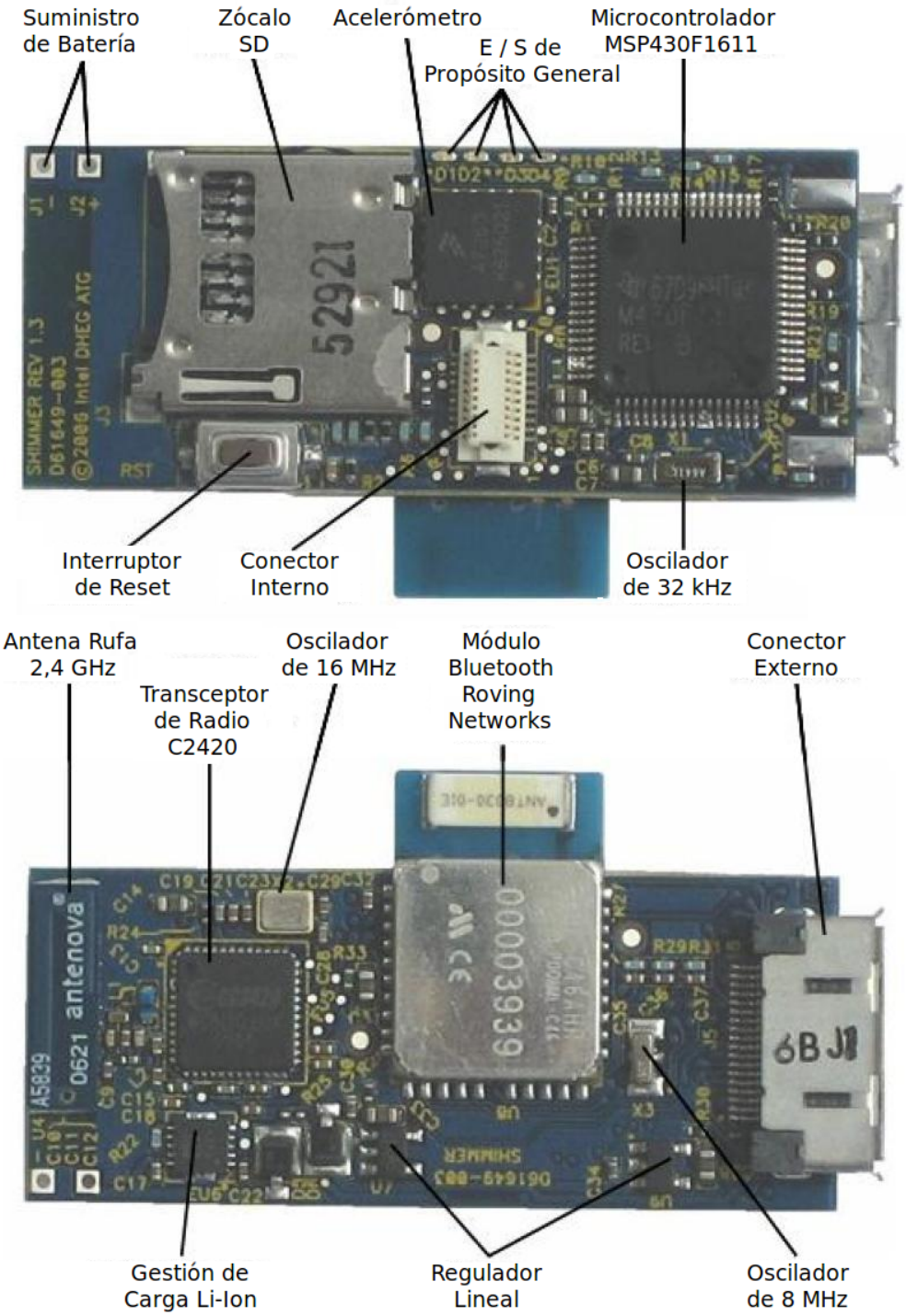


Figura 2.1: SHIMMER.

industrias donde se están utilizando las RSI para la monitorización y la mejora de la producción, la electrónica de consumo y el rendimiento deportivo. SHIMMER ofrece una plataforma de desarrollo extremadamente amplia que permite a los investigadores superar la mayoría de las barreras debidas al hardware y centrarse en el valioso análisis y la interpretación de los datos obtenidos. Con un rápido giro hacía las RSI para ayudar a la monitorización, el abordaje y la interacción con nuestro entorno y el deseo de integrar dispositivos de sondeo constante en los productos existentes, SHIMMER suministra una plataforma líder en estas tecnologías. SHIMMER provee de una solución real para el desarrollo de capacidades de sondeo inalámbrico entre un rango de sectores incluyendo la sanidad, la defensa, la producción y la monitorización ambiental. SHIMMER fue diseñada originalmente por Intel en el año 2006 como una plataforma para futuras invetigaciones en la vida asistida y las soluciones sanitarias remotas. Desde octubre de 2009 SHIMMER dispone de la revisión 2.0 que ofrece una gran funcionalidad y mayores opciones que la original. Actualmente la división de investigación de SHIMMER (SHIMMER Research), fundada en enero de 2008 tiene clientes en más de 28 países en todo el mundo mientras la plataforma de sensores continua creciendo y desarrollándose.

2.1.1. Características

- La Plataforma de Sensores Inalámbricos de SHIMMER: los sensores inalámbricos han ido progresando recientemente gracias al desarrollo de la tecnología micro electromecánica (*MEMS*). Sin embargo, las tradicionales restricciones como el tamaño, el peso una comunicación inalámbrica fiable y el consumo energético son problemas a tener en cuenta. SHIMMER ofrece una plataforma extensible que permite sobrepasar las tradicionales limitaciones de hardware y software mediante las siguientes características:
  * Muy bajo consumo de energía
  * Fácil encendido
  * Tamaño reducido: 50 mm x 25 mm x 12,5 mm
  * Peso reducido: 15 gramos
  * Conexión instantánea mediante Bluetooth o radio 802.15.4
- Características de la Plataforma:
  * Factor de forma compacto, ligero y portátil (Peso: 15 gramos, Volumen: 50 mm x 25 mm x 12,5 mm)
  * Comunicación inalámbrica via Bluetooth y 802.15.4 (WML-C46A, CC2420)
  * Captura de datos sin conexión Almacenamiento mediante tarjetas MicroSD 2 Gigabytes
  * *CPU* MSP430 (10 Kbytes de RAM, 48 Kbytes de memoria *Flash*, 8 canales de 12 bits A/D) a 8 MHz con punteo de datos por SD
  * Plataforma libre, basada en TinyOS
  * Conectores de expansión internos y externos
  * Incluye una simple interfaz serie de comandos para Bluetooth
  * Pila para *TCP/IP* integrada para 802.15.4
  * Acelerómetro MEMS de 3 ejes integrado con rango seleccionable
  * Sensor de inclinación y vibración intergado
  * Gestión integrada de la *Batería Li-ion*
  * Compatible con la plataforma software gráfica *BioMOBIUS*
- Accesorios y Placas de Expansión Disponibles:
  * Sondeo cinemático - Diseñado para capturas de movimiento. El diseño usa un par de giróscopos de doble eje.

* Electrocardiógrafo microalimentado de 3 vías (ECG) Usado para la captura de datos ECG

* Electromiógrafo microalimentado de 2 vías (EMG) Usado para la evaluación y el registro de la señal de activación de los músculos.

* Sensor Pasivo de Infrarrojos

* Base de lectura y programación USB Para programar y aplicaciones donde es preferible la comunicación serie por cable. También actúa como un cargador para SHIMMER.

* Base de carga para SHIMMER Cargador de batería para 6 SHIMMERs a la vez

* Placa de interrupciones analógicas para un rápido prototipado. Permite la conexión a SHIMMER de múltiples señales analógicas.

2.1.2. Resumen de capacidades:

- *E/S* (Entrada / Salida):

  * Propósito: Capturar un sensor y los datos del usuario

  * Componentes integrados:

  - Acelerómetro de 3 ejes usando acelerómetros MEMS de escala libre microalimentados MMA7260Q 1.5.2.4.6g dentro del *módulo A/D* de la CPU.

  - 3 *LED*s de color de estado

  - Interruptor de encendido/apagado programable por el usuario o mediante reseteo

  - Sensor de inlinación/vibración una opción de sondeo de mínimo consumo que también puede usarse como una aplicación de entrada para conmutar entre transiciones de estados de consumo

- Procesamiento:

  * Propósito: Controlar el estado operativo, suministrar la mejor calidad de la señal, alertas operacionales y mensajes

  * Componentes: MSP430

  - 10 Kbytes de RAM, 48 Kbytes de memoria Flash

  - Hasta 8 MHz

  - 8 canales de 12 bits A/D

  - Consumo extremadamente bajo durante los periodos de inactividad

  - Solución probada en aplicaciones de sondeo médico

- Almacenamiento:

  * Propósito: Impedir las perdidas de datos mientras esté en movimiento, durante cortes de red o mientras se cambien las baterías

  * Componentes: Ranura MicroSD

  - Hasta 2 Gbytes disponibles actualmente

  - Tarjeta SD con ruta de datos *MUX* para una velocidad máxima de transferencia mientras esté fijada sobre la base y control de consumo suave con detección de desbordamiento de la base

  - Memoria Flash basada en tecnología *NAND*, con un consumo de 20 mA en modo lectura/escritura.

- Comunicaciones:
  * Propósito: Gran fiabilidad, movilidad estandarizada
  * Componentes:

  Radio 802.15.4:

  - Chipcon CC2420
  - Antena GigaAnt Rufa de 4,1 dBi

  Radio Bluetooth de Clase 2:

  - Diseño basado en el módulo Mitsumi WML-C46N CSR

- Factor de Forma:
  * Propósito: Portabilidad
  * Componentes:

  - El volumen mínimo de los sensores dentro del encastre es: 50 mm x 25 mm x 12,5 mm y 10 gramos de peso sin el módulo Bluetooth. Comparable a una barra de labios.

  - El encastre inicial es duradero y del tamaño de un mechero Zippo y tiene posibilidades para placas de expansión y Bluetooth. El encastre puede montarse en un brazalete de un reproductor MP3.

- Vida Operativa y Consumo:
  * Propósito: Larga vida operativa, fácil mantenimiento y despliegue
  * Componentes:

  - La duración operativa prevista por el diseño es de 10 días mientras se realiza un muestreo de 6 canales a 50 Hz con una batería de 250 mAH.

  - La especificación de la batería en estado desconectado y guardado es superior a un año

  - Cargador de batería de Li-ion integrado

  - Posibilidad de monitorizar e indicar el estado de la alimentación

  - Características de control de consumo incluyendo un botón de consumo bajo y un apagado bajo batería baja

2.1.3. Software

El entorno operativo de TinyOS es altamente recomendable para el diseño, la implementación, las pruebas y la validación del software empotrado de SHIMMER (*firmware*). TinyOS ofrece ahorro debido a las extensas librerías de plataforma entrecruzada en código abierto. Reutilizar las aplicaciones en SHIMMER es una ventaja clave del entorno de TinyOS y acelera significativamente el desarrollo del software y el proceso de validación. El código de la plataforma de SHIMMER es mantenido activamente. SHIMMER es una plataforma oficial en TinyOS-2.x y puede utlizarse bajo TinyOS 1.x. Las funcionalidades actuales incluyen:

- Almacenamiento de memoria Flash MicroSD
- Sistema de archivos *FAT*
- Pila de *IP* para 802.15.4
- Configuración Bluetooth, gestión de la conexión y el flujo de las transferencias de datos
- Módulo de reloj en tiempo real
- Control y configuración de periféricos
- Monitorización del consumo

2.1.4. Hardware

La **figura 2.2** ilustra el diagrama de bloques de la placa de SHIMMER y el interconexionado entre los dispositivos integrados. El elemento central de la plataforma es el microprocesador de bajo consumo MSP430 que controla la operatividad del dispositivo. La CPU se comunica con los distintos periféricos a través de modulos de expansión interna/externa. Captura el sondeo de datos desde el conversor analógico-digital (*ADC*) de 8 canales. Casi cualquier característica de la CPU existe en la implementación de SHIMMER. La CPU configura y controla varios periféricos integrados mediante pines de E/S, algunos de los cuales están disponibles en los conectores de expansión interna/externa. La CPU tiene un conversor analógico-digital (ADC) de 12 bits y 8 canales que se usa para capturar el sondeo de datos desde el acelerómetro y las expansiones de los sensores, como el ECG, la cinemática, el *GSR* o el EMG. La expansión externa permite la comunicación desde y hacia la placa usando la estación de acoplamiento. La placa de SHIMMER tiene un socket de memoria flash MicroSD integrado para un almacenamiento adicional así como tres diodos emisores de luz (LED) para mostrar información. Para el flujo de datos inalámbrico, la plataforma está equipada con un módulo Bluetooth y otro de radio 802.15.4.

2.2. Comunicación Inalámbrica de SHIMMER

2.2.1. 802.15.1 Bluetooth Radio

Antes de hablar de un posible algoritmo para establecer una red de sensores, es conveniente conocer un poco las peculiaridades de la tecnología Bluetooth, ya que nuestro algoritmo se basará en dicha tecnología.

Principales características de la tecnología Bluetooth

La tecnología Bluetooth está basada en la comunicación mediante radiofrecuencia. Hay tres tipos de clases de dispositivos Bluetooth en función de su potencia máxima y por tanto el alcance máximo del dispositivo, en nuestro caso y como en la mayoría de dispositivos del mercado, el Bluetooth de SHIMMER es de clase 2 y tiene un alcance aproximado de 25 metros.

Cómo establecer una conexión

Por defecto, las conexiones no requieren autentificación, y si la requieren, la contraseña por defecto suele ser 1234. Una vez establecida la conexión, estaremos por defecto en el modo data mode, para introducir comandos deberemos pasar al modo command mode introduciendo tres símbolos del dólar, de esta forma podremos introducir comandos durante 60 segundos u otro tiempo configurable.

Modos de operación

- Slave mode (SM, 0): Modo por defecto. Permite ser descubierto por otros dispositivos y aceptar conexiones. También pueden realizarse conexiones de salida.
- Master mode (SM, 1): Este modo se utiliza para iniciar conexiones y no recibirlas. Este modo no permite ser descubierto ni aceptar conexiones.
- Trigger Master mode (SM, 2): En este modo, el dispositivo se conectará automáticamente con un dispositivo exclavo preconfigurado cuando un carácter sea recibido por el UART local. La conexión estará abiera hasta que hayan pasado entre 1 y 255 segundos de inactividad (configurable).
- Auto-Connect (Master mode) (SM, 3): Este modo se configura por comando o por encendido. El dispositivo conectará con otro dispositivo cuya dirección esté prealmacenada, y si no hay nada prealmacenado almacenará la dirección del primero que encuentre cuyo COD coincida. En este modo,la conexión no

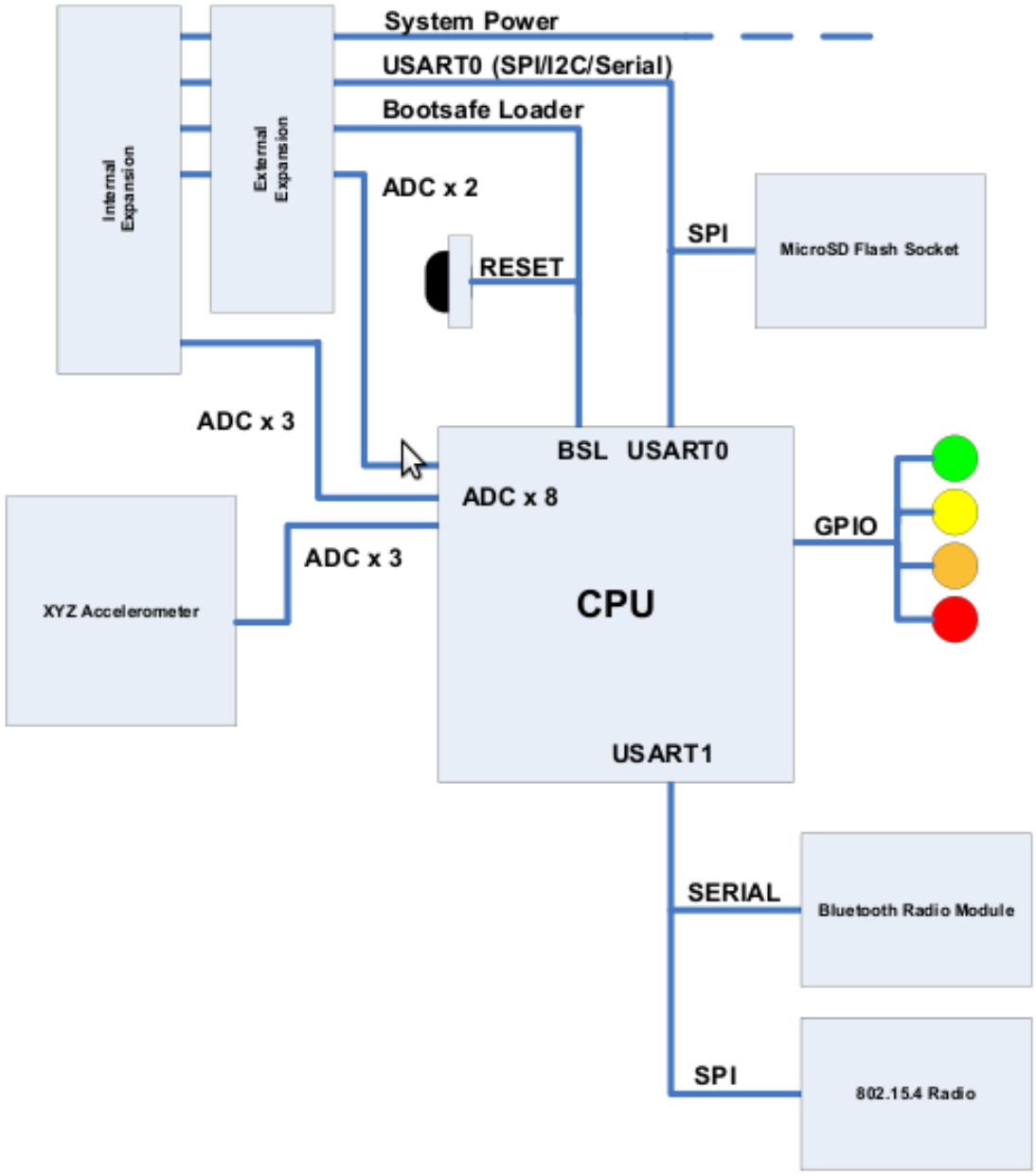


Figura 2.2: Estructura hardware de SHIMMER.

puede romperse por comandos y si el dispositivo se desconecta intentará conectarse de nuevo hasta que tenga éxito.

- Auto-Connect (DRT mode) (SM, 4): Este modo se debe configurar por comando. Este modo opera igual que el (SM, 3), a diferencia que la conexión y desconexión puede ser configurable con un interruptor externo.
- Auto-Connect (ANY mode) (SM, 5): Este modo se debe configurar por comando. Este modo opera igual que el (SM, 4), a diferencia que cada vez que el interruptor PIO es puesto a ON, se hace una búsqueda y el primer dispositivo que se encuentra es el elegido para conectarse. No se usan direcciones almacenadas.

Posibles configuraciones

- COMMAND MODE vs DATA MODE: Al encender el dispositivo, se establecerá el modo data mode. Para cambiar al modo command mode es necesario enviar `"$$$"` por el puerto serie o a través de una conexión Bluetooth remota, el dispositivo responderá `"CMD"`. Para salir del modo `"command mode"` es necesaria la orden `"---<cr>"` y entonces el dispositivo responderá `"END"`. Los comandos son aceptados en modo ASCII. Si el dispositivo está en modo Master Mode, no se puede establecer el modo command mode a través de una conexión Bluetooth remota.
- Configuración local: La configuración del puerto serie ha de ser:

  - 115,200 bps (baudios por segundo)

  - 8 bits

  - No Parity (Sin Paridad)

  - 1 stop bit

  - Hardware flow control enabled (Control de flujo por hardware activado) Cuando el dispositivo esté en el modo command mode, devolverá `"AOK"` para comandos correctos y `"ERR"` para comandos incorrectos. Devolverá `"?"` para comandos no reconocidos.
- Configuración local (via Bluetooth): Se debe conectar desde un Bluetooth remoto y realizar las mismas acciones que para la configuración local. Cuando se haya terminado de configurar, se envía el comando `"---"` para salir del modo de configuración y permitir el envío normal de datos.

Para más información relativa al hardware de Bluetooth de SHIMMER, consultar [INC].

2.2.2. 802.15.4 ZigBee Radio

Dado que el algoritmo que se desarrolla en este proyecto está basado en la tecnología Bluetooth, en este apartado solo se van a comentar brevemente lo que es la tecnología ZigBee y las diferencias que tiene con respecto a la tecnología Bluetooth.

ZigBee es el nombre de la especificación de un conjunto de protocolos de alto nivel de comunicación inalámbrica para su utilización con radio digital de bajo consumo, basada en el estándar IEEE 802.15.4 de *WPAN* (wireless personal area network). Su objetivo son las aplicaciones que requieren comunicaciones seguras con baja tasa de envío de datos y maximización de la vida útil de sus baterías. En principio, el ámbito donde se prevé que esta tecnología cobre más fuerza es en domótica debido a diversas características que lo diferencian de otras tecnologías:

- Su bajo consumo
- Su topología de red en malla
- Su fácil integración (se pueden fabricar nodos con muy poca electrónica).

La relación entre IEEE 802.15.4-2003 y ZigBee es parecida a la existente entre IEEE 802.11 y Wi-Fi Alliance. La especificación 1.0 de ZigBee se aprobó el 14 de diciembre de 2004 y está disponible a miembros del grupo de desarrollo (ZigBee Alliance). Un primer nivel de suscripción, denominado adopter, permite la creación de productos para su comercialización adoptando la especificación por 3500 dólares anuales. Esta especificación está disponible al público para fines no comerciales en la petición de descarga. La revisión actual de 2006 se aprobó en diciembre de dicho año. ZigBee utiliza la banda ISM para usos industriales, científicos y médicos; en concreto, 868 MHz en Europa, 915 en Estados Unidos y 2,4 GHz en todo el mundo. Sin embargo, a la hora de diseñar dispositivos, las empresas optarán prácticamente siempre por la banda de 2,4 GHz, por ser libre en todo el mundo. El desarrollo de la tecnología se centra en la sencillez y el bajo coste más que otras redes inalámbricas semejantes de la familia WPAN, como por ejemplo Bluetooth. El nodo ZigBee más completo requiere en teoría cerca del 10% del hardware de un nodo Bluetooth o Wi-Fi típico; esta cifra baja al 2% para los nodos más sencillos. No obstante, el tamaño del código en sí es bastante mayor y se acerca al 50% del tamaño del de Bluetooth. Se anuncian dispositivos con hasta 128 kB de almacenamiento. En 2006 el precio de mercado de un transceptor compatible con ZigBee se acerca al dólar y el precio de un conjunto de radio, procesador y memoria ronda los tres dólares. En comparación, Bluetooth tenía en sus inicios (en 1998, antes de su lanzamiento) un coste previsto de 4-6 dólares en grandes volúmenes; a principios de 2007, el precio de dispositivos de consumo comunes era de unos tres dólares. La primera versión de la pila suele denominarse ahora ZigBee 2004. La segunda versión y actual a junio de 2006 se denomina ZigBee 2006, y reemplaza la estructura *MSG / KVP* con una librería de clusters, dejando obsoleta a la anterior versión. ZigBee Alliance ha comenzado a trabajar en la versión de 2007 de la pila para adecuarse a la última versión de la especificación, en concreto centrándose en optimizar funcionalidades de nivel de red (como agregación de datos). También se incluyen algunos perfiles de aplicación nuevos, como lectura automática,automatización de edificios comerciales y automatización de hogares en base al principio de uso de la librería de clusters. En ocasiones ZigBee 2007 se denomina Pro, pero Pro es en realidad un perfil de pila que define ciertas características sobre la misma. El nivel de red de ZigBee 2007 no es compatible con el de ZigBee 2004-2006, aunque un *nodo RFD* puede unirse a una red 2007 y viceversa. No pueden combinarse routers de las versiones antiguas con un coordinador 2007. Los protocolos ZigBee están definidos para su uso en aplicaciones embebidas con requerimientos muy bajos de transmisión de datos y consumo energético. Se pretende su uso en aplicaciones de propósito general con características autoorganizativas y bajo coste (redes en malla, en concreto). Puede utilizarse para realizar control industrial, albergar sensores empotrados, recolectar datos médicos, ejercer labores de detección de humo o intrusos o domótica. La red en su conjunto utilizará una cantidad muy pequeña de energía de forma que cada dispositivo individual pueda tener una autonomía de hasta 5 años antes de necesitar un recambio en su sistema de alimentación.

Para más información relativa al hardware de Bluetooth de SHIMMER, consultar [All].

Comparativa entre ZigBee y Bluetooth

ZigBee es una tecnlogía muy similar al Bluetooth pero la transmisión es más lenta y por tanto es más apropiada para *Domótica* y productos dependientes de batería cuyas transferencias no son grandes. Bluetooth tiene una velocidad de transmisión mayor, aunque también el consumo; por tanto se usa para informátia casera y teléfonos.

Capítulo 3

Simuladores

3.1. Introducción

Con el sistema operativo de TinyOS se pueden hacer pruebas de diseño e implementación para SHIMMER. Existen varios simuladores del MSP430, uno de ellos es el Tmote Sky (ver **figura 3.1**), que tiene las siguientes ventanas:

- Monitor de pila.
- Monitor del ciclo de trabajo: mide el uso de la CPU, el trabajo de la radio de transmisión y recepción y el consumo de los LEDs.
- USART Port Output: utiliza el dispositivo periférico para transmitir y recibir de forma síncroma.
- Panel de control de la simulación: permite visualizar el código, depurar, ejecutar paso a paso, visualizar pila de ejecución y mostrar estadísticas.
- Imagen del nodo.
- Consola de comandos: para la realización de tareas como resetear la CPU.

3.2. Adaptación de un primer ejemplo para Tmote Sky

Lamentablemente Tmote Sky no es del todo compatible con los programas compilados con SHIMMER debido a un mapeo diferente de puertos, ya que los sensores no todos contienen el mismo hardware. Una prueba que hemos hecho es compilar un primer ejemplo relacionado con el funcionamiento de los LEDs (archivo `Blink.nc` que viene en el propio manual de SHIMMER [SHI08]). Esta programado en nesC y las facilidades proporcionadas por TinyOS para la plataforma SHIMMER. Este ejemplo no funciona correctamente en Tmote Sky, sin embargo al investigar el mapeo de los LEDs que usa Tmote Sky pudimos observar que si cambiábamos algunas direcciones del mapeo de puertos de los LEDs en un fichero interno de TinyOS para la plataforma SHIMMER e introducíamos un valor adecuado al mapeo de Tmote Sky y a su frecuencia, entonces el programa sí hacía su función, en este caso el parpadeo constante de los LEDs. Dicho fichero se encuentra en la siguiente ruta:

`tinyos-1.x/contrib/handhelds/tos/platform/shimmer/hardware.h`

Los cambios en el mapeo de puertos relativos a los LEDs han sido los siguientes y se han detectado a partir del esquemático de la **figura 3.2**:

Puertos antiguos pertenecientes al mapeo del SHIMMER:

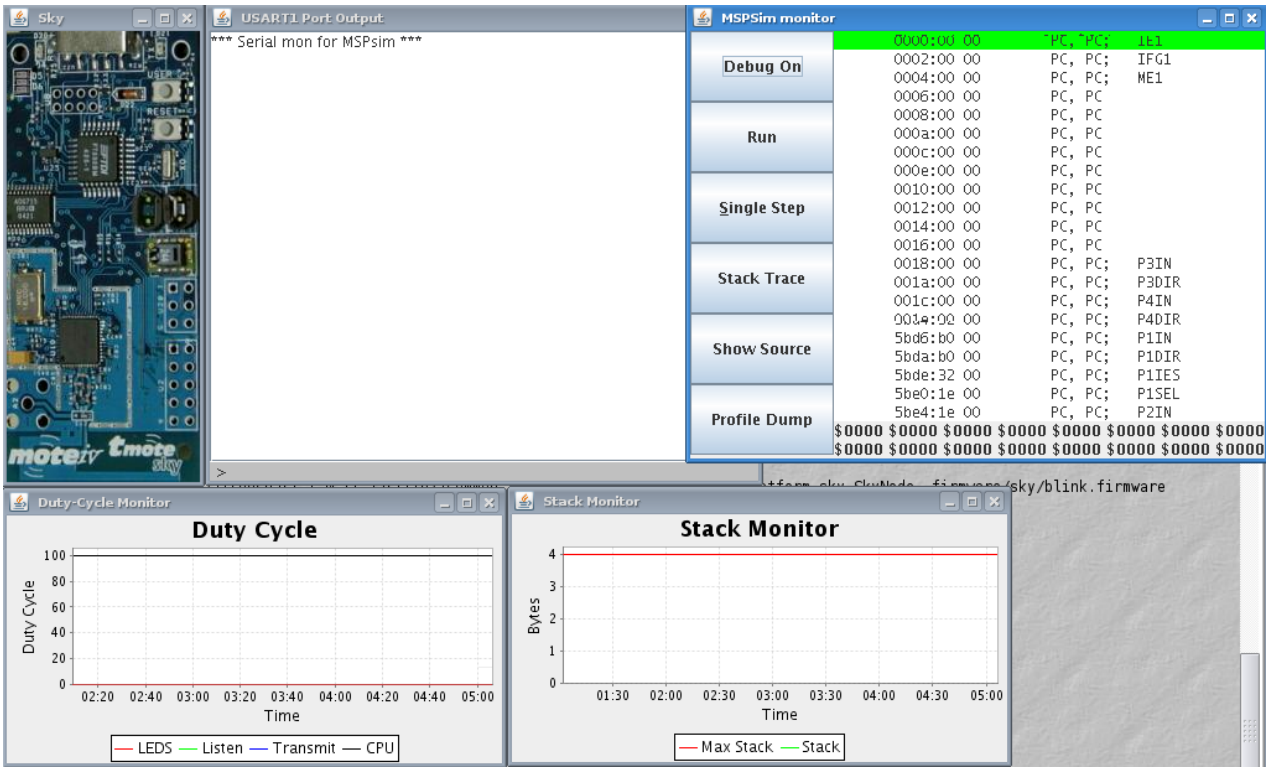


Figura 3.1: Vista del simulador Tmote Sky.

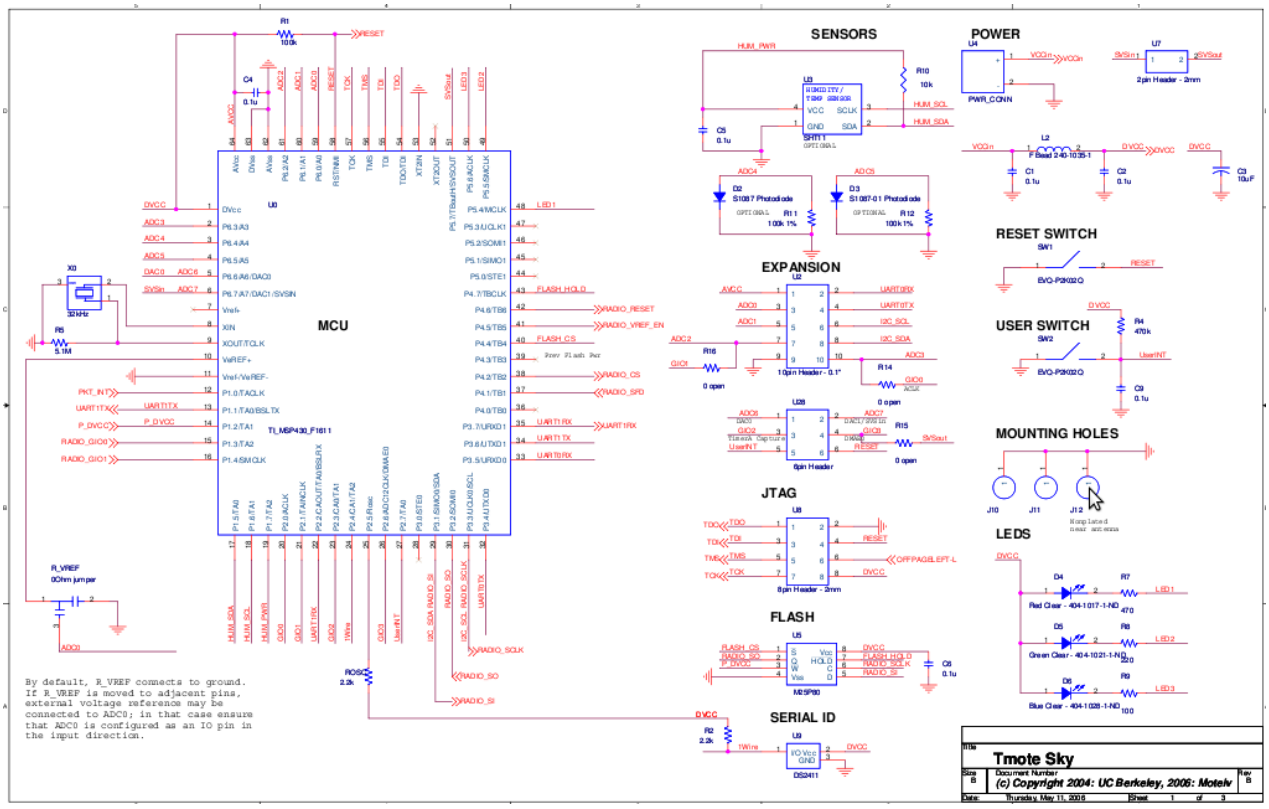


Figura 3.2: Mapeo de puertos de Tmote Sky.

```
TOSH_ASSIGN_PIN(RED_LED,    4, 0);
TOSH_ASSIGN_PIN(ORANGE_LED, 4, 1);
TOSH_ASSIGN_PIN(YELLOW_LED, 4, 2);
TOSH_ASSIGN_PIN(GREEN_LED,  4, 3);
```

Modificaciones realizadas:

```
TOSH_ASSIGN_PIN(RED_LED, 5, 4);
TOSH_ASSIGN_PIN(GREEN_LED, 5, 5);
TOSH_ASSIGN_PIN(YELLOW_LED, 5, 6);
```

3.3. Simulador MSPsim

El curso anterior se desarrolló un simulador compatible con SHIMMER [AGMNPG09] llamado MSPsim con el que empezamos a contar más tarde. Después de unas cuantas adaptaciones ha permitido realizar algunas aplicaciones para probar la comunicación a través del puerto serie del simulador.

Este simulador tiene incorporado un monitor de potencia y genera un fichero log con el consumo de la radio y la CPU en cada ejecución. En la **figura 3.3** puede observarse el simulador, trabajando con la plataforma SHIMMER.

Después de las siguientes adaptaciones hemos podido probar el ejemplo del apartado anterior y hacer unos primeros programas descritos en el capítulo siguiente. Dichas adaptaciones se efectuarán en el simulador en el archivo siguiente:

```
msp-sim/se/sics/mspsim/platform/shimmer/MoteIVNode.java
```

Las adaptaciones han sido:

- Mapeo de los puertos del ZigBee: la implementación que incorporaba el simulador inicialmente no correspondía con el hardware del SHIMMER.

  Para modificar el mapeo de los puertos conforme a la implementación del SHIMMER se han realizado los siguientes cambios:

  Asignación de puertos antiguos de la ZigBee:

  ```
  /*P1.0 - Input: FIFOP from CC2420 */
  /*P1.3 - Input: FIFO from CC2420 */
  /*P1.4 - Input: CCA from CC2420 */
  public static final int CC2420_FIFOP = 0;
  public static final int CC2420_FIFO = 3;
  public static final int CC2420_CCA = 4;
  ```

  Asignación correcta:

  ```
  /* P1.0 - Input: FIFO from CC2420 */
  /* P1.2 - Input: SFD from CC2420 (with IRQ) */
  /* P2.6 - Input: FIFOP from CC2420 */
  /* P2.7 - Input: CCA from CC2420 */
  public static final int CC2420_FIFO  = 0;
  public static final int CC2420_SFD   = 2;
  public static final int CC2420_FIFOP = 6;
  public static final int CC2420_CCA   = 7;
  ```

- Incorporación de los puertos relativos al Bluetooth que no estaban presentes en la versión del simulador inicial:

```
/* Port 1 */
public static final int BT_PIO = 5;
public static final int BT_RTS = 6;
public static final int BT_CTS = 7;
/* Port 3 */
public static final int BT_TXD = 6;
public static final int BT_RXD = 7;
/* Port 5 */
public static final int BT_RST = 5;
```

- Fijar los cambios para los colores de los LEDs tal y como deben aparecer en el SHIMMER:

  Asignación errónea de los colores de los LEDs:

```
public static final int BLUE_LED = 0x00;
public static final int GREEN_LED = 0x30;
public static final int RED_LED = 0x20;
public static final int ORANGE_LED = 0x10;
```

  Asignación correcta:

```
public static final int GREEN_LED = 0x08;
public static final int BLUE_LED = 0x04;
public static final int RED_LED = 0x02;
public static final int ORANGE_LED = 0x01;
```

- Por último y para que funcionen todos los cambios anteriores sobre el resto del fichero, hemos adaptados algunas instrucciones y declaraciones como se muestra en la siguiente salida del comando `diff`:[1]

```
>   protected IOPort port3;
>   protected IOPort port6;
106c120
<     IOUnit unit = cpu.getIOUnit("Port 5");
---
>     IOUnit unit = cpu.getIOUnit("Port 6");
108,109c122,123
<       port5 = (IOPort) unit;
<       port5.setPortListener(this);
---
>       port6 = (IOPort) unit;
>       port6.setPortListener(this);
125,127c139,141
<
<     IOUnit usart0 = cpu.getIOUnit("USART 0");
<     if (usart0 instanceof USART) {
---
```

[1] Comando mediante el cual se busca diferencias entre dos ficheros.

```
>    /* USART1 se comparte entre CC2420 (SPI Mode) y Bluetooth (serial mode) */
>     IOUnit usart1 = cpu.getIOUnit("USART 1");
>     if (usart1 instanceof USART) {
129,130c143,144
<       radio.setCCAPort(port1, CC2420_CCA);
<       radio.setFIFOPPort(port1, CC2420_FIFOP);
---
>       radio.setCCAPort(port2, CC2420_CCA);
>       radio.setFIFOPPort(port2, CC2420_FIFOP);
131a146
>       radio.setSFDPort(port1, CC2420_SFD);
133,137c148,152
<       ((USART) usart0).setUSARTListener(this);
<       port4 = (IOPort) cpu.getIOUnit("Port 4");
<       if (port4 != null) {
<         port4.setPortListener(this);
<         radio.setSFDPort(port4, CC2420_SFD);
---
>       ((USART) usart1).setUSARTListener(this);
>       port5 = (IOPort) cpu.getIOUnit("Port 5");
>       if (port5 != null) {
>         port5.setPortListener(this);
>         radio.setSFDPort(port5, CC2420_SFD);
139c154,171
<     }
---
>     }
>
>     unit = cpu.getIOUnit("Port 3");
>     if (unit instanceof IOPort) {
>       port3 = (IOPort) unit;
>       port3.setPortListener(this);
>     }
> //     IOUnit usart1 = cpu.getIOUnit("USART 1");
> //     if (usart1 instanceof USART) {
> //       ((USART) usart1).setUSARTListener(this);
> //       usart1.write(7, 0x30, false, 8);
> //     }
>
>     unit = cpu.getIOUnit("Port 4");
>     if (unit instanceof IOPort) {
>       port4 = (IOPort) unit;
>       port4.setPortListener(this);
>     }
235c267
<     if (source == port5) {
---
>     if (source == port4) {
246c278
<     } else if (source == port4) {
```

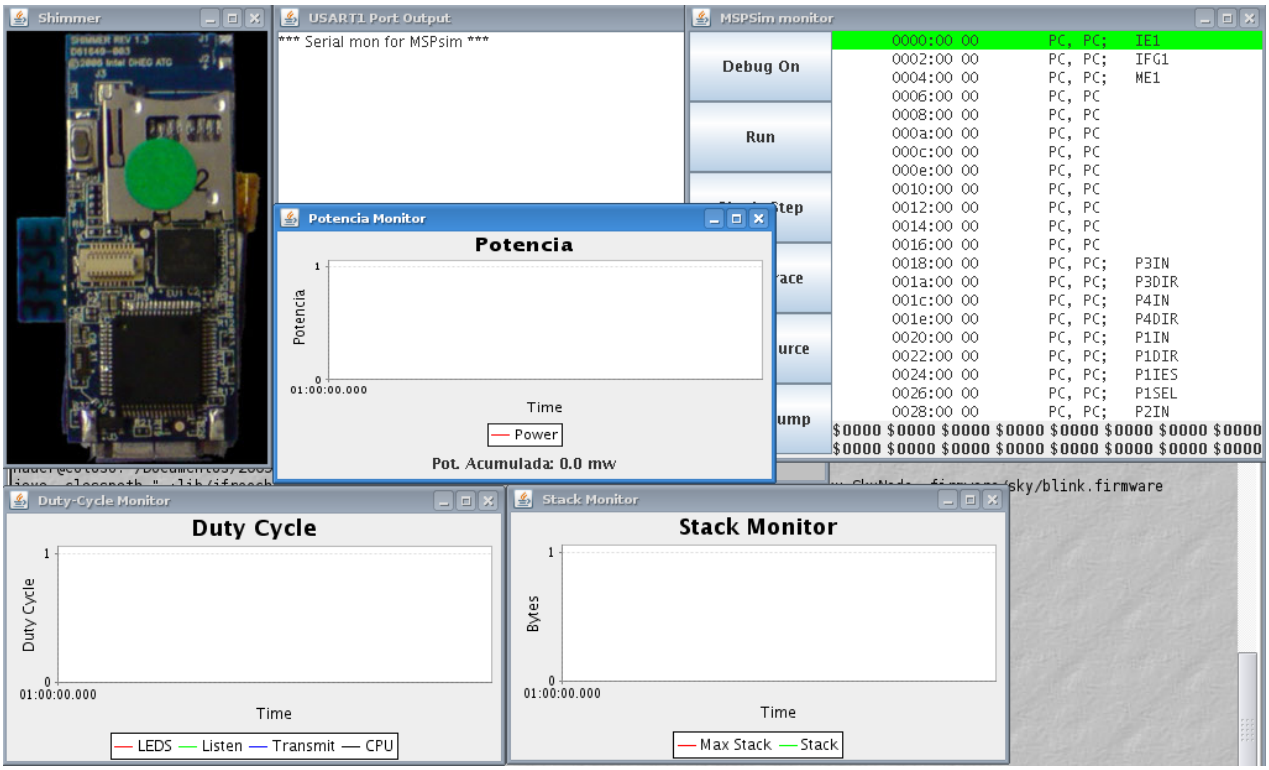


Figura 3.3: Simulador MSPsim para la plataforma SHIMMER.

```
---
>     } else if (source == port5) {
```

Capítulo 4

Programación del Puerto Serie USART1

4.1. Introducción al UART MODE

El MSP430 tiene dos puertos serie (*USART0 y USART1*), mediante los cuales se pueden enviar y recibir datos. En nuestro caso hemos trabajado con la recepción y transmisión de cadenas de caracteres e interpretación de comandos a través del puerto serie *USART1*. Para ello hemos programando varios ejemplos empleando entrada y salida programada y mediante interrupciones. En el modo asíncrono, los conectores del USART del MSP430 se conectan mediante dos pines (*URXD y UTXD*). Se selecciona el modo UART borrando el bit *SYNC*. Este modo incluye:

- 7 u 8 bits de datos con paridad par, impar o sin paridad.
- Transmisión independiente de recepción mediante registros de desplazamiento.
- Detección de comienzo de recepción y despertado de los modos *LPM*.
- Velocidad de transmisión programable.
- Soporte para la detección y eliminación de errores.

En UART MODE el USART transmite y recibe caracteres uno a uno de forma asíncrona a otro dispositivo. El tiempo de transmisión de cada carácter depende de la velocidad configurada en el USART. La estructura hardware del USART se muestra en la **figura 4.1**.

El formato de carácter del UART, tiene un bit de comienzo (*ST*), 7-8 bits de datos (*D7..0*), un bit de paridad par, impar o no paridad (*PA*); un bit de dirección (*AD)* y uno o dos bits de parada (*SP*)(ver **figura 4.2**).

4.1.1. Inicialización del USART

El USART se resetea mediante el bit *SWRST* o un *PUC* (después de un PUC, el bit SWRST es configurado automáticamente para hacer un reset). Un reset mediante el bit *SWRST* no altera el estado de los bits de recepción y transmisión (*URXEx* y *UTXEx*). El proceso de inicialización del USART es:

1. Configurar *SWRST*.
2. Inicializar todos los registros del USART con *SWRST = 1*.

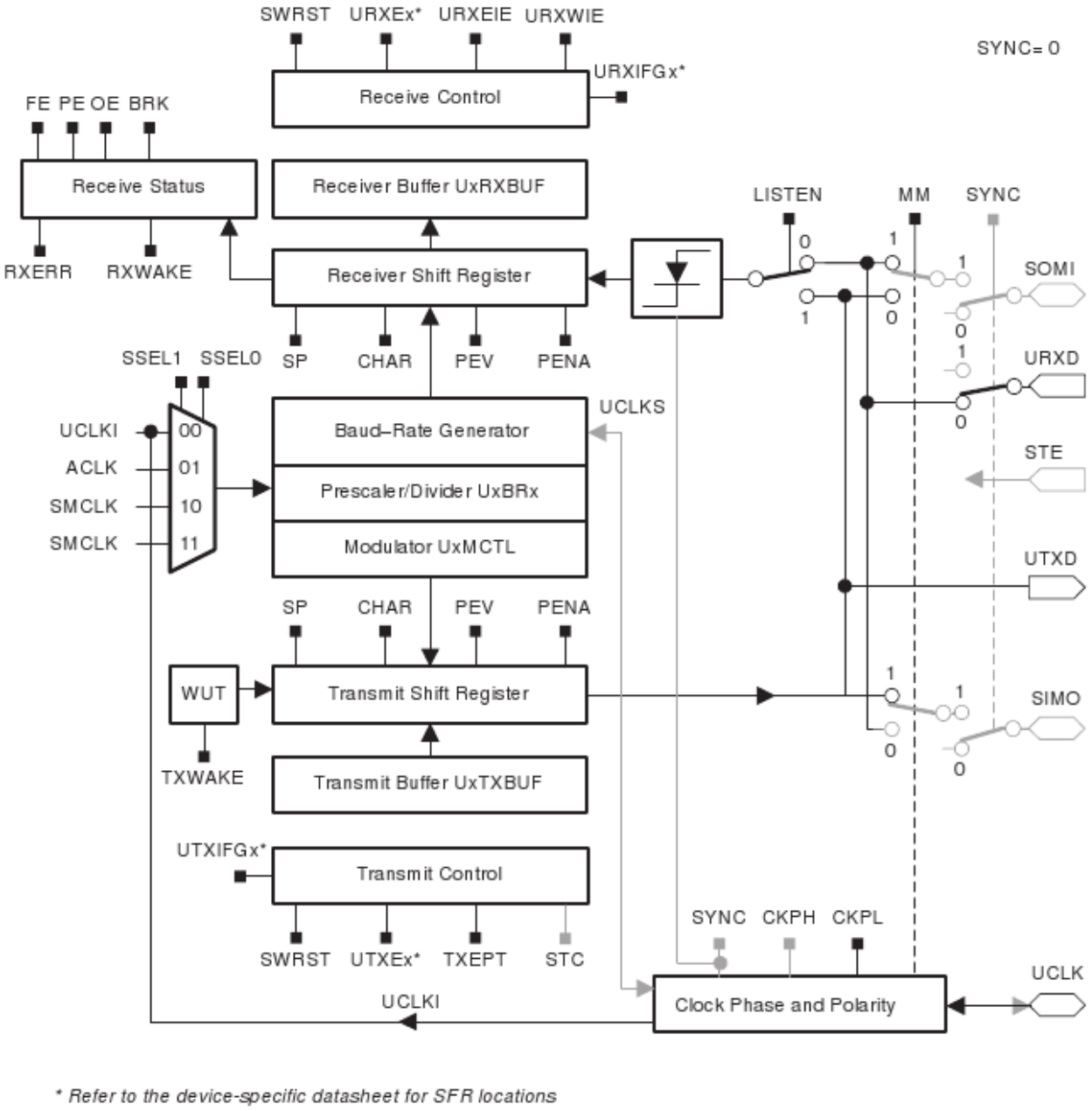


Figura 4.1: Estructura del USART.

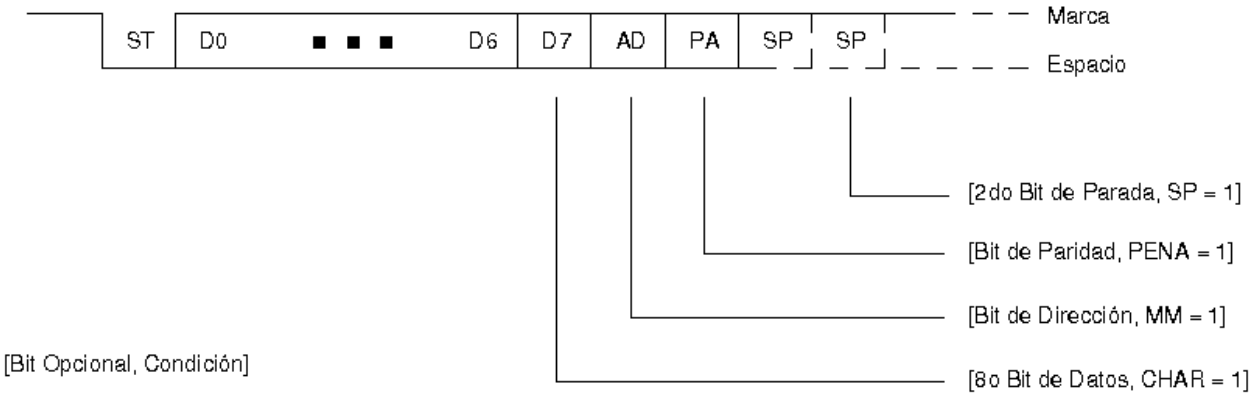


Figura 4.2: Formato de carácter.

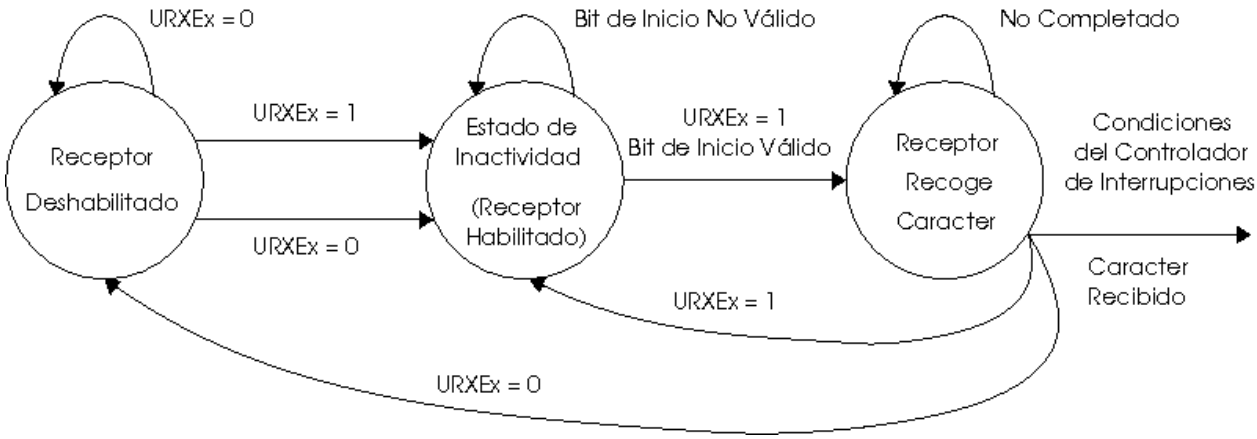


Figura 4.3: Diagrama de estados de USART en modo de recepción.

3. Activar USART mediante *Mex SFRx* (URXEx y/o UTXEx).

4. Borrar *SWRST* vía software.

5. Activar interrupciones mediante *IEx SFRs*.

4.1.2. Recepción desde el USART

El bit de activación de recepción (*URXEx*) activa o desactiva la recepción de datos. Desde el estado de recepción, en el momento que se recibe un valor correcto de bit de comienzo por el buffer de recepción (*UxRXBUF*) se recibe el carácter y vuelve al estado de recepción. En cualquier momento, si *URXEx = 0*, se desactiva la recepción (tanto en el estado de recepción como si está en plena operación de recibir, en cuyo caso dicha operación se detiene). Por tanto, la recepción es carácter a carácter. Un diagrama del funcionamiento aquí descrito puede observarse en la **figura 4.3**.

El bit de interrupción *URXIFGx* está activo cada vez que un carácter es recibido y cargado en el buffer de recepción *UxRXBUF*. Se genera una petición de interrupción de recepción cuando *URXIEXx* y *GIE* están

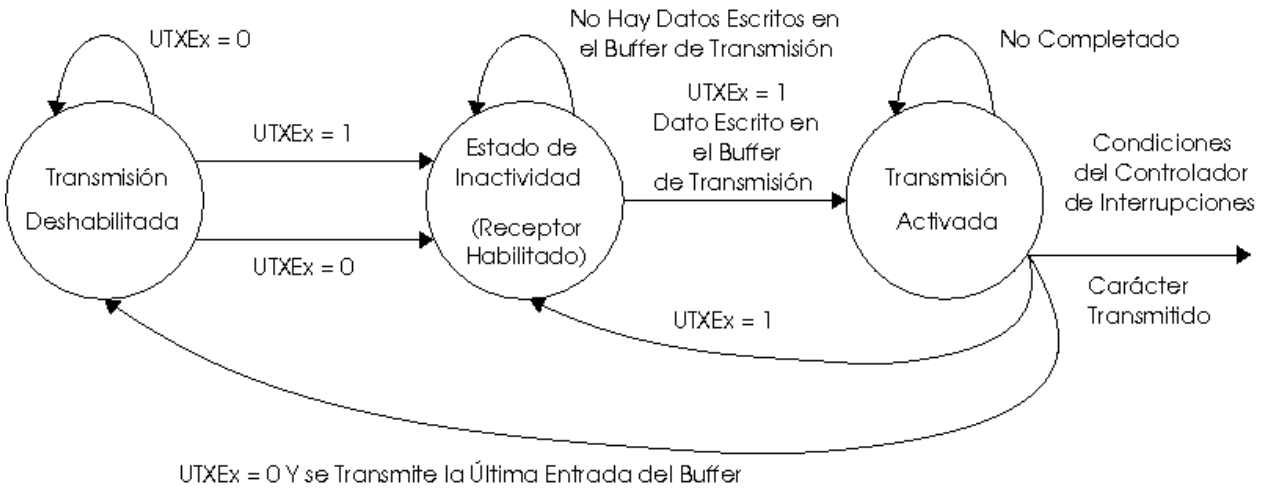


Figura 4.4: Diagrama de estados de USART en modo de transmisión.

activas. *URXIFGx* y *URXIEx* son reseteadas con un reset del sistema. *URXIFGx* es reseteada si la interrupción es atendida (*URXSE = 0*) o *UxRXBUF* es leído.

4.1.3. Transmisión por el USART

El bit de activación de transmisión (*UTXEx*) activa o desactiva la recepción de datos. El estado de transmisión se activa con *UTXEx = 1*. Cuando este estado está activo y se escribe en el buffer de transmisión (*UxTXBUF*), se transmite el carácter en el próximo cambio de *BITCLOCK*. Con *UTXEx = 0* se desactiva el estado de transmisión, y si es durante la transmisión de un carácter, este último terminará de transmitirse. Por tanto, la transmisión es carácter a carácter. Un diagrama de este funcionamiento aparece en la **figura 4.4**

El bit de interrupción *UTXIFGx* indica al *UxTXBUF* que está preparado para recibir otro carácter. Este bit se activa después de un reset. Si *UTXIEx* y *GIE* están activados se genera una interrupción. *UTXIFGx* es reseteado automáticamente si la petición de interrupción es atendida o si se escribe un carácter en *UxTXBUF*.

4.1.4. Registros de control y estado

En la **figura 4.5** se muestra una tabla con los registros programables del USART1.

4.2. Recepción y transmisión de caracteres

Para la programación de recepción de caracteres, tenemos un método principal que primero configura inicialmente los LEDs, el temporizador "watchdog" y las interrupciones globales; después llama a un método que inicializa "USART1" en modo "UART" haciendo un reset y configurando los registros de transmisión y recepción y sus correspondientes interrupciones; finalmente dejamos el "MSP430" en el estado de suspensión *LPM0*, que permite esperar en modo bajo consumo una interrupción de recepción o transmisión (el estado de suspensión es muy adecuado para este sistema, porque se busca el menor consumo posible). Estos ejemplos los hemos estudiado mediante el simulador MSPsim, que ha sido adaptado. Este simulador nos permite simular el apagado y encendido de los LEDs y la entrada y salida de datos por el puerto serie. La **figura 4.6** muestra el estado del SHIMMER justo antes de enviar el comando de encendido de los LEDs, por otra parte la **figura 4.7** muestra el estado del SHIMMER justo antes de apagarlo. También nos ofrece información variada del consumo y potencia. Concretamente nuestros ejemplos simulan que reciben una cadena de caracteres (finalizada por un

Registro	Forma Corta	Tipo de Registro	Dirección	Estado Inicial
Registro de control USART	U1CTL	Lectura / escritura	078h	001h con PUC
Registro de control de transmisión	U1TCTL	Lectura / escritura	079h	001h con PUC
Registro de control de recepción	U1RCTL	Lectura / escritura	07Ah	000h con PUC
Registro de control de modulación	U1MCTL	Lectura / escritura	07Bh	Sin cambio
Registro 0 de control de velocidad en baudios	U1BR0	Lectura / escritura	07Ch	Sin cambio
Registro 1 de control de velocidad en baudios	U1BR1	Lectura / escritura	07Dh	Sin cambio
Registro del buffer de recepción	U1RXBUF	Lectura	07Eh	Sin cambio
Registro del buffer de transmisión	U1TXBUF	Lectura / escritura	07Fh	Sin cambio
Registro 2 de habilitación del módulo SFR	ME2	Lectura / escritura	005h	000h con PUC
Registro 2 de habilitación de la interrupción SFR	IE2	Lectura / escritura	001h	000h con PUC
Registro 2 de marca de la interrupción SFR	IFG2	Lectura / escritura	003h	020h con PUC

Figura 4.5: Tabla de registros de estado y control de USART1.

`ENTER`) por el puerto serie que nosotros introducimos mediante el teclado, y la transmiten carácter a carácter de forma ordenada observándolo nosotros por la pantalla.

4.2.1. Entrada y salida programada

La entrada y salida programada no es la forma más adecuada de programar en este sistema (normalmente en ninguno), ya que el "MSP430" debe mantenerse en un bucle esperando a que el buffer de entrada tenga caracteres que hayan sido introducidos y por ello es más difícil dejarlo en estado de suspensión. Nuestro programa almacena caracteres de entrada mediante un buffer implementado como un "array" circular. Cada vez que se recibe un carácter, se introduce en el buffer y cuando el carácter recibido es un `ENTER`, comienza la transmisón. La transmisión está implementada mediante otro bucle que transmite de uno en uno cada carácter del buffer y simultáneamente lo vacía. Para ello ha sido necesario dejar un espacio de tiempo entre la transmisión de cada carácter para que puedan transmitirse todos completamente y no se solapen; éste es uno de los motivos por los que la entrada y salida programada no es adecuada, junto con el problema añadido de que las recepciones estarían inhabilitadas durante el tiempo que dura la transmisión. La transmisión de cada carácter se hace copiando el carácter a transmitir en el buffer de salida.

4.2.2. Entrada y salida mediante interrupciones

La forma más adecuada de programar en este sistema es mediante interrupciones, de esta forma el "MSP430" puede estar en estado de suspensión esperando una interrupción de recepción. Las interrupciones cambian los registros de estado y guardan el estado actual para poder volver al terminar la rutina de interrupción. En nuestro programa, cuando se recibe un carácter, se almacena en el buffer de recepción y se provoca una interrupción de recepción que almacena en nuestro buffer el carácter recibido, finaliza la interrupción y el procesador vuelve al estado de suspensión. En el momento que se recibe un `ENTER`, se transmite el primer carácter del buffer copiándolo en el buffer de transmisión. Al igual que en la mayoría de procesadores, las interrupciones de transmisión se provocan cuando termina la transmisión, en este caso de cada carácter; por tanto de esta forma se transmite el primer carácter, se provoca la primera interrupción de transmisión que transmite el siguiente carácter y así hasta que se vacía el buffer. Esta forma de programar, como se ha

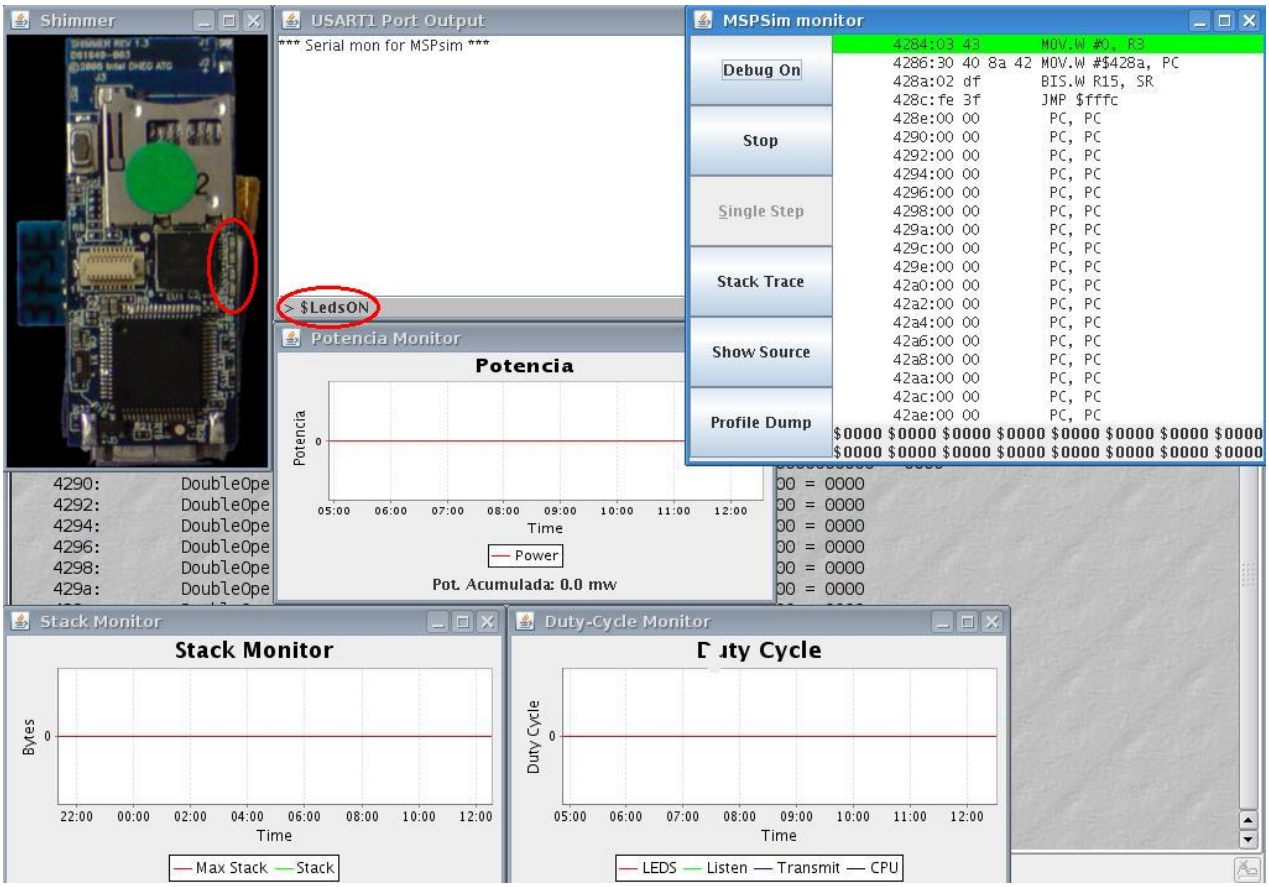


Figura 4.6: Estado del SHIMMER justo antes de recibir el comando de encendido de los LEDs.

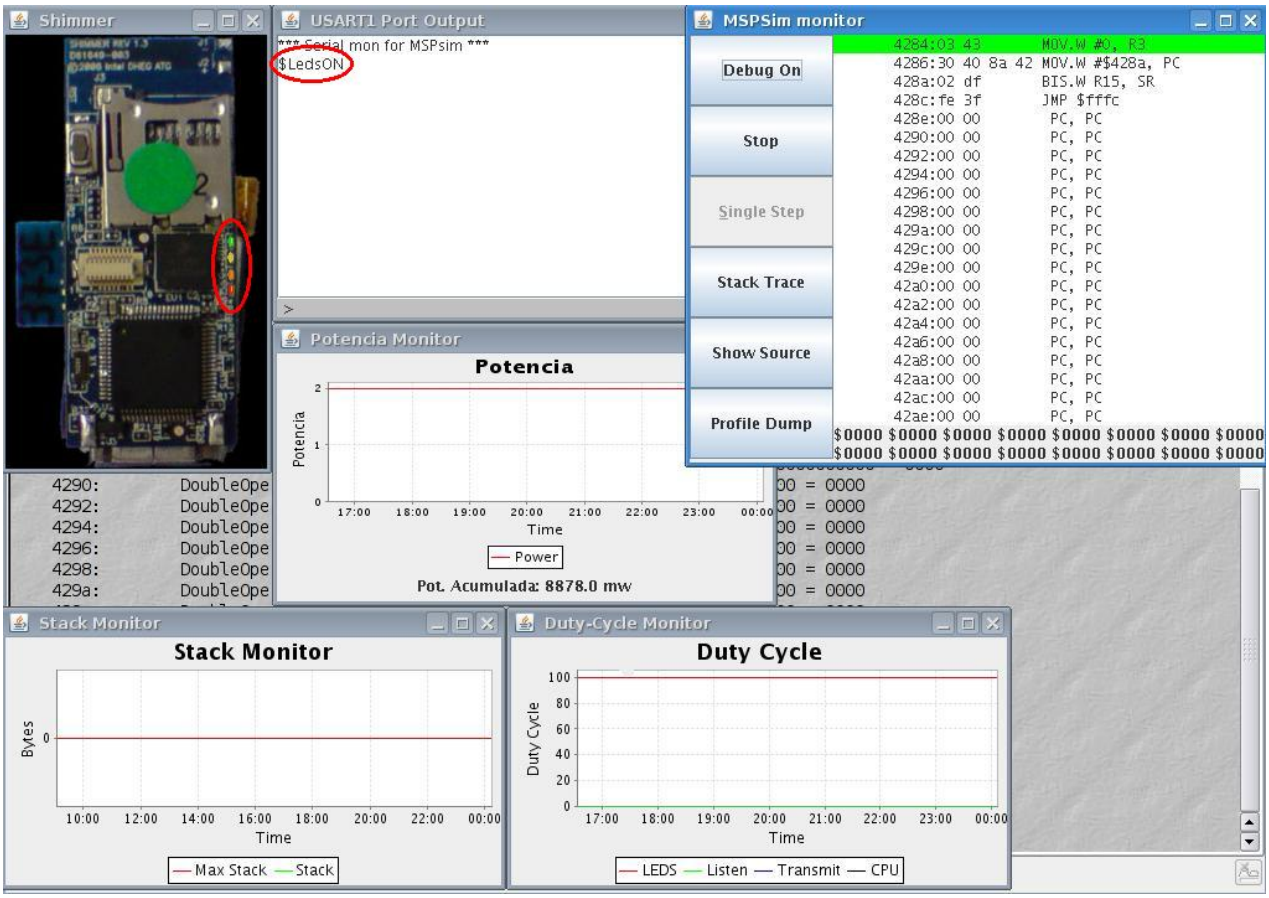


Figura 4.7: Estado del SHIMMER justo antes de recibir el comando de apagado de los LEDs.

comentado antes, es más adecuada porque el procesador no espera a que terminen de transmitirse todos los caracteres, sino que permanece en suspensión mientras no esté transmitiendo caracteres y durante el tiempo de transmisión de un carácter. La recepción de caracteres también funciona de esta forma y permite al procesador estar en modo suspensión hasta que se detecta un carácter entrante y se activa la correspondiente interrupción de recepción. Para otros programas algo más complejos que implementen otro tipo de tareas también es adecuado el uso de interrupciones, porque permiten al procesador estar ejecutando instrucciones hasta que llegan las interrupciones y continuarlas al terminar la interrupción en vez de entrar en el sistema de suspensión.

4.2.3. Procesamiento de pequeñas órdenes

A partir de la programación de entrada y salida de caracteres mediante interrupciones, hemos ampliado el ejemplo anterior para que al introducir un `ENTER` además de transmitir los caracteres anteriores al `ENTER`, intenta procesarlos; en nuestro caso interpreta un comando que enciende los LEDs y otro que los apaga.

Capítulo 5

Diseño e Implementación de Algoritmos de Encaminamiento

5.1. Introducción

A lo largo de los siguientes puntos se abordarán los problemas y soluciones que se han planteado a la hora de resolver los dos aspectos principales a los que debe enfrentarse el algoritmo. Éstos son: buscar y elegir a qué nodo transmitir, y cómo realizar dicha transmisión.

La elección de C como lenguaje de implementación obedece principalmente a la existencia de compiladores cruzados para otras arquitecturas (por ejemplo el MSP430GCC para los sensores SHIMMER [SHI08]). La adaptabilidad de la implementación a otras plataformas queda así en gran parte simplificada.

Por otra parte, se ha elegido Bluetooth como tecnología de transmisión, por ser un hardware muy presente en muchos tipos de dispositivos y sensores fijos y móviles. Además se puede acceder muy fácilmente a la interfaz de programación y al código de implementación de herramientas que sirven como ejemplo para el entorno de sistemas operativos Linux.

Nuestro trabajo en este aspecto ha requerido hacer primero un diseño del algoritmo para más tarde programarlo en lenguaje C. Para la implementación del algoritmo en C se han usado funciones predefinidas en la *API* de BlueZ [HH] y además funciones de C y de UNIX relativas al manejo de ficheros, formateo de cadenas y apertura y cierre de sockets (consultar la librería C de GNU [GNU]) Una vez que se ha comenzado a implementar, se han ido hecho pruebas poco a poco para mejorar el algoritmo teniendo que volver varias veces al diseño para discutirlo y mejorarlo.

5.2. Diseño del Algoritmo

El objetivo del algoritmo es poder enviar datos de un nodo origen a alguna de las estaciones bases, siendo estas últimas definidas previamente. Una de las dificultades que ha de solventar el algoritmo es la comunicación entre el nodo origen y el nodo destino cuando están alejados y no es posible la comunicación directa, por tanto es necesario en estos casos la intervención de nodos intermedios que reenvíen de un nodo a otro el mensaje facilitando que llegue correctamente a su destino. El algoritmo planteado es apropiado para el uso de tecnología Bluetooth en sensores.

Aunque se han propuesto muchos algoritmos para el encaminamiento en redes de sensores, en general éstos presentan el problema de su complejidad y la falta de movilidad de los sensores [FPH05]. En dichos algoritmos se parte de una red de sensores fijos y se observa que existe una alta dependencia de la integridad estructural de dicha red de sensores. Además se puede apreciar una alta complejidad en la implementación del establecimiento de las rutas de comunicación en la red. Por otra parte el algoritmo que se propone aquí tiene

como puntos clave la alta movilidad de los sensores y la simple y rápida readaptación frente a posibles cambios en la topología de la red.

Se plantean varios problemas como el formato de los envíos, a quién enviarlos y cómo han de reaccionar los intermediarios para que lleguen al destino.

5.2.1. Algoritmo de renombramiento y envío de información entre nodos

Es posible que en cualquier momento un dispositivo de la red desee enviar información, ya sea propia del dispositivo o recibida de otros dispositivos. Todos los nodos pertenecientes a la red hacen cada cierto tiempo una exploración de dispositivos para conocer el estado actual de nodos alcanzables. Una vez hecha esta exploración, el nodo emisor enviará directamente los datos a una estación base si ésta es alcanzable. Sin embargo, si no es alcanzable, deberá enviárselos al nodo con el que menos reenvíos se requieran para llegar finalmente a alguna estación base. De esta forma se asegura que la información pase por el menor número de intermediarios posible y por tanto el envío sea lo más rápido posible.

Para que se cumplan estas condiciones, las estaciones base están previamente definidas y se nombran como 1 y cada vez que los nodos hacen una exploración de dispositivos alcanzables, se renombran automáticamente al número inmediatamente superior al menor encontrado. Se hace referencia al conocido como *nombre amigable* del dispositivo (ver **apartado 5.3.1**). Cuando un dispositivo quiere enviar información, intentará enviarla al nodo que esté a su alcance y tenga como nombre el menor número posible de forma que la información viaje lo más rápido posible y de la forma más directa como se muestra en la **figura 5.1**. Es necesaria una correcta sincronización entre el nodo emisor y el receptor, ya que la información se envía por tramas y el nodo emisor debe esperar a que el nodo receptor esté preparado.

Para este algoritmo existe un problema de la cuenta al infinito o convergencia lenta. Este problema sucede en los casos en los que por la desaparición de un nodo y la lejanía con el resto, un grupo de nodos se queda incomunicado respecto a las estaciones base. Cuando esto sucede, los nodos de dicho grupo continúan haciendo exploraciones y encuentran siempre al resto de nodos del grupo incrementando lentamente y de uno en uno en cada exploración su renombrado hasta infinito.

Una posible idea para abordar esta situación es no tener en cuenta para el renombramiento propio el siguiente escenario: un nodo que ha estado conectado a otro, en una posterior exploración detecta que el *nombre amigable* del último se ha incrementado de forma que tiene un número inmediatamente superior al primero. En la **figura 5.2** se ilustra un ejemplo.

Esta solución es demasiado restrictiva, ya que la situación que se plantea entre los anteriores sensores no sólo se da en el caso de que queden aislados. Por ejemplo, tal y como se muestra en la **figura 5.3**, si no se ha producido una incomunicación con las estaciones base, puede haber casos en los que se impida el renombramiento cuando sí se debe realizar para poder llegar a alguna de las estaciones base.

5.2.2. Formato de envío

Los sensores se mueven de forma continua, por ello cada cierto tiempo hacen una búsqueda de dispositivos e inician de nuevo el renombramiento. El movimiento de los sensores puede provocar que se interrumpa un envío de datos al salir un nodo del alcance de otro. Esto puede provocar que las tramas lleguen en desorden y a través de diferentes intermediarios y por ello sea necesario reconstruir la información a su llegada a la estación base. Para ello y usando como referencia el formato de tramas de HDLC (High-Level Data Link Control) [Sta04] se ha decidido un formato de trama similar, ya que es muy útil para llevar cuenta del número de trama enviado y/o recibido, y así poder reconstruir la información correctamente en el nodo base.

Tipos de tramas

Las tramas que se envían de un dispositivo a otro pueden ser de dos tipos. Por un lado están las tramas generales de información y por otro, las respuestas *ACK* de Supervisión. Las tramas de información se usan para enviar datos e información relacionada a dichos datos como el momento de creación de la trama, el emisor

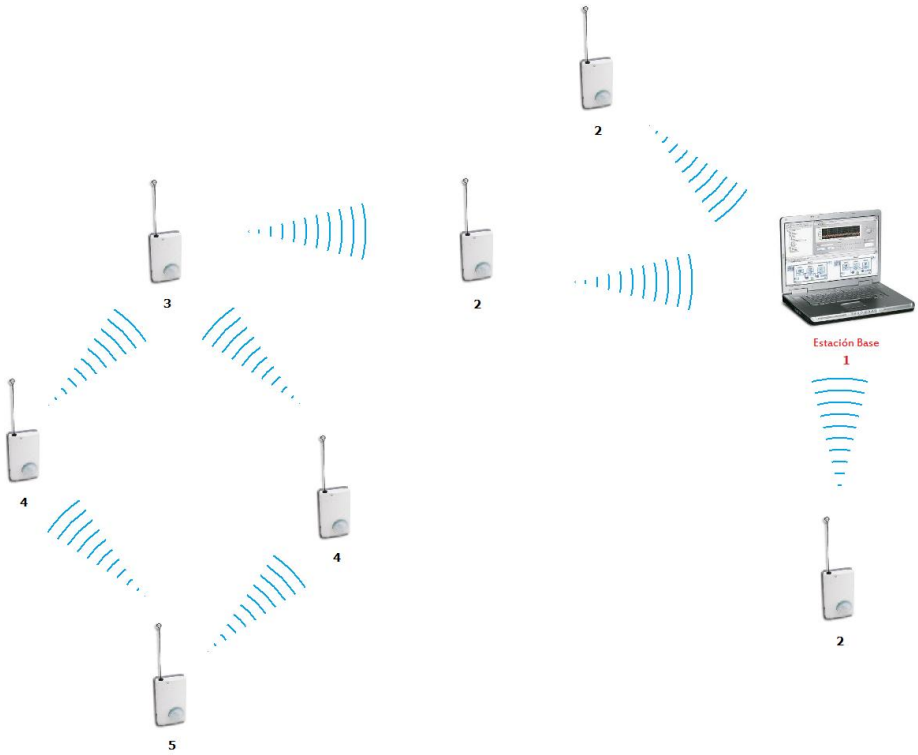


Figura 5.1: Esquema del algoritmo de renombramiento y envío.

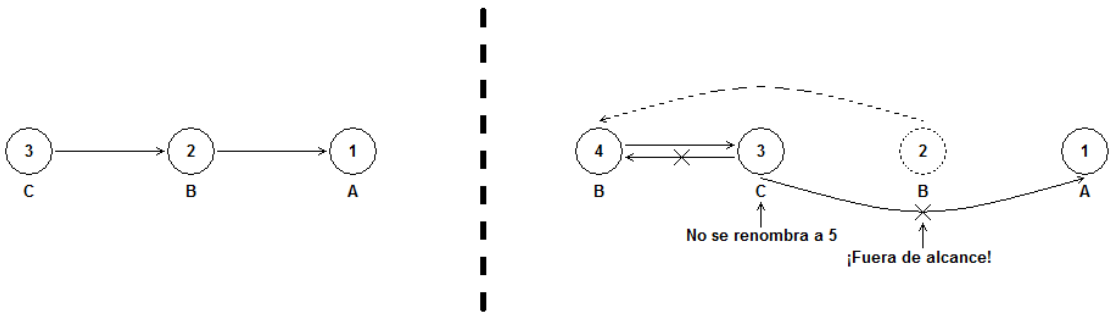


Figura 5.2: Ejemplo con nodos aislados en la red.

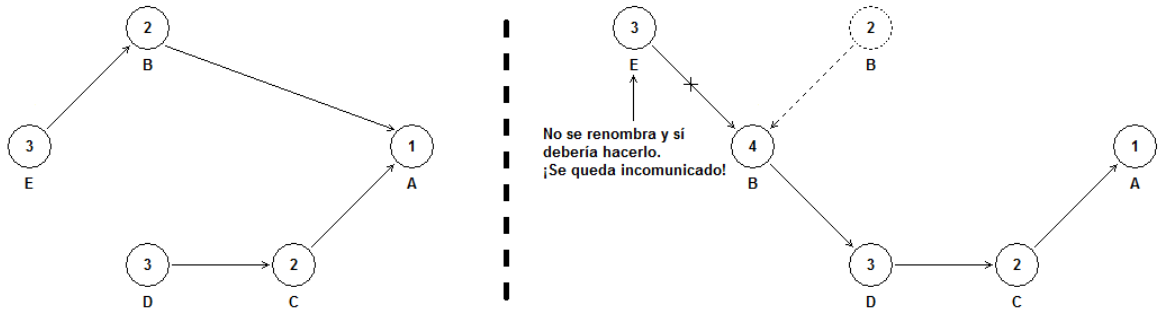


Figura 5.3: Ejemplo de mal funcionamiento de la propuesta de solución de cuenta al infinito.

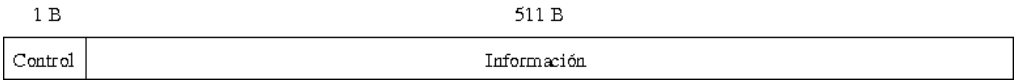


Figura 5.4: Formato general de una trama.

inicial y los datos que se envían. Las respuestas ACK se usan para confirmar los datos que se han recibido y si el dispositivo receptor está listo o no para recibir la siguiente trama de información.

Formato de tramas

Las tramas de información tienen una longitud máxima de 512 bytes y una longitud mínima de 1 byte. La longitud de cada trama depende del nivel de ocupación del campo de información.

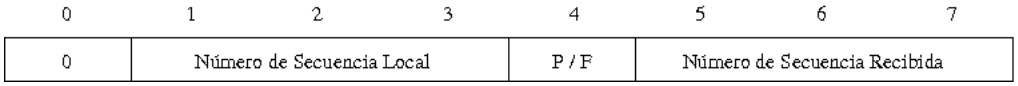
Todas las tramas tienen un primer byte de control basado en HDLC a pesar de que este formato excede las necesidades actuales, ya que vamos a utilizar, como se verá en el apartado **5.3.3**, sockets L2CAP orientados a conexión. En cualquier caso se ha elegido para dejar abiertas posibles modificaciones futuras en las que se tenga acceso al control de flujo. Este primer byte es el encargado de indicar:

- Tipo de trama: De información si el primer bit es 0 o de supervisión si los dos primeros bits son 10.
- Número de secuencia local: se usan 3 bits para indicar un número de secuencia local que interesa para determinar el correcto orden de las tramas recibidas.
- Tipo de ACK: Se usarán dos tipos de respuestas ACK, las *Receiver Ready* (RR confirmación y solicitud de la siguiente trama) y las *Receiver Not Ready* (RNR confirmación y espera).

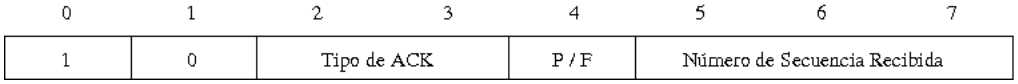
Las respuestas ACK de tipo RR sirven para indicar al emisor que una trama se ha recibido correctamente y que está preparado para recibir la siguiente. En cambio las de RNR indican que se ha recibido correctamente pero no está aún listo para recibir la siguiente. En ambos casos se necesita enviar el número de secuencia local que se desea confirmar al emisor.

Para las tramas de información, además de ese primer byte de control, se utilizan otros 511 bytes para adjuntar información. Para el envío de datos se utilizan:

- 1 byte para información de tiempo.
- 6 bytes para la MAC origen (MAC de donde proviene la trama inicialmente).

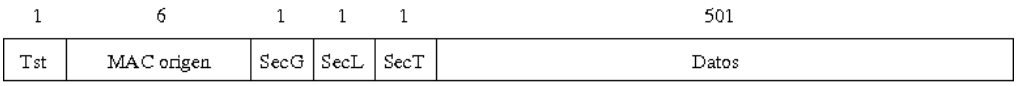


(a) Trama de Información



(b) Trama de Supervisión

Figura 5.5: Formato del Byte de Control.



Tst – Secuencia Timestamp

SecG – Numero de Secuencia Global

SecL – Numero de Secuencia Local

SecT – Numero Total de Secuencia

Figura 5.6: Formato de los 511 bytes de carga útil de una trama de información.

- 1 byte para el número de secuencia global que junto con la MAC sirve para identificar el flujo de información de forma única.
- 1 byte para el número de secuencia local que sirve para comprobar que las tramas llegan de forma ordenada.
- 1 byte para el número total de secuencias de los datos, que sirve para saber cuándo se ha recibido la última trama.
- Los 501 bytes restantes para transmitir los datos en sí.

5.2.3. Escenario de pruebas

Para una prueba de viabilidad en este diseño se ha dispuesto un escenario que presenta todos los elementos involucrados.

Existen varios nodos clientes intermediarios, uno de ellos enviará datos a otro para que este último los reenvíe hasta que lleguen a una estación base. Esta última tendrá como *nombre amigable* el 1. Este escenario funcionará de la siguiente manera:

La estación base

Está nombrada a 1 y se encarga de recibir tramas de información de uno o varios dispositivos y de confirmarlas mediante respuestas ACK. Por otra parte debe reconstruir e interpretar las tramas que recibe procesando los datos que inicialmente los clientes han enviado.

Un cliente intermediario

Decide en un momento dado enviar datos a alguna estación base de la red. Cada cierto tiempo debe realizar una búsqueda de dispositivos y almacenar todos los dispositivos encontrados renombrándose a sí mismo como `número_nodo_más_bajo + 1`, como ya se mencionó en el **apartado 5.2.1**. Intentará enviar la primera trama a los dispositivos nombrados con un `número_propio - 1` hasta que haya uno que acepte. Cada vez que se envíe una trama se esperará una respuesta ACK de confirmación. La misma respuesta ACK indica la trama que se ha recibido y si se está listo para recibir más datos o se debe esperar. En caso de tener que esperar demasiado tiempo se puede enviar a otro nodo alternativo si existe. En el peor de los casos también se puede cancelar el envío si no disponemos de más nodos libres al alcance con un número menor.

Además de poder enviar datos generados, servirá de enlace entre clientes y estaciones base u otros intermediarios que no pueden verse directamente por la distancia. Debe recibir y confirmar las tramas mediante respuestas ACK y reenviarlas al nodo menor que encuentre para facilitar el viaje hasta una estación base.

5.3. Detalles de la Implementación

Para implementar el algoritmo, se utilizarán las librerías pertenecientes a la pila de protocolos de *Bluez* [HH]. Además, se han usado las explicaciones y ejemplos que pueden encontrarse en [SH]. El objetivo es realizar una implementación a bajo nivel, de manera que la solución propuesta trabaje en máquinas Linux. Así se podrá validar el funcionamiento del algoritmo de encaminamiento antes de pasar a su implementación en los nodos sensores.

Los cambios que habría que hacer para el funcionamiento fuera del entorno Linux actual, serán relativos a la interfaz que otras opciones presenten con respecto a su dispositivo de comunicación: aunque sea también Bluetooth, no tiene por qué funcionar con la pila de BlueZ, o incluso puede que se utilicen otras tecnologías, como ZigBee, etc.

Esto se haría de la siguiente manera:

- Prescindir de las funciones que usan las librerías *bluetooth*, *hci hcilib*, *L2CAP* de *BlueZ*, pues pertenecen a la pila de protocolos Bluetooth de Linux. En su lugar, se deberá configurar el dispositivo Bluetooth de SHIMMER, mediante sus correspondientes comandos AT, enviados a través del puerto serie configurando previamente el módulo Bluetooth del SHIMMER en modo programación.

- Prescindir del uso de sockets Bluetooth con el protocolo *L2CAP*, pues de nuevo es funcionalidad implementada por BlueZ. En el caso del SHIMMER, enviaremos las tramas a través del puerto serie, programando el módulo Bluetooth del SHIMMER en modo datos.

- Para manejar el puerto serie, podemos usar el software que desarrollamos y probamos a tal efecto en el simulador. Consultar el **apartado 4.1** (que se encuentra a partir de la página 23).

El resto de la implementación del algoritmo no se vería afectada por modificaciones.

Una vez claras las principales diferencias a tener en cuenta para cambiar de entorno, nos centramos en la implementación del algoritmo en *GNU/Linux* con el compilador *gcc* y la api y herramientas de BlueZ.

5.3.1. Renombramiento dinámico de los dispositivos

Los dispositivos Bluetooth pueden llevar un *nombre amigable* independiente de su dirección hardware. Esta última es de la misma naturaleza que las *direcciones MAC de Ethernet*. El nombre puede ser fijado por el usuario a través de la herramienta *hciconfig*, o bien utilizando la siguiente función de la librería *hci_lib*:

```
int hci_write_local_name(int dd, const char *name, int to)
```

Esta función toma como parámetros el identificador del dispositivo `dd`, que es un entero que identifica al `socket hci` abierto para que el ordenador se comunique con la interfaz Bluetooth que se desea configurar, un puntero al *nombre* que se le quiera dar, y un timeout `to` para que le dé tiempo al ordenador anfitrión a escribir el nuevo nombre en el dispositivo. Hemos combrobado experimentalmente que funciona bien otorgando un timeout de **100 mseg**. Devuelve 0 en caso de éxito, y -1 en caso de error.

La función usada por `hci_lib` que encapsula los detalles de bajo nivel de la apertura de dicho `socket hci` para el parámetro `dd` necesaro para la función anterior es :

```
int hci_open_dev(int dev_id)
```

En este caso, la función toma como parámetro `dev_id`, que es un entero que identifica a la *interfaz Bluetooth* correpondiente al dispositivo. Se van numerando en orden a partir del 0, de manera que si le pasamos un 0, estaremos haciendo referencia a la interfaz `hci0`, si le pasamos 1 referenciamos a `hci1` ... Si queremos que se asocie automáticamente a la primera interfaz libre (si es que tuviéramos más de una interfaz hardware Bluetooth en el computador), o a la única caso de estar libre (si sólo hubiera una), podemos pasar como parámetro `NULL`. Como valor devuelto, obtendremos un entero que representa el *descriptor de fichero* asociado al *socket* abierto, o -1 en caso de que se produzca un error.

Para poder llevar a cabo la exploración de los dispositivos que puedan estar al alcance de una determinada estación, usando la api de BlueZ, se hará uso de lo siguiente:

```
int hci_inquiry(int dev_id, int len,
        int num_rsp, const uint8_t *lap,
            inquiry_info **ii, long flags)
```

El primer parámetro es el valor devuelto por la función `hci_open_dev`, identificando al `socket hci` correspondiente. El siguiente parámetro `num_rsp` es un entero que fija el número máximo de estaciones que queremos que nuestro dispositivo detecte. Del siguiente parámetro, basta saber que para nuestros propósitos debe ir a `NULL`. El siguiente, que es un puntero a una estructura de la librería `hci` (`inquiry_info`), almacenará las direcciones hardware de los dispositivos encontrados, que vamos a necesitar posteriormente para poder leer sus nombres. El último parámetro es un valor que se debe fijar con la macro `IREQ_CACHE_FLUSH`, para el correcto funcionamiento la exploración. El valor devuelto es un entero que representa el número de estaciones encontradas (-1 si se ha producido un fallo).

Finalmente, para poder leer los nombres de los dispositivos remotos encontrados, y saber así a cual nos tenemos que conectar (renombrándonos además en consecuencia con la función `hci_write_local_name`) usaremos la función :

```
int hci_read_remote_name(int dd,
        const bdaddr_t *bdaddr, int len, char *name, int to)
```

El parámetro `dd` es el identificador del socket abierto con la función `hci_open_dev`. El parámetro `len` es la longitud máxima que puede contener el nombre de un dispositivo. El siguiente, es un puntero al parámetro de salida que nos interesa obtener, que es el nombre del dispositivo remoto. Por último, el valor de `timeout`, que como hicimos en el caso de la función `hci_write_local_name`, y por la misma razón, fijaremos en 100 **mseg**. Estas son las funciones de la librería `hci_lib` de BlueZ necesarias para componer la función `void scaneo (void)`, con la que se lleva a cabo el objetivo de *renombramiento* que explicamos en este apartado. El nombre que un dispositivo se otorga a sí mismo, como ya se ha explicado en el apartado **5.2**, no tiene por qué ser el mismo siempre (de ahí el nombre *Renombramiento Dinámico*). Dada la naturaleza potencialmente móvil de la red de sensores, cada cierto tiempo, cada dispositivo de la red (a excepción de la *estación base*), llamará a esta última función, para redetectar a las estaciones más cercanas (pueden haber cambiado) y renombrarse en consecuencia. Puede consultarse el resto de código del algoritmo implementado para resover el problema de este apartado, en `src/cliente.c` , en concreto, la función `void scaneo (void)`.

Si se desea ampliar información sobre las herramientas usadas en este punto, se puede consultar la siguiente dirección web (*Bluetooth Programming in C with BlueZ*):

http://people.csail.mit.edu/albert/bluez-intro/c404.html

También se remite al lector al documento [SHR07], capítulo 3.

5.3.2. Establecimiento de la conexión

Implementada la parte del algoritmo destinada a descubrir qué nodos hay al alcance, y a cuál de ellos conectar, se muestra cómo se lleva a cabo dicha conexión con la *api* de BlueZ.

La interfaz *hci* permite actuar directamente sobre el hardware. Así, a través de dicha interfaz se pueden configurar los dispositivos Bluetooth del computador, para efectuar exploraciones, renombramientos, etc ...

Por encima de esta capa física, Bluetooth necesita establecer *mecanismos de control de enlace* entre los dispositivos interesados, para poder establecer de manera óptima y segura una conexión y mantenerla en el tiempo, con objeto de poder transmitir información.

La primera capa que cumple este cometido es L2CAP.[1] Implementa un protocolo basado en paquetes que puede ser configurado cumpliendo diferentes niveles de fiabilidad.

En el modo de funcionamiento que interesa para el propósito del algoritmo, L2CAP implementa un esquema de *transmisión - confirmación* por cada paquete que el emisor envía, de manera que antes de enviar el siguiente paquete, se espera un tiempo (*timeout*) bastante elevado hasta que se recibe confirmación. Si ese tiempo se agota, se producirá un error, y no se sigue con la transmisión (se supone que el destinatario ya no está disponible).

Elección del protocolo de control de enlace.

Puede parecer que es más apropiado elegir la interfaz de la capa RFCOMM,[2] pues está presente en todas las implementaciones de la pila de protocolos para diversos sostemas operativos (Symbian OS, Mac OS X, Windows, Linux ...), y por tanto este diseño sería más portable. Pero no se persigue portablidad a tan alto nivel, se persigue **eficiencia**, y que la adaptación de esta implementación a entornos con arquitecturas más especializadas sea razonablemente sencilla.

Por tanto, *se elige la capa L2CAP* por ser lo suficientemente cercana al dispositivo como para ser eficiente, permitiendo trabajar con sockets de manera muy parecida a los *sockets de C en Unix* [Chu] y [Már04].

Habiendo explicado cómo se puede averiguar a qué dispositivo conectar y qué *protocolo de control de enlace* se va a usar, se utilizará la interfaz `bluetooth.h` y `l2cap.h` de la *api* de BlueZ, para establecer *sockets Bluetooth* que utilicen como protocolo L2CAP.

Sockets Bluetooth

Un socket es una herramienta de programación que permite que se pueda establecer una comunicación entre dos procesos a través de una red. En el caso de Bluetooth, a nivel de la capa de control del enlace de datos, hay dos tipos de sockets, coincidiendo con los dos protocolos ya vistos:

- RFCOMM sockets: no se usarán.
- L2CAP sockets: son los elegidos, pues a ese nivel se establecen las conexiones.

5.3.3. Sockets L2CAP

Según se explica en el **apartado 5.2.3**, se toman en consideración dos tipos de dispositivos: nodos clientes intermediarios y estaciones base. Se detalla ahora la implementación de cada una de ellos.

[1] Logical Link Control and Adaptation Protocol
[2] Radio Frecuency Communications

El cliente intermediario

Genera o reenvía datos. Comportándose como generador de información, cada cierto tiempo el nodo realiza una búsqueda de dispositivos y se renombra adecuadamente mediante el mecanismo de exploración. Éste permite conocer no sólo el *nombre amigable* de los dispositivos al alcance, además obtiene sus direcciones MAC. Así puede determinar la dirección del dispositivo al que interesa conectar.

Una vez obtenida dicha MAC, el nodo pasa a solicitar la conexión.

A través de una estructura `sockaddr_l2` se almacenan los parámetros de conexión (la dirección MAC de envío, la familia y el puerto). Acto seguido, se forman las tramas a partir del flujo de datos que constituye la naturaleza del envío. El primer byte de cada trama se rellenará con la correspondiente información de control.

A continuación se componen los siguientes campos de la trama correspondientes al campo `información`. Éstos son:

-TimeStamp: Obtenido mediante la función `time()` de C.

-MAC Origen: A través de la función de BlueZ:

```
int hci_read_bd_addr(int dd, bdaddr_t *bdaddr, int to)
```

- Número de secuencia global, que junto a la MAC origen identifican al flujo de datos de forma única.
- Número de secuencia local del fragmento de datos que se envía.
- Número total de fragmentos de datos.
- Datos.

Para proceder al envío se necesita abrir un socket a través de la siguiente función:

```
int socket(AF_BLUETOOTH,SOCK_SEQPACKET,BTPROTO_L2CAP)
```

Con el socket abierto y como se muestra a continuación, se conecta con el nodo elegido usando el socket que hemos creado, la estructura de la MAC y su tamaño:

```
int connect (int socket, struct sockaddr *addr,
    socklen_t length)
```

A partir del momento en que la conexión ha sido realizada, se procede a enviar cada trama. Éstas se almacenan en un buffer de caracteres.

Enviada cada trama, hay que esperar su confirmación ACK de respuesta mediante una función que se encarga de crear un socket adecuado para la recepción y de procesar dicha ACK, si se recibe. Cuando la respuesta ACK es recibida se forma la siguiente trama que corresponda.

En su comportamiento como dispositivo de reenvío, el nodo implementa su funcionalidad de la siguiente manera: se crea un socket para la recepción y el nodo queda a la escucha. Cada vez que se recibe una trama, procesa la información de control y diseña una respuesta ACK de confirmación y espera, cerrando la conexión y el socket con la estación fuente. A continuación se abren adecuadamente para reenviar la trama al dispositivo destino; al igual que con el cliente, espera la ACK correspondiente y si todo ha sido correcto cierra de nuevo las conexiones y los sockets y las reabre para enviar la correspondiente respuesta ACK de confirmación y petición de continuar al dispositivo origen.

La estación base

Se encarga de recibir tramas y procesar la información que albergan. Además debe enviar las respuestas ACK.

Este nodo está permanentemente dispuesto para que le llegue información: al iniciarse crea un socket para la recepción de datos y se pone a la escucha. Para ello se asocia por ejemplo el puerto 0x1001[3] del primer

[3]El rango de puertos para L2CAP es 0x1001 - 0x7FFF

adaptador Bluetooth disponible y usa la función `bind()` de C para que se asocie el socket `s` que está abierto, con la dirección MAC correspondiente. Se crea un socket para la recepción y el nodo queda a la escucha. Cada vez que se recibe una trama, procesa la información de control y diseña una respuesta ACK de confirmación y receptor listo. Se muestra un ejemplo de toda esta secuencia:

```
loc_addr.l2_family = AF_BLUETOOTH;
loc_addr.l2_bdaddr = *BDADDR_ANY;
loc_addr.l2_psm = htobs (0x1001);
bind (s, (struct sockaddr *) &loc_addr, sizeof (loc_addr));
```

Ya está listo para ponerse a la escucha usando la siguiente función:

```
int listen (int socket, int n)
```

La estación base acepta una conexión del cliente y recibe cada trama, para ello usa:

```
int accept (int socket, struct sockaddr *addr,
    socklen_t *length_ptr)
```

Una vez recibida cada trama, se trata primero el primer byte que lleva la información de control y después el resto de campos para poder interpretar su significado. Finalmente y antes de ponerse a la escucha para recibir más tramas, envía la respuesta ACK.

Las funciones más importantes que hemos implementado para ello son:

```
void tratarPrimerByte (unsigned char buf[Tamaño de trama])
```

Esta función recibe por parámetro un buffer de caracteres con la trama y trata la información de control mediante operaciones binarias.

```
void responder (int cliente)
```

Esta función crea un socket adecuado para envío de información y diseña la respuesta ACK en función de la trama recibida y las necesidades del servidor.

5.3.4. Características de la Funcionalidad del Algoritmo

Se señalan las siguientes características:

- Atender a varios nodos: los dispositivos deben poder atender varias conexiones a la vez como se muestra en el diagrama de flujo de la **figura 5.7**.
- Ahorro de energía: los intentos de envío y/o recepción por parte de los nodos, deberán ir acompañados de un *timeout*, que haga que no se queden bloqueados indefinidamente si no es posible acometer con éxito alguna de estas acciones.
- Doble funcionalidad: el cliente intermediario hará un reparto de tiempo para realizar su tarea de envío de tráfico propio y su función de reenvío de tráfico generado por otros.

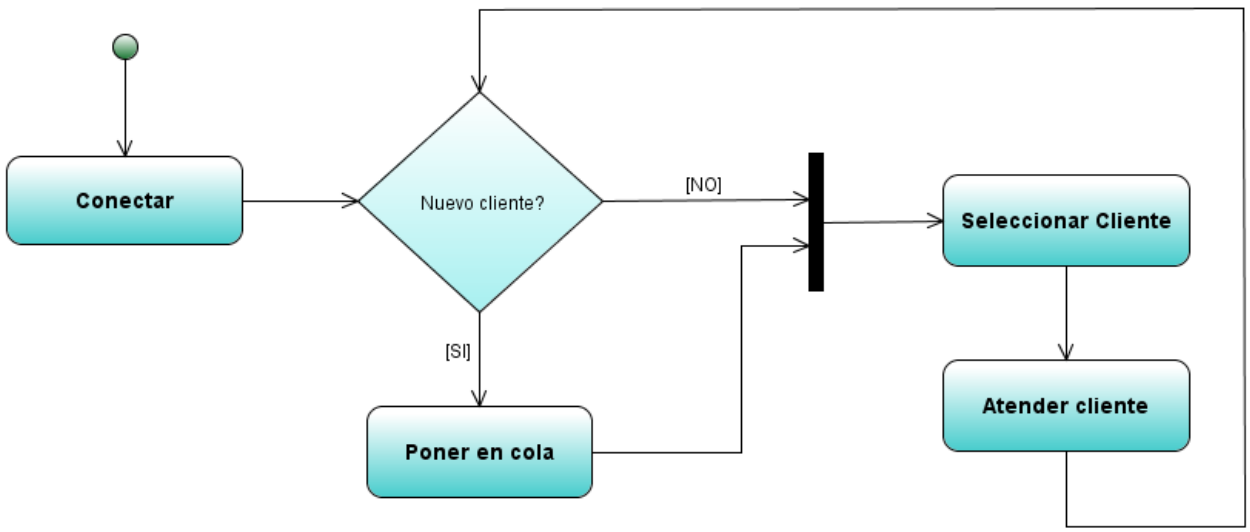


Figura 5.7: Diagrama de flujo de la implementación.

Sockets L2CAP no bloqueantes

Para implementar estas ideas, la herramienta básica es el uso de sockets no bloqueantes. A continuación se muestra cómo configurarlos de esta manera:

```
....
 1.- s = socket(AF_BLUETOOTH, SOCK_SEQPACKET, BTPROTO_L2CAP);
 //Puerto 0x1001 del primer adaptador Bluetooth disponible
 2.- loc_addr.l2_family = AF_BLUETOOTH;
 3.- loc_addr.l2_bdaddr = *BDADDR_ANY;
 4.- loc_addr.l2_psm = htobs(0x1001);
 //Ponemos el socket creado s en modo no bloqueante
 5.- sock_flags = fcntl (s, F_GETFL, 0);
 6.- fcntl (s, F_SETFL, sock_flags | O_NONBLOCK );
 7.- bind(s, (struct sockaddr *)&loc_addr, sizeof(loc_addr));
 8.- listen(s,maxClientes);
          ....
     int clientActual;
     clientActual=accept(s,(struct sockaddr*)&rem_addr,&opt);
          ....
```

Las líneas interesantes de este código son la 5 que obtiene los *flags* del descriptor del socket, y la 6 que les aplica las máscaras adecuadas para ponerlos en modo no bloqueante. Así, la función `accept(...)` no queda bloqueada a la espera de un nuevo cliente, sino que se continúa con la ejecución. En cualquier caso, siempre que se produzca una nueva conexión por parte de una estación, se añadirá a la lista de clientes libres con la que se gestiona todas las conexiones que haya.

Una vez aceptada o no una nueva conexión, siempre se avanza de manera que se exploren todos los nodos que se estén tratando, mediante un bucle de encuesta.

```
...
```

```
 while(cond_parada==0){
    ...
    if(numClientes>0){
       FD_ZERO(&readfds);
       FD_ZERO(&writefds);
       int k;
       for (k=0;k<maxClientes;k++){
          if (!clientLibres[k])
             FD_SET(client[k],&readfds);
          maxfd=client[0];
          for (k=1;k<maxClientes;k++){
             if (client[k]>maxfd); maxfd=client[k];
          }
       status = select(maxfd +1, &readfds, &writefds, NULL,NULL);
       //PROCESAR : Aquí se hace la encuesta y se actúa en consecuencia
       }
       ...
    }
  ...
}
```

Se acaba de mostrar un ejemplo extraído del código de nuestra implementación. En él se observa que `client` es el vector de nodos, y `clientLibres` es el vector auxiliar para marcar las posiciones ocupadas por los nodos conectados. A cada nodo conectado se le asocia el evento que puede activar, y con la función `select(...)` se está a la espera de que se produzca algún evento[4].

El último parámetro de `select(...,NULL)` establece el timeout que el nodo usará para consultar eventos. Tal y como está, la espera es indefinida hasta que se produzca alguno, pero puede configurarse a elección, como se muestra en el siguiente ejemplo (aquí el timeout será de 5 segundos):

```
struct timeval timeout;
timeout.tv_sec = 5;
timeout.tv_usec = 0;
status = select(maxfd +1, &readfds, &writefds, NULL,&timeout);
```

Consumido este timeout, se puede aprovechar para cambiar la funcionalidad y hacer que el nodo trabaje durante cierto tiempo (o cierto número de tramas) generando datos.

Otra opción, posiblemente más efectiva (el timeout puede que no se consuma durante bastante tiempo), es implementar un temporizador que lance la rutina de generación de tráfico si lo hubiere.

[4] Al tratarse de piconets no esperamos que el vector tenga más de 7 nodos. El procesamiento (reenvío o guardar en disco) también se efectúa muy rápido.

Capítulo 6

Manual de Usuario

6.1. Introducción

En este capítulo se explica cómo instalar todas las herramientas asociadas al SHIMMER y su utilización. Además se detalla cómo compilar, instalar y ejecutar el software desarrollado en el **capítulo 4** en el simulador MSPsim para SHIMMER.

Por otra parte se adjuntan instrucciones para comprobar el correcto funcionamiento de la pila de protocolos de Bluetooth en Linux. Finalmente se muestra cómo utilizar las librerías de la API de BlueZ para compilar programas.

6.2. Tecnología SHIMMER

SHIMMER incluye varias herramientas, algunas de ellas son:

- TinyOS: es un sistema operativo dirigido por eventos, adecuado para trabajar con el software diseñado con SHIMMER. Se incluye el entorno de desarollo que permite diseñar, implementar, simular y testear software diseñado con SHIMMER. Se basa en el lenguaje de programación nesC.
- Compilador de nesC.
- Paquete de utilidades MSP430-ToolS.
- Manual de hardware y software.

6.2.1. Creación de un Entorno de Desarrollo de SHIMMER

Es necesario disponer del árbol fuente completo del sistema operativo TinyOS. Dicho árbol está en la carpeta `tinyos-1.x` contenida en el CD de desarrollo de SHIMMER. El entorno de desarrollo SHIMMER / TinyOS es compatible con un gran número de plataformas, sin embargo se ha probado de manera oficial en las siguientes:

- Linux RedHat 9.0
- Windows 2000
- Windows XP

Además de las citadas, nosotros la hemos probado con éxito en las siguientes:

- Ubuntu Linux 8.04
- Ubuntu Linux 9.04
- Ubuntu Linux 9.10
- Debian Linux Lenny
- OpenSuse 11.2

Es necesario añadir las siguientes variables de entorno al entorno de desarrollo de manera permanente incluyendo las siguientes líneas de instrucción dentro del archivo `.bashrc`:

```
export TOSDIR=/<whatever>/tinyos-1.x/tos
export TOSROOT=/<whatever>/tinyos-1.x
export MAKERULES=`ncc -print-tosdir`/../tools/make/Makerules
export TinyOS_NP=none
```

Después de modificar dichas variables de entorno, es necesario cerrar la sesión y volver a abrirla; de otra manera hay que ejecutar las utilidades desde una consola de comandos o shell de la cual se han actualizado su entorno incluyendo la siguiente línea de instrucción: source /.bashrc

Para instalar el entorno de programación nesC en el entorno que se va usar hay que seguir los siguientes pasos:

1. Prerrequisitos: Instalar o tener instalado el paquete Java SDK de Sun (`j2sdk-1_4_2_14-linux-i586.bin`) disponible en la carpeta Linux contenida en el CD de desarrollo de SHIMMER o de manera alternativa descargando la última versión desde la siguiente url:

   http://java.sun.com.

   Se puede usar el tutorial de instalación de Java disponible en esta otra url:

   http://java.sun.com/j2se/1.4.2/install-linux.html.

   Esta instalación estará terminada cuando el fichero binario javac sea visible en el entorno usado.

   ```
   javac
   Usage: javac <options><source files> ............
   ```

2. Localizar el fichero `nesc-1.1.2b.tar.gz` en la carpeta Linux disponible en el CD de Desarrollo de SHIMMER o de manera alternativa descargando la última publicación de nesC desde la dirección

   http://sourceforge.net/projects/nescc.

3. Extraer en cualquier localización familiar del disco el contenido del archivo anterior:

   ```
   tar zxvf nesc-1.1.2b.tar.gz
   cd nesc-1.1.2b
   ```

4. Instalar el paquete (es posible que sea necesario tener los paquetes de construcción esenciales en caso de que el programa automake no esté instalado) introduciendo las siguientes líneas de órdenes por consola:

   ```
   ./configure
   make
   sudo make install
   ```

5. Para poder utilizar nesC con el sistema TinyOS es necesario instalar los paquetes ncc, mig etc. Dichos paquetes se encuentran en las carpetas:

   `tinyos-1.x/tools/src/ncc` (para la versión TinyOS 1.x)

   o bien:

   `tinyos-2.x/tools` (para la versión TinyOS 2.x)

   Para proceder a dicha instalación, introducir las las siguientes líneas de órdenes por consola:

   ```
   cd tinyos-1.x/tools/src/ncc
   ./Bootstrap
   ./configure
   sudo make install
   ```

Por último, es necesario instalar las utilidades de MSP430, para ello es necesario realizar los siguientes pasos:

1. Extraer el archivo `msp430-12Jan2005.tar.gz` contenido en la carpeta Linux disponible en el CD de Desarrollo de SHIMMER e instalar las utilidades en la carpeta `/usr/local/msp430.`

2. En vez de añadir `/usr/local/msp430/bin` al PATH local, es mejor enlazar simbólicamente cada fichero de `/usr/local/msp430/bin` a `/usr/local/bin` que debería de estar en el PATH. La siguiente orden enlaza simbólicamente el archivo `msp430-gcc`:

   ```
   ln -s /usr/local/msp430/bin/msp430-gcc /usr/local/bin/msp430-gcc
   ```

   Realizar la misma acción para todos los archivos contenidos dentro del directorio `/usr/local/msp430/bin` excepto para `msp430-bsl`. Es recomendable usar un script para llevar a cabo dichas operaciones:

   ```
   ln -s /usr/local/msp430/bin/* /usr/local/bin/*
   ```

   A continuación:

   ```
   msp430-gcc
   cd $HOME
   msp430-gcc
   msp430-gcc: no input files
   ```

3. Dicha respuesta de `msp430-gcc` significa que se encuentra ahora en el entorno local.

4. Crear un enlace simbólico desde `/usr/local/bin/msp430-bsl`

   a `/tinyos1.x/contrib/handhelds/tools/src/mspgcc-pybsl/bsl.py` mediante las siguientes líneas de órdenes:

   ```
   sudo ln -s ~/tinyos-1.x/contrib/handhelds
       /tools/src/mspgcc-pybsl/bsl.py /usr/local/bin/msp430-bsl
   ```
   [1]

[1] se trata de una única orden

6.2.2. Prueba del Entorno de Desarrollo en Linux

Para chequear y ver si el entorno está funcionando: Ir a la carpeta de la aplicación Blink:

```
cd ~/tinyos-1.x/contrib/handhelds/apps/Blink
make clean shimmer
```

Se debería ver el compilador ncc compilando los archivos sin errores y al final debería poder leerse por pantalla lo siguiente:

```
compiled Blink to build/shimmer/main.exe
3170 bytes in ROM
44 bytes in RAM
```

El archivo generado `main.exe` es el resultado de la compilación mediante nesC para TinyOS de `Blink.nc`. Se encarga de encender y apagar los leds del SHIMMER a una determinada frecuencia.

6.2.3. Configuración de Desarrollo de TinyOS

Para desarrollar aplicaciones para *Motes* usando TinyOS, hay que comprender la estructura del mismo, y cómo se especifica la plataforma destino para la que se quiere generar el *firmware*. Hay que señalar que el sistema operativo TinyOS no está diseñado específicamente para sensores de la marca SHIMMER, sino para una amplia gama de sensores inalámbricos (Telos, Tmote, MICA ...). La distribución de TinyOS-1.x es la versión concreta que usamos en este proyecto. Si bien no es la más actual (sería la 2.x) sí es más estable, y por eso nuestra elección. Puede ampliarse la información aquí presentada, consultando [Liu05].

Esta estructura de directorios obedece a la siguiente disposición:

- `/apps` contiene la implementación del diverso firmware creado para las diferentes plataformas. Dicho firmware creado en *nesC* y usando las librerías de TinyOS podrá ser compilado para cualquier plataforma aceptada.
- `/doc` contiene la documentación propia del sistema operativo y del lenguaje *nesC*. Además puede contener documentación generada por el propio *nesC* durante el desarrollo de programas[2]
- `/tools` contiene diversos *scripts* para configurar objetivos concretos (plataformas para las que se implementa SHIMMER, MICA ...), para cargar el firmware en dichas plataformas, etc...
- `/tos` es el directorio donde se encuentra la implementación de las diversas librerías que TinyOS proporciona para extender y abstraer la funcionalidad del hardware (implementadas en *nesC* igualmente). Además, los archivos .h (*hardware.h* por ejemplo) con la especificación concreta del hardware (mapeo de puertos, direcciones de registros de control y configuración, etc...) que permiten que se pueda generar firmware ejecutable adaptado a cada implementación.

La otra opción es trabajar directamente sobre el hardware. Para ello, podemos implementar sistemas desarrollados en C, utilizando un *compilador cruzado* para el procesador de *Texas Instruments MSP430*. Es el compilador *MSP430GCC*, que está desarrollado a partir del compilador *gcc* para *Linux*, de *GNU*. Siguiendo esta segunda opción, al no tener las facilidades proporcionadas por el sistema operativo, hay que configurar y programar el harware específico de cada implementación. Para ello hay que estudiar el *datasheet* de cada procesador específico de la familia *MSP430*, y el del *mote* al que da servicio dicho procesador.

[2] nesC dispone de herramientas para generar documentación en formato html

6.3. Compilador Cruzado MSPGCC

El software que hemos desarrollado ha sido implementado directamente en C prescindiendo del sistema operativo TinyOS. Para ello hemos utilizado el compilador cruzado MSPGCC que genera un ejecutable adecuado para la arquitectura del procesador MSP430 incorporado a SHIMMER.

Para compilar un programa hay que ejecutar el fichero `MakeFile` anterior con la orden `make <nombrePrograma>`. Se incluye dicho fichero en **B.1** y también y el programa que permite encender y apagar los leds mediante interrupciones a través de una orden por el puerto serie,

6.4. Utilización del simulador MSPsim

Cada vez que se compila un programa se alamacena su ejecutable asociado en la siguiente ruta:

```
firmware/shimmer/<nombrePrograma>
```

Para lanzarlo, es necesario ejecutar el fichero script `lanza` alojado en la raíz de la estructura de archivos del simulador. Para ello se ejecuta la siguiente orden:

```
sh lanza firmware/shimmer/<nombrePrograma>
```

Una vez lanzado el simulador, aparecerán las ventanas correspondientes y en la consola del sistema la cabecera `MSPSim>`. En dicha cabecera se utiliza la orden `start` para que comience la simulación, y la orden `stop` para pararla. Para salir del simulador, se utiliza la orden `quit`. Una vez iniciada la simulación del programa de encendido y apagado de los leds, se introducen las siguientes órdenes a través de la ventana USART1 Port Output que actúa de monitor para el puerto serie:

Para encender los leds:

```
$LedsON
```

Para apagarlos:

```
$LedsOFF
```

Consultar más detalles en el **apartado 4.2**.

6.5. Prueba de conexión Bluetooth en Linux

Se va a probar el funcionamiento de una conexión Bluetooth en Linux entre dos computadores. Se realizará una prueba a nivel de perfiles de aplicación (ver **figura 6.1**). Esto nos permitirá comprobar de una manera bastante directa, si efectivamente funciona el hardware disponible.

Para la prueba de conexión y envío de datos inicial entre dos dispositivos Bluetooth, utilizaremos el *perfil de aplicación OBEX*[3] (ver **figura 6.1**). Este perfil de aplicación permite enviar un archivo entre dos dispositivos que tengan establecida una conexión Bluetooth. En esta prueba se implementará un cliente en C, que hará una llamada al siguiente programa:

```
ussp-push <mac destino>@9 <nombre_origen> <nombre_destino>
```

[3]Object Exchange

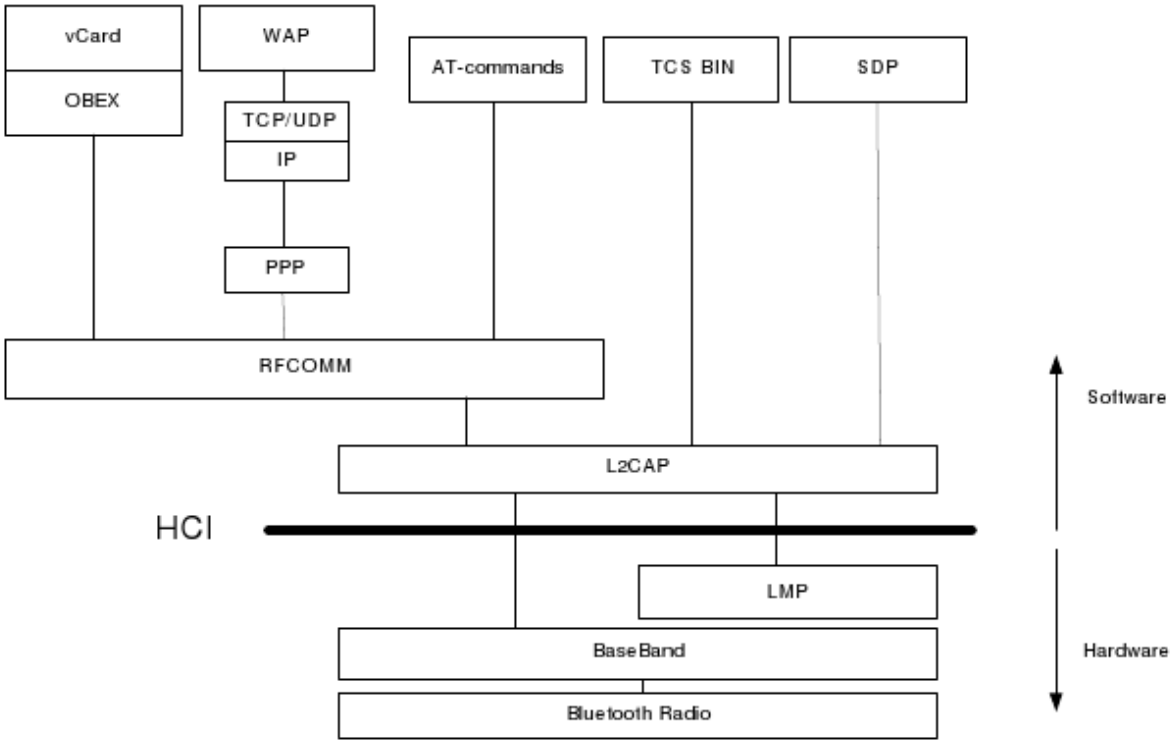


Figura 6.1: Arquitectura de Bluetooth.

Este programa envía datos desde el dispositivo local a un otro. Hay que indicarle la *MAC Bluetooth* del dispositivo destino, el canal que espera utilizar el perfil (en este caso es el 9, por eso `@9`), y hay que indicar también el nombre del archivo que se quiere enviar, y el nombre con el que se recibirá en el destino. Para que funcione correctamente, la prueba implementada (o el comando del programa si se quiere lanzar directamente por consola) se debe ejecutar con permisos de *superusuario* si estamos en computadores Linux. Una vez tratado el cliente, necesitamos un servidor que esté a la escucha para recibir el archivo, y constatar así la corrección de la conexión y el envío. Cualquier dispositivo compatible con el *perfil de aplicación* OBEX (por ejemplo un teléfono móvil con Bluetooth) puede servirnos para tal propósito, y hemos probado con éxito el envío de un archivo a través del cliente de prueba que aquí mencionamos, a teléfonos móviles con Bluetooth. No obstante, para comprobar el funcionamiento entre dos computadores Linux, que es lo que nos interesa, podemos implementar un servidor en C que efectuará la llamada al programa:

```
obexpushd -B
```

Este servidor está a la escucha de conexiones entrantes, aceptándolas y recibiendo archivos a través del perfil *OBEXD*. De nuevo el servidor de prueba (o el comando si se lanza directamente por consola) debe ser ejecutado con permisos de *superusuario* en computadores Linux.

6.6. Configuración de Dispositivo Bluetooth en Linux

El Bluetooth se puede configurar de manera sencilla usando dos aplicaciones integradas dentro de casi todas las distribuciones basadas en Linux:

- `hcitool`
- `hciconfig`

Las principales utilidades que proporciona hcitool son:

- `hcitool [-h]`
- `hcitool [-i <hciX>] [comando [parámetros de comando]]`[4]

donde:

Opciones:

`-h, --help`

Muestra una lista de los comandos disponibles

`-i <dispositivo>`

Este comando se aplica al dispositivo hciX, que debe ser el nombre del dispositivo Bluetooth instalado. Si no se especifica ningún valor de hciX el comando será enviado al primer dispositivo Bluetooth disponible.

Comandos:

`dev`

Muestra los dispositivos Bluetooth locales.

[4] En Linux, los interfaces HCI se nombran *hciX*, donde X es un natural de 0 en adelante y único para cada interfaz.

`scan`

Realiza una consulta de dispositivos remotos. Para cada dispositivo encontrado se muestran el dispositivo y el nombre correspondiente.

Por su parte, hciconfig ofrece las siguientes opciones:

```
hciconfig -h

hciconfig [-a]

hciconfig [-a] [comando [parámetros de comando]]
```

donde:

Opciones:

`-h, --help`

Muestra una lista de los comandos disponibles

`-a, --all`

Además de la información básica muestra las propiedades, el tipo de paquete, la política de enlace, el nombre, la clase y la versión.

Comandos:

`up`

Abre e inicializa un dispositivo HCI.

`down`

Cierra un dispositivo HCI.

`pscan`

Habilita el escaneo de página, deshabilita el escaneo de consulta.

`iscan`

Habilita el escaneo de consulta, deshabilita el escaneo de página.

`piscan`

Habilita los escaneos de página y de consulta.

`name [nombre]`

Fija el nombre del dispositivo local al valor de nombre. Si no se especifica ningún valor muestra el nombre del dispositivo local.

Es muy importante tener en cuenta que para usar los comandos de hciconfig es necesario escribirlos por consola como superusuario del sistema mediante la opción `sudo`.

6.7. Compilación de programas en C utilizando la API de BlueZ

Para compilar el código desarrollado a partir del **apartado 5.3** es necesario tener instalado la API de BlueZ [HH]. Tanto las librerías como el código fuente se pueden obtener a través de la página web oficial o a través a algún repositorio de software.

Se pueden hacer dos tipos de compilaciones: si sólo se cuenta con las librerías ya compiladas, se puede enlazar mediante la siguiente orden:

```
gcc -o <nombreEjecutable> <nombreFichero.c> -lbluetooth
```

Sin embargo, si se cuenta con el código fuente, se puede realizar una compilación estática.
Se adjuntan los siguientes ejemplos que hemos utilizado como escenario de pruebas:

1. Ejemplo con una estación que actúa como fuente de datos, otra como intermediaria y otra como estación base. Es necesario configurar un fichero `path.h` para incluir las rutas del fichero que actúa como origen de los datos en la estación fuente y del fichero en que se guardan los datos recibidos en la estación base. Se adjunta en **B.2**.

2. Ejemplo con una estación base con funcionalidad ampliada capaz de tratar a varios clientes a la vez de forma alterna y trama por trama. Se adjunta en **B.3** con una fuente de datos específica que prescinde de hacer una exploración. De esta forma se conecta directamente con la estación base (se pasa la dirección hardware de esta última mediante argumento en la línea de comandos) para facilitar las pruebas.

Capítulo 7

Conclusiones y Trabajo Futuro

7.1. Programación del puerto serie USART1

Se ha hecho un estudio del funcionamiento del puerto serie del MSP430 para más tarde programar algunos ejemplos de recepción y transmisión de caracteres. El puerto serie recibe y transmite caracteres mediante entrada y salida programada y mediante interrupciones y el MSP430 procesa series de estos caracteres y enciende y apaga los leds en función de estas series. Estos ejemplos, explicados en el **apartado 4.2**, se pueden probar en el simulador MSPsim o adaptar para el el propio SHIMMER y sirven para tomar un primer contacto con la programación para el MSP430. Una vez estudiado algún algoritmo de encaminamiento como el que hemos diseñado e implementado en el **capítulo 5**, se podría adaptar para este tipo de sensores.

7.2. Diseño e implementación de un algoritmo de encaminamiento

En el capítulo 5 se han presentado el diseño y la implementación de un algoritmo de encaminamiento enfocado a redes de sensores móviles. Este algoritmo ha sido diseñado detalladamente y más tarde programado en C a bajo nivel usando la API de Bluetooth BlueZ con el objetivo de dejar abierta la posibilidad de adaptarlo para una red de sensores. Las características más importantes del algoritmo son:

- El algoritmo permite que un sensor cualquiera perteneciente a una red pueda enviar datos a otro sensor cercano y perteneciente a la misma red de forma que estos datos lleguen finalmente a una estación base y ésta los procese adecuadamente.

- Los nodos intervienen en la construcción del camino con una sencilla exploración de dispositivos y no tienen que esperar recepciones de datos adicionales para configurar la ruta. De esta forma se espera que el algoritmo sea eficiente y tenga el menor consumo posible.

- Mediante el sistema de confirmaciones basado en HDLC se implementa un control de flujo tal que no se procede a enviar la siguiente trama de un conjunto de datos hasta que el destinatario ha confirmado que está listo para recibirla. Esto previene posibles errores derivados del envío de la transmisión de información a un nodo que no esté en ese momento disponible. También ayuda al bajo consumo de energía, ya que se previene el trabajo no efectivo.

- La alta fiabilidad que tiene el algoritmo, ya que sigue funcionando a pesar de que haya fallos en nodos o se produzcan movimientos que impliquen cambios en la topología de la red. Las exploraciones periódicas y su consecuente renombramiento dinámico hacen esto posible.

7.2.1. Trabajo Futuro

Se podría adaptar este algoritmo para que trabaje directamente en una red de sensores. Para ello, es posible que no se pudiera usar la pila de protocolos BlueZ y habría que programarlo directamente en el entorno correspondiente.

Si se cuenta con la infraestructura necesaria para probar el funcionamiento de este algoritmo con un número de sensores adecuado, se podrán tomar las medidas pertinentes de consumo que lleven a fijar los tiempos que se adjudiquen al *timeout* del que hemos hablado en el **apartado 5.3.4**. De igual forma se podría medir cada cuanto tiempo es conveniente generar la interrupción de cambio de funcionalidad de los clientes intermediarios.

La tecnología Bluetooth no es la única posible a la hora de elegir una interfaz de comunicación apropiada para este algoritmo. Existen otras opciones como por ejemplo ZigBee, basada en el estándar IEEE 802.15.4, que favorece el bajo consumo en las comunicaciones inalámbricas. Así pues, se podrá adaptar este algoritmo a una red formada por sensores SHIMMER, ya que cuentan con dos tipos de interfaces, la de tipo Bluetooth y la de tipo ZigBee.

Además, es conveniente encontrar una solución al problema de cuenta a infinito, ya que de momento, al presentarse las situaciones citadas en el **apartado 5.2.1**, nos encontramos con el problema de que pueden quedar sensores aislados en situaciones en las que no deberían.

Apéndice A

Glosario de Términos y Acrónimos

- **Acelerómetro:** Cualquier instrumento destinado a medir aceleraciones.
- **ACK:** ACKNOWLEDGEMENT (Acuse de Recibo). En comunicaciones entre computadores, mensaje que se envía para confirmar que un mensaje o un conjunto de mensajes han llegado. Si el terminal de destino tiene capacidad para detectar errores, el significado de ACK es "ha llegado y además ha llegado correctamente".
- **A/D**, **ADC:** Analogue to Digital (Analógico-Digital), Analogue to Digital Converter (Conversor Analógico-Digital). Transcripción de señales analógicas en señales digitales, con el propósito de facilitar su procesamiento (codificación, compresión, etc.) y hacer la señal resultante (la digital) más inmune al ruido y otras interferencias a las que son más sensibles las señales analógicas.
- **API:** Application Programming Interface (Interfaz de Programación de Aplicaciones). Conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. Usados generalmente en las bibliotecas.
- **BioMOBIUS:** Plataforma de investigación desarrollada por TRIL Centre, basada en tecnología abierta y compartida que permite al investigador un desarrollo rápido y sofisticado de soluciones tecnológicas para la investigación biomédica y desarrollada bajo la filosofía de proveer una plataforma de tecnología común que integra hardware, software, servicios y sensores.
- **Bluetooth:** Especificación industrial para Redes Inalámbricas de Área Personal (WPAN) basada en el estándar IEEE 802.15.1 que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda ISM de los 2,5 GHz.
- **BlueZ:** Pila Bluetooth oficial de Linux. Su meta es lograr una implementación de los estándares inalámbricos Bluetooth para Linux. En 2006, la pila soporta todos los protocolos y niveles de la especificación de base. Está disponible a partir de la versión 2.4.6 del núcleo.
- **CPU:** Central Processing Unit (Unidad de Central de Procesamiento). Componente de un ordenador, que interpreta las instrucciones y procesa los datos contenidos en los programas de la computadora. Las CPU proporcionan la característica fundamental de la computadora digital (la programabilidad) y son uno de los componentes necesarios encontrados en las computadoras de cualquier tiempo, junto con el almacenamiento primario y los dispositivos de entrada / salida. Se conoce como microprocesador el CPU que es manufacturado con circuitos integrados.
- **ECG:** Electrocardiograma (ECG/EKG, del alemán Elektrokardiogramm). Representación gráfica de la actividad eléctrica del corazón, que se obtiene con un electrocardiógrafo en forma de cinta continua.

Es el instrumento principal de la electrofisiología cardíaca y tiene una función relevante en el cribado y diagnóstico de las enfermedades cardiovasculares, alteraciones metabólicas y la predisposición a una muerte súbita cardiaca. También es útil para saber la duración del ciclo cardíaco.

- **EMG:** Electromiografía o Electromiograma. Técnica de diagnóstico médico consistente en un estudio neurofisiológico de la actividad bioeléctrica muscular. Clásicamente, el mismo término EMG engloba también a la electroneurografía (el estudio de los nervios que transmiten la orden motora al aparato muscular) si bien en la actualidad se usa cada vez más en este sentido la palabra electroneuromiografía (ENMG). La técnica consiste en la aplicación de pequeños electrodos de bajo voltaje en forma de agujas en el territorio muscular que se desea estudiar, midiendo la respuesta y la conectividad entre los diferentes electrodos.

- **E/S:** Entrada / Salida (también abreviado E/S o I/O del original en inglés Input / Output). En computación es la colección de interfaces que usan las distintas unidades funcionales (subsistemas) de un sistema de procesamiento de información para comunicarse unas con otras, o las señales (información) enviadas a través de esas interfaces. Las entradas son las señales recibidas por la unidad, mientras que las salidas son las señales enviadas por ésta. El término puede ser usado para describir una acción; "realizar una entrada/salida"se refiere a ejecutar una operación de entrada o de salida. Los dispositivos de E/S los usa una persona u otro sistema para comunicarse con una computadora. De hecho, a los teclados y ratones se los considera dispositivos de entrada de una computadora, mientras que los monitores e impresoras son vistos como dispositivos de salida de una computadora. Los dispositivos típicos para la comunicación entre computadoras realizan las dos operaciones, tanto entrada como salida, y entre otros se encuentran los módems y tarjetas de red.

- **Ethernet:** Estándar de redes de computadoras de área local con acceso al medio por contienda CSMA / CD cuyo nombre viene del concepto físico de ether (éter) y que forma la base para la redacción del estándar internacional IEEE 802.3. Define las características de cableado y señalización de nivel físico y los formatos de tramas de datos del nivel de enlace de datos del modelo OSI.

- **FAT:** File Allocation Table (Tabla de Asignación de Archivos). Sistema de archivos desarrollado para MS-DOS, así como el sistema de archivos principal de las ediciones no empresariales de Microsoft Windows hasta Windows Me. Se utiliza en disquetes, para entornos multiarranque y tarjetas de memoria.

- **Firmware:** (Programación en firme) Bloque de instrucciones de programa para propósitos específicos, grabado en una memoria de tipo no volátil (ROM, EEPROM, flash,...), que establece la lógica de más bajo nivel que controla los circuitos electrónicos de un dispositivo de cualquier tipo. Al estar integrado en la electrónica del dispositivo es en parte hardware, pero también es software, ya que proporciona lógica y se dispone en algún tipo de lenguaje de programación. Funcionalmente, el firmware es el intermediario (interfaz) entre las órdenes externas que recibe el dispositivo y su electrónica, encargado de controlar a ésta última para ejecutar correctamente dichas órdenes externas.

- **Flash:** Manera desarrollada de la memoria EEPROM que permite que múltiples posiciones de memoria sean escritas o borradas en una misma operación de programación mediante impulsos eléctricos, frente a las anteriores que sólo permite escribir o borrar una única celda cada vez.

- **SHIMMER:** Sensing Health with Intelligence, Modularity, Mobility and Experimental Reusability (Sondeando la Salud con Inteligencia, Modularidad, Movilidad y Reutilización Experimental)

- **GSR:** Galvanic Skin Response (Respuesta Galvánica de la Piel). Método de medida de la resistencia eléctrica de la piel vinculado a la investigación de la actividad electrodermal relacionado principalmente con las fluctuaciones espontáneas.

- **GNU / Linux:** Uno de los términos empleados para referirse a la combinación del núcleo o kernel libre similar a Unix denominado Linux, que es usado con herramientas de sistema GNU. Su desarrollo es uno

de los ejemplos más prominentes de software libre; todo su código fuente puede ser utilizado, modificado y redistribuido libremente por cualquiera bajo los términos de la GPL (Licencia Pública General de GNU) y otra serie de licencias libres.

- **Giróscopo:** Dispositivo mecánico formado esencialmente por un cuerpo con simetría de rotación que gira alrededor de su eje de simetría. Cuando se somete el giroscopio a un momento de fuerza que tiende a cambiar la orientación del eje de rotación su comportamiento es aparentemente paradójico ya que el eje de rotación, en lugar de cambiar de dirección como lo haría un cuerpo que no girase, cambia de orientación en una dirección perpendicular a la dirección "intuitiva".

- **HDLC:** High-Level Data Link Control (Control de Enlace de Datos de Alto Nivel). Protocolo de comunicaciones de propósito general punto a punto, que opera a nivel de enlace de datos, basado en ISO 3309 e ISO 4335. Surge como una evolución del anterior SDLC y proporciona recuperación de errores en caso de pérdida de paquetes de datos, fallos de secuencia y otros, por lo que ofrece una comunicación confiable entre el transmisor y el receptor.

- **IRP:** InfraRed Pasive (InfrarRojos Pasivo). El Sensor de Infrarrojos Pasivo es un dispositivo electrónico capaz de medir la radiación electromagnética infrarroja de los cuerpos en su campo de visión formado únicamente por el fototransistor con el cometido de medir las radiaciones provenientes de los objetos.

- **IP:** Internet Protocol (Protocolo de Internet). Protocolo no orientado a conexión usado tanto por el origen como por el destino para la comunicación de datos a través de una red de paquetes conmutados.

- **LED:** Light-Emitting Diode (Diodo Emisor de Luz). Dispositivo semiconductor (diodo) que emite luz incoherente de espectro reducido cuando se polariza de forma directa la unión PN del mismo y circula por él una corriente eléctrica. Este fenómeno es una forma de electroluminiscencia. El color depende del material semiconductor empleado en la construcción del diodo y puede variar desde el ultravioleta, pasando por el visible, hasta el infrarrojo.

- **Li-ion:** Lithium-ion battery (Batería de Iones de Litio). Dispositivo diseñado para almacenamiento de energía eléctrica que emplea como electrolito una sal de litio que procura los iones necesarios para la reacción electroquímica reversible que tiene lugar entre el cátodo y el ánodo.

- **L2CAP:** Logical Link Control and Adaptation Protocol (Protocolo de Control y Adaptación del Enlace Lógico). Protocolo utilizado dentro de la pila de protocolos de Bluetooth para pasar paquetes con y sin orientación a la conexión a sus capas superiores incluyendo tanto al Host Controller Interface (HCI) como directamente al gestor del enlace.

- **MAC**, **MAC Address:** Media Access Control (Control de Acceso al Medio), Media Access Control Address (Dirección de Control de Acceso al Medio). Identificador de 48 bits (6 bloques hexadecimales) que corresponde de forma única a una ethernet de red, también conocido como la dirección física en cuanto identificar dispositivos de red. Es individual, cada dispositivo tiene su propia dirección MAC determinada y configurada por el IEEE.

- **MEMS:** Sistemas Microelectromecánicos (Microelectromechanical Systems, MEMS) se refieren a la tecnología electromecánica, micrométrica y sus productos, y a escalas relativamente más pequeñas (escala nanométrica) se fusionan en sistemas nanoelectromecánicos (Nanoelectromechanical Systems, NEMS) y Nanotecnología.

- **MICA:** Módulo de nodos sensores de segunda generación usado para la investigación y el desarrollo de redes de sensores inalámbricos de bajo consumo desarrollado por el grupo de investigación de la Universidad Berkley de California.

- **MSP430:** Familia de microcontroladores producidos por Texas Instruments. Construido con una CPU de 16 bits, el MSP430 está diseñado para aplicaciones empotradas de bajo costo y bajo consumo de energía.
- **MUX:** Multiplexor. Dispositivo que puede recibir varias entradas y transmitirlas por un medio de transmisión compartido. Para ello divide el medio de transmisión en múltiples canales, para que varios nodos puedan comunicarse al mismo tiempo.
- **nesC:** network embedded systems C (sistemas empotrados de red en C). Dialecto del lenguaje de programación C optimizado para las limitaciones de memoria de las redes de sensores. Existen varias herramientas que completan y facilitan su uso, escritas en su mayoría en Java y en Bash. Otras herramientas y librerías asociadas están escritas principalmente en C.
- **NAND:** Not-AND (No-Y). Puerta lógica que realiza la operación de producto lógico negado.
- **OBEX:** OBject EXchange (Intercambio de datos, también denominado Infra Red OBEX). Protocolo de comunicaciones que facilita el intercambio de objetos binarios entre dispositivos una de cuyas primeras aplicaciones populares de OBEX tuvo lugar en la PDA Palm III que junto con sus múltiples sucesoras utilizaron OBEX para intercambiar tarjetas de negocio, datos e incluso aplicaciones.
- **RFCOMM:** Radio Frequency COMMunication (Comunicación por Radio Frecuencia). Conjunto simple de protocolos de transporte, basado en el estándar ETSI TS 07.10. y construido sobre el protocolo L2CAP que proporciona sesenta conexiones simultáneas para dispositivos bluetooth emulando puertos serie RS-232.
- **RAM:** Random Access Memory (Memoria de Acceso Aleatorio) Memoria desde donde el procesador recibe las instrucciones y guarda los resultados. Es el área de trabajo para la mayor parte del software de un computador.
- **SD**, **MicroSD:** Secure Digital, Micro Secure Digital. Formato de tarjeta de memoria que se utiliza en dispositivos portátiles tales como cámaras fotográficas digitales, PDAs, teléfonos móviles e incluso videoconsolas, entre muchos otros. Estas tarjetas tienen unas dimensiones de 32 mm x 24 mm x 2,1 mm. Existen dos tipos: unos que funcionan a velocidades normales, y otros de alta velocidad que tienen tasas de transferencia de datos más altas.
- **Socket:** Concepto abstracto por el cual dos programas (posiblemente situados en computadoras distintas) pueden intercambiar cualquier flujo de datos de manera fiable y ordenada. Queda definido por una dirección IP, un protocolo de transporte y un número de puerto.
- **TelosB:** Módulo de sensores inalámbricos de bajo consumo y código abierto desarrollado por la empresa Crossbow diseñado para la realización de experimentos vanguardistas por la comunidad investigadora.
- **Timestamp:** Secuencia de caracteres, que denotan la hora y fecha (o alguna de ellas) en la cual ocurrió determinado evento. Esta información es comúnmente presentada en un formato consistente, lo que permite la fácil comparación entre dos diferentes registros y seguimiento de progresos a través del tiempo.
- **TinyOS:** Sistema operativo de código abierto basado en componentes para redes de sensores inalámbricas. Está escrito en el lenguaje de programación nesC como un conjunto de tareas y procesos que colaboran entre sí. Está diseñado para incorporar nuevas innovaciones rápidamente y para funcionar bajo las importantes restricciones de memoria que se dan en las redes de sensores.
- **Tmote Sky:** Plataforma de nodos sensores diseñada para aplicaciones de redes de sensores con altas tasas de datos y un consumo extremadamente bajo, basada en el microcontrolador MSP430 F1611 (procesador RISC de 16 bits).

- **TCP/IP:** Transmission Control Protocol (Protocolo de Control de Transmisión) e Internet Protocol (Protocolo de Internet). Conjunto de protocolos de red en los que se basa Internet y que permiten la transmisión de datos entre redes de computadoras. En ocasiones se le denomina conjunto de protocolos TCP/IP, en referencia a los dos protocolos más importantes que la componen por cuestiones históricas.
- **UART**, **USART:** Universal Asynchronous Receiver-Transmitter (Transmisor-Receptor Asíncrono Universal), Universal Synchronous / Asynchronous Receiver-Transmitter (Transmisor-Receptor Síncrono / Asíncrono Universal). Controla los puertos y dispositivos serie y está integrado en la placa base o en la tarjeta adaptadora del dispositivo. Sirve principalmente para manejar las interrupciones de los dispositivos conectados al puerto serie y convertir los datos en formato paralelo, transmitidos al bus de sistema, a datos en formato serie, para que puedan ser transmitidos a través de los puertos y viceversa.
- **USB:** Universal Serial Bus (Bus Universal en Serie). Puerto que sirve para conectar periféricos a una computadora, creado en 1996 por siete empresas: IBM, Intel, Northern Telecom, Compaq, Microsoft, Digital Equipment Corporation y NEC.
- **ZigBee:** Especificación de un conjunto de protocolos de alto nivel de comunicación inalámbrica para su utilización con radios digitales de bajo consumo, basada en el estándar IEEE 802.15.4 de redes inalámbricas de área personal (wireless personal area network, WPAN). Su objetivo son las aplicaciones que requieren comunicaciones seguras con baja tasa de envío de datos y maximización de la vida útil de sus baterías.

Apéndice B

Listado de Código del Software Implementado

B.1. Programación Puerto Serie

Se adjunta el archivo MakeFile que permite compilar el software desarrollado para la programación del puerto serie USART1 del SHIMMER:

```
CC = msp430-gcc
OPTIONS = -O2 -Wall -g
SOURCES = bluetooth.c
SOURCES2 = bluetooth_int.c
SOURCES3 = bluetooth_int2.c
SOURCES4 = blink_int.c
MCU = -mmcu=msp430x1611
OBJC = msp430-objcopy
OBJD = msp430-objdump
.PHONY: clean
bluetooth: $(SOURCES)
        $(CC) $(MCU) $(OPTIONS) -c -o bluetooth.o bluetooth.c
        $(CC) $(MCU) -o bluetooth.elf bluetooth.o
        $(OBJC) -O ihex bluetooth.elf bluetooth.a43
        $(OBJD) -dSt bluetooth.elf >bluetooth.lst

bluetooth_int : $(SOURCES2)
        $(CC) $(MCU) $(OPTIONS) -c -o bluetooth_int.o bluetooth_int.c
        $(CC) $(MCU) -o bluetooth_int.elf bluetooth_int.o
        $(OBJC) -O ihex bluetooth_int.elf bluetooth_int.a43
        $(OBJD) -dSt bluetooth_int.elf >bluetooth_int.lst

bluetooth_int2 : $(SOURCES3)
        $(CC) $(MCU) $(OPTIONS) -c -o bluetooth_int2.o bluetooth_int2.c
        $(CC) $(MCU) -o bluetooth_int2.elf bluetooth_int2.o
        $(OBJC) -O ihex bluetooth_int2.elf bluetooth_int2.a43
        $(OBJD) -dSt bluetooth_int2.elf >bluetooth_int2.lst
```

```
bluetooth_int3 : $(SOURCES3)
        $(CC) $(MCU) $(OPTIONS) -c -o bluetooth_int3.o bluetooth_int3.c
        $(CC) $(MCU) -o bluetooth_int3.elf bluetooth_int3.o
        $(OBJC) -O ihex bluetooth_int3.elf bluetooth_int3.a43
        $(OBJD) -dSt bluetooth_int3.elf >bluetooth_int3.lst

blink_int : $(SOURCES4)
        $(CC) $(MCU) $(OPTIONS) -c -o blink_int.o blink_int.c
        $(CC) $(MCU) -o blink_int.elf blink_int.o
        $(OBJC) -O ihex blink_int.elf blink_int.a43
        $(OBJD) -dSt blink_int.elf >blink_int.lst

clean:
        rm -f *.o *.elf *.lst *.a43
```

A continuación se incluye el código del programa que permite apagar y encender los leds mediante comandos enviados a través del puerto serie:

```
/* Primer programa que usa USART1 en modo UART para comunicarse
   con el chip RovingNetworks (bluetooth) */
#include <msp430x16x.h>

/* Archivo cabecera para el tratamiento de interrupciones */
#include <signal.h>

char buffer[128];
int ppio = 0;
int final = 0;


/******************** Declaración de funciones ************************************/

void delay (unsigned long int d);
void usart1_init ();
void transmision (void);
void recibirOrden (void);
interrupt (USART1TX_VECTOR) usart1TxIsr (void);
interrupt (USART1RX_VECTOR) usart1RxIsr (void);


/********************** Implementación de las funciones ***************************/

void mdelay (unsigned int d) // No se utiliza
{
    for (; d>0; d--)
    {
        TACTL = TACLR + TASSEL1; /* Detiene el temporizador */
        TACCTL0 = SCS; /* Captura sincrona */
        TACCR0 = 8000; /* Inicia el temporizador para 1 ms */
        TACTL = MC0; /* Programa Timer_A en modo Up Mode */
```

```
        /* Tiempo de espera hasta que se fije TACCR0 CCIFG */
        while ((TACCTL0 & 1) != 1)
            nop();
        TACCTL0 ^= CCIFG; /* Puesta a cero de CCIFG */
    }
    TACTL = TACLR + TASSEL1; /* Detiene el temporizador */
}

/* Lazo de retardo mediante Fuerza Bruta */
void delay (unsigned long int d) // No se utiliza
{
    for (; d>0; d--)
    {
        nop ();
        nop ();
    }
}

/* Inicialización de USART1 en modo UART */
void usart1_init ()
{
    U1CTL = SWRST; // Bit de software reset
    U1BR1 = 0; /* Para 115000 baudios, suponiendo BRCLK = 1 MHz (ver página 13.16) */
    U1BR0 = 0x09;
    U1MCTL = 0x08;
    ME2 |= UTXE1 + URXE1; // Se habilita el módulo USART1 para transmitir y recibir
    U1CTL &= ~SWRST; // Reset a 0

    /* Para poder enviar datos por el bluetooth, además de configurar el USART1
    hay que poner la línea BT_RST (P5.5) en el estado adecuado (para RN-41 el
    reset es activo a baja, por lo que hay que mantener la señal en estado alto
    para el funcionamiento normal del módulo */
    P5OUT |= 0x20;
    IE2 |= UTXIE1 + URXIE1; /* Se habilitan las interrupciones de emisión y
                               recepción de USART */
}

void transmision (void)
{
    buffer[final] = U1RXBUF;
    if (final < 127)
        final++;
    else
        final = 0;
    ppio++;
    U1TXBUF = buffer[ppio - 1];
}

/* Rutina de tratamiento de interrupciones de transmisión de USART1 */
/* Una vez llega y se transmita el primer caracter saltará continuamente hasta que se
```

```
   transmitan los demás caracteres. */
interrupt (USART1TX_VECTOR) enablenested usart1TxIsr (void)
{
    if (ppio != final)
    {
        U1TXBUF = buffer[ppio];
        if (ppio < 127)
            ppio++;
        else
            ppio = 0;
    }
}

/* Rutina de tratamiento de interrupciones de recepción de USART1 */
/* Cada vez que se reciba un carácter diferente de ENTER se almacenan los caracteres
leídos en un buffer.
En el momento en que se reciba un ENTER, se entra en recibirOrden (), que trata
las posibles órdenes que se hayan introducido (si empiezan por '$') y en todo caso
transmite el primer carácter del buffer al buffer de transmisión para que se transmita
y se activará una interrupción de transmisión. */
interrupt (USART1RX_VECTOR) enablenested usart1RxIsr (void)
{
    if (U1RXBUF != '\n')
    {
        buffer[final] = U1RXBUF;
        if (final < 127)
            final++;
        else
            final = 0;
    }
    else
    {
        recibirOrden ();
    }
}

void recibirOrden (void)
{
    if (buffer[ppio] != '$')
        transmision ();

    // LedsON:
    else if ((final ? ppio >= 7) && ((buffer[ppio + 1] == 'L')
            && (buffer[ppio + 2] == 'e')
            && (buffer[ppio + 3] == 'd') && (buffer[ppio + 4] == 's')
            && (buffer[ppio + 5] == 'O') && (buffer[ppio + 6] == 'N')))
    {
        P4OUT = 0x00;
        transmision ();
    }
```

```
    // LedsOFF:
    else if ((final ? ppio >= 8) && ((buffer[ppio + 1] == 'L')
            && (buffer[ppio + 2] == 'e')
            && (buffer[ppio + 3] == 'd') && (buffer[ppio + 4] == 's')
            && (buffer[ppio + 5] == 'O') && (buffer[ppio + 6] == 'F')
            && (buffer[ppio + 7] == 'F')))
    {
        P4OUT = 0x0F;
        transmision ();
    }
    else
    {
        transmision ();
    }
}

int main (void)
{
    // unsigned char a;
    /* Inicialización del Temporizador Maestro a apagado */
    WDTCTL = WDTPW | WDTHOLD;
    TACTL = TACLR + TASSEL1;  /* Detiene el temporizador */

    /* Inicialización de los Puertos de Salida
       a GND, y P4.[0..3] a 0 (LEDs encendidos) */
    P1OUT = 0x00;
    P2OUT = 0x00;
    P3OUT = 0x00;
    P4OUT = 0x0F;
    P5OUT = 0x00;
    P6OUT = 0x00;

    /* No hay módulo de control de E / S */
    P1SEL = 0x00;
    P2SEL = 0x00;
    P3SEL = 0x00;
    P4SEL = 0x00;
    P5SEL = 0x00;
    P6SEL = 0x00;

    /* Seleccionando P3.6 (BT_TXD) como Salida y P3.7 (BT_RXD) como Entrada */
    P1DIR = 0x00;
    P2DIR = 0x00;
    P3DIR = 0x40;
    P4DIR = 0x0f;
    P5DIR = 0x00;
    P6DIR = 0x00;

    /* No hay Interrupciones en los Pines de los Puertos */
```

```
    P1IES = 0x00;
    P2IES = 0x00;
    P1IE  = 0x00;
    P2IE  = 0x00;

    eint (); // Se habilitan las interrupciones globales

    /* Reseteo por software de USART1 */
    usart1_init ();

    // Modo de Reposo a la espera de una interrupción:
    LPM0;
    nop ();
}
```

B.2. Escenario de Pruebas del Algoritmo de Encaminamiento

A continuación se muestra el código de la parte indicada en el ejemplo 1 del apartado 6.7:
Se adjunta el código de la fuente de datos:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <unistd.h>
#include <time.h>
#include <bluetooth/bluetooth.h>
#include <bluetooth/l2cap.h>
#include <bluetooth/hci.h>
#include <bluetooth/hci_lib.h>
#include <sys/ioctl.h>
#include <errno.h>
#include "path.h"

#define pathSize 128
#define headerSize 22
#define tamTipoTrama 1                  // Posición 0 de bufEnviar
#define tamTimeStamp 1                  // Posición 1 de bufEnviar
#define tamMAC 17                       // Posición 2-18 de bufEnviar
#define tamNumSecGlob 1                 // Posición 19
#define tamNumSecLoc 1                  // Posición 20
#define tamNumSecTotales 1              // Posición 21
#define dataSize 490                    // Posición 22-512

struct dispositivos {char nombre[20]; char MAC[8];};
struct pequeno {char nombre[20]; char MAC[18]; int numero;};
struct pequeno peke;
char miMAC[17];
```

```
unsigned char cont8;
unsigned char control[8];
int recibirACK (int s);

void scaneo (void);
void envio (void);
void sacarMAC (void);
int meterInfoCONTROL (void);
int modificar_timeout(char buf[tamMAC], int timeout);

int main (int argc, char **argv)
{
    cont8 = 0;
    memset (control,0,sizeof (control));
    scaneo (); // Realiza un scan y guarda el mas pequeño encontrado en el struct peke
    envio ();
    return 0;
}

void sacarMAC (void)
{
    bdaddr_t bdaddr;
    int to = 100;
    int s;
    int dev_id;
    dev_id = hci_get_route (NULL);
    s = hci_open_dev (dev_id);
    hci_read_bd_addr (s, &bdaddr, to);
    memset (miMAC, 0, sizeof (miMAC));
    ba2str (&bdaddr, miMAC);
}

void envio (void)
{
    int j;
    int Total = headerSize + dataSize;

    /* Numero de partes: Se supone un tamaño maximo de fichero
       128 megabytes --> 128 MB / 512 B = 2e8 --> 8 bits --> 1 B. */
    char ruta[pathSize] = PATH;
    int fd = open (ruta, O_RDONLY);
    int bytes_read;
    int estado;
    int tamArch;
    struct stat buffer;
    estado = fstat (fd, &buffer);
    tamArch = buffer.st_size;

    // Se almacena en dest la MAC de destino.
    struct sockaddr_l2 addr = {0};
```

```
int s, status;
char dest[17];
int k;

for (k = 0; k < 17; k++)
{
    dest[k] = peke.MAC[k];
}
int s_send;

// Se establecen los parámetros de conexión:
addr.l2_family = AF_BLUETOOTH;
addr.l2_psm = htobs (0x1001);
str2ba (dest, &addr.l2_bdaddr); // Para la MAC.

// Conexión al servidor
/* Se crean 2 buffers, uno para leer de fichero en tramas
   de 512 y otro para enviar con la trama completa */
unsigned char buf[dataSize]; // Para los datos de 512 bytes
unsigned char bufEnviar[Total]; // Para los bytes restantes con cabecera incluida.

// Vaciado del buffer:
for (j = 0; j < Total; j++)
{
    bufEnviar[j] = 0;
}

// Se realiza una llamada a función que rellena el primer byte (byte inicial CONTROL):
bufEnviar[0] = meterInfoCONTROL ();

// TimeStamp:
time_t timeStamp = time (0);
int auxTiempo = 0;
auxTiempo = (int) timeStamp;
bufEnviar[tamTipoTrama] = auxTiempo;

// MAC:
sacarMAC ();
int h;
for (h = 0; h < tamMAC; h++)
{
    bufEnviar[tamTipoTrama + tamTimeStamp + h] = miMAC[h];
}

// NumSecGlobal:
int numSecGlob = 0;
bufEnviar[tamTipoTrama + tamTimeStamp + tamMAC] = numSecGlob;

// Division del fichero y envio:
char trozoActual[1];
```

```
trozoActual[0] = 1; // En este caso sería el primer trozo (trozo 1).
bufEnviar[tamTipoTrama + tamTimeStamp + tamMAC + tamNumSecGlob] = trozoActual[0];

unsigned int numTrozosTotal = (tamArch / dataSize) + 1;
unsigned char numTrozos[1];
numTrozos[0] = numTrozosTotal;
bufEnviar[tamTipoTrama + tamTimeStamp + tamMAC + tamNumSecGlob +
          tamNumSecLoc] = numTrozosTotal;

int i, x;
unsigned int aux1 = 1;
int res_write;
s_send = socket (AF_BLUETOOTH, SOCK_SEQPACKET, BTPROTO_L2CAP);
status = connect (s_send, (struct sockaddr *) &addr, sizeof (addr));
while ((bytes_read = read (fd, buf, sizeof buf)) > 0)
{
    for (x = 0; x < dataSize; x++)
    {
        bufEnviar[x + headerSize] = buf[x];
    }
    unsigned int aux2 = numTrozos[0];
    if (status == 0)
    {
        res_write = write (s_send, bufEnviar, bytes_read + headerSize);
        if (res_write == -1)
            perror ("Fallo en el write!");
        printf ("%s %d %s %d%s\n", "Enviando trama", aux1, "de", aux2,".");
        printf (" %s %d %s\n", "Tamaño de trama:", bytes_read + headerSize, ".");
        printf ("  %s %d %s\n", "Contenido en datos de la trama:", bytes_read, ".");
        printf ("   %s %d\n", "Marca de tiempo:", bufEnviar[tamTipoTrama]);
        printf ("    %s %d\n", "Número de secuencia global:", bufEnviar[tamTipoTrama +
                 tamTimeStamp + tamMAC]);
    }
    else
        printf ("%s %d %s %d %s\n", "Conexión fallida: Trama", aux1, "de", aux2,
                "fallida!");

    // Vaciado del buffer
    for (i = 0; i < Total; i++)
    {
        bufEnviar[i] = 0;
    }
    aux1++;

    // Se espera la recepción de ACK:
    int resp = recibirACK (s_send);

    if (resp == 1) // Si el ACK ha sido confirmado de forma esperada, se sigue.
    {
        // Construccion de la cabecera para la siguiente trama:
```

```
    bufEnviar[0] = meterInfoCONTROL ();
    time_t timeStamp = time (0);
    int auxTiempo = 0;
    auxTiempo = (int) timeStamp;
    bufEnviar[tamTipoTrama] = auxTiempo;
    for (h = 0; h < tamMAC; h++)
    {
        bufEnviar[tamTipoTrama + tamTimeStamp + h] = miMAC[h];
    }
    bufEnviar[tamTipoTrama + tamTimeStamp + tamMAC] = numSecGlob;
    bufEnviar[tamTipoTrama + tamTimeStamp + tamMAC + tamNumSecGlob] = aux1;
    bufEnviar[tamTipoTrama + tamTimeStamp + tamMAC + tamNumSecGlob +
              tamNumSecLoc] = numTrozosTotal;
    // sleep (2);
}
/* Si el ACK ha sido confirmado pero hay que esperar, se llama a recibirACK
   para esperar otro ACK para seguir: */
else
{
    if (resp == 2)
    {
        // Espera de 10 segundos:
        close (s_send);
        close (s);
        struct sockaddr_l2 loc_addr = {0}, rem_addr = {0};
        socklen_t opt = sizeof (rem_addr);

        // Se crea el socket
        s = socket (AF_BLUETOOTH, SOCK_SEQPACKET, BTPROTO_L2CAP);

        // Se asocia el puerto 0x1001 del primer adaptador bluetooth disponible
        loc_addr.l2_family = AF_BLUETOOTH;
        loc_addr.l2_bdaddr = *BDADDR_ANY;
        loc_addr.l2_psm = htobs (0x1001);
        int aux = 0;
        aux = bind (s, (struct sockaddr *) &loc_addr, sizeof (loc_addr));
        listen (s, 1);

        // Se acepta una conexión
        s_send = accept (s, (struct sockaddr *) &rem_addr, &opt);
        ba2str (&rem_addr.l2_bdaddr, buf);
        printf ("%s\n", "Esperando la posible respuesta ACK que permita
                 continuar...");
        fprintf (stderr, "Conexión aceptada desde %s\n", buf);

        if (recibirACK (s_send) == 1)
        {
            // Construccion de la cabecera para la siguiente trama:
            memset (bufEnviar, 0, sizeof (bufEnviar));
            bufEnviar[0] = meterInfoCONTROL ();
```

```
                    time_t timeStamp = time (0);
                    int auxTiempo = 0;
                    auxTiempo = (int) timeStamp;
                    bufEnviar[tamTipoTrama] = auxTiempo;
                    for (h = 0; h < tamMAC; h++)
                    {
                        bufEnviar[tamTipoTrama + tamTimeStamp + h] = miMAC[h];
                    }
                    bufEnviar[tamTipoTrama + tamTimeStamp + tamMAC] = numSecGlob;
                    bufEnviar[tamTipoTrama + tamTimeStamp + tamMAC + tamNumSecGlob] = aux1;
                    bufEnviar[tamTipoTrama + tamTimeStamp + tamMAC + tamNumSecGlob +
                              tamNumSecLoc] = numTrozosTotal;
                }
                else
                {
                    printf ("%s\n", "Número de intentos de envio agotado!");
                    exit (-1);
                }
            }
        }
    }
    close (s);
    close (fd);
}

int recibirACK (int s_send)
{
    unsigned char ACK = 0;
    unsigned char aux[1] = {0};
    int nread = read (s_send, aux, sizeof (aux));
    printf ("%d\n", nread);
    if (nread == -1)
    {
        perror ("Fallo al leer la cabecera!");
    }
    ACK = aux[0];
    unsigned char cabecera = ACK >> 6;

    // Solo se guarda el número de sec local de respuesta recibido:
    unsigned int secLocalResp = ACK & 7;
    unsigned int tipoACK = (ACK >> 4) & 3;
    if (cabecera == 2) // Trama de Supervisión.
    {
        if (tipoACK == 0) // Recibido y sigue:
        {
            if (secLocalResp == cont8)
            {
                printf ("%s\n", "La trama enviada ha sido confirmada, se enviará la
                siguiente existente.");
                return 1;
```

```
            }
            else
            {
                printf ("%s %d%s\n", "La trama enviada NO ha sido confirmada, es necesario
                         enviar desde la trama", secLocalResp, ".");
                return -1;
            }
        }
        else if (tipoACK == 1) // Recibido y parada:
        {
            if (secLocalResp == cont8)
            {
                printf ("%s\n", "La trama que se ha enviado ha sido confirmada
                         y se realiza una parada.");
                return 2;
            }
            else
            {
                printf ("%s %d%s\n\n", "La trama enviada NO ha sido confirmada,
                         es necesario enviar desde la trama", secLocalResp,
                        ", y se ha realizado una parada.");
                return -2;
            }
        }
    }
    else
    {
        printf ("%s\n", "La trama recibida no es una trama esperada de supervisión.");
        printf ("%s %d\n", "Se ha recibido:", secLocalResp);
        return -3;
    }
}

int meterInfoCONTROL (void)
{
    /*
        --> Byte de control:    //tamTipoTrama//
            * 1 bit para indicar tipo de control, 0 si Información y 1 ecc.
            * En caso de que sea de información, los 3 bits siguientes para el
              Nº de secuencia para el flujo, el resto no se usan.
            * En caso de que no sea de información, el bit 2 se pone a 0 para indicar
              supervisión, los bits 3 y 4 se usan para codificar el tipo de ACK
              y los bits 6-7-8 para el número de secuencia recibido.
    */

    unsigned char auxiliar = cont8 << 4;
    cont8++;
    cont8 = cont8 % 8;
    return auxiliar;
}
```

```
void scaneo (void)
{
    inquiry_info *ii = NULL;
    int max_rsp, num_rsp;
    int dev_id, sock, len, flags;
    int i;
    max_rsp = 10;
    int pequeEnc = 0;
    struct dispositivos dispos[max_rsp];

    peke.numero = 10;
    int encontrados = 0;
    dev_id = hci_get_route (NULL);
    sock = hci_open_dev (dev_id);
    if (dev_id < 0 || sock < 0)
    {
        perror ("Error de apertura del socket!");
        exit (1);
    }
    len  = 8;

    flags = IREQ_CACHE_FLUSH;
    ii = (inquiry_info*) malloc (max_rsp * sizeof (inquiry_info));

    memset (peke.nombre, 0, sizeof (peke.nombre));
    memset (peke.MAC, 0, sizeof (peke.MAC));

    num_rsp = hci_inquiry (dev_id, len, max_rsp, NULL, &ii, flags);
    if( num_rsp < 0 )
        perror ("Error al realizar el inquiry!");

    if (num_rsp == 0)
    {
        printf ("%s\n", "No se han detectado dispositivos Bluetooth.");
        exit (-1);
    }
    for (i = 0; i < num_rsp; i++)
    {
        ba2str (&(ii+i)->bdaddr, dispos[i].MAC);
        memset (dispos[i].nombre, 0, sizeof (dispos[i].nombre));
        if (hci_read_remote_name (sock, &(ii + i)->bdaddr, sizeof (dispos[i].nombre),
            dispos[i].nombre, 0) < 0)
        {
            strcpy (dispos[i].nombre, "desconocido");
        }
        printf ("%s  %s\n", dispos[i].MAC, dispos[i].nombre);
        int k, x;
        if (pequeEnc == 1)
        {
```

```
            if ((atoi (dispos[i].nombre) < peke.numero) && (atoi (dispos[i].nombre) >= 2))
            {
                peke.numero = atoi (dispos[i].nombre);
                for (k = 0; k < 20; k++)
                    peke.nombre[k] = dispos[i].nombre[k];
                for (x = 0; x < 18; x++)
                    peke.MAC[x] = dispos[i].MAC[x];
            }
        }
        else if ((atoi (dispos[i].nombre) >= 2))
        {
            peke.numero = atoi (dispos[i].nombre);
            for (k = 0; k < 20; k++)
                peke.nombre[k] = dispos[i].nombre[k];
            for (x = 0; x < 18; x++)
                peke.MAC[x] = dispos[i].MAC[x];
            pequeEnc = 1;
        }
        encontrados++;
    }
    if (pequeEnc == 0)
    {
        printf("%s\n","No se han detectado dispositivos Bluetooth disponibles.");
        exit(-1);
    }

    printf ("%s %s %s\n" ,"La menor estación encontrada es:", peke.nombre, peke.MAC);
    int aux;
    char nuevoNombre[20];
    memset (nuevoNombre, 0, sizeof (nuevoNombre));
    aux = atoi (peke.nombre);
    aux = aux + 1;

    // Se transforma el entero a cadena
    sprintf (nuevoNombre, "%d", aux);

    // Función de hci_lib, que renombra al dispositivo bluetooth local
    hci_write_local_name (sock, nuevoNombre, 100);
    printf ("%s %s\n", "Por tanto, el nuevo nombre del dispositivo local
            es:", nuevoNombre);
    free (ii);
    close (sock);
    return;
}
```

Se adjunta el código del nodo intermediario:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
```

```
#include <sys/socket.h>
#include <sys/stat.h>
#include <unistd.h>
#include <time.h>
#include <bluetooth/bluetooth.h>
#include <bluetooth/l2cap.h>
#include <bluetooth/hci.h>
#include <bluetooth/hci_lib.h>
#include "path.h"

#define pathSize 128
#define headerSize 22
#define tamTipoTrama 1
#define tamTimeStamp 1
#define tamMAC 17
#define tamNumSecGlob 1
#define tamNumSecLoc 1
#define tamNumSecTotales 1
#define dataSize 490

int main (int argc, char **argv)
{
    cont8 = 0;
    scaneo ();
    struct sockaddr_l2 loc_addr = {0}, rem_addr = {0}, addr = {0};
    memset (buf, 0, sizeof buf);
    unsigned char bufAux[dataSize] = {0};
    char dirCliente[17] = {0};
    char rutaAux[pathSize] = PATHSERVERAUX;
    char ruta[pathSize] = PATHSERVER;
    unsigned char byte[8];
    int s, client, bytes_read, fd, fdAux;
    int i, j, k,status;
    int cont = 1;
    int numTramas = 1;
    socklen_t opt = sizeof (rem_addr);

    // Vaciado de los buffers
    for (i = 0; i < (dataSize + headerSize); i++)
    {
        buf[i] = 0;
    }

    for (j = 0; j < dataSize; j++)
    {
        bufAux[j] = 0;
    }

    // Se crea el socket
    s = socket (AF_BLUETOOTH, SOCK_SEQPACKET, BTPROTO_L2CAP);
```

```
// Se asocia el puerto 0x1001 del primer adaptador bluetooth disponible
loc_addr.l2_family = AF_BLUETOOTH;
loc_addr.l2_bdaddr = *BDADDR_ANY;
loc_addr.l2_psm = htobs (0x1001);
int aux = 0;
aux = bind (s, (struct sockaddr *) &loc_addr, sizeof (loc_addr));
fd = open (ruta, O_WRONLY | O_TRUNC | O_CREAT, 0666);
fdAux = open (rutaAux, O_WRONLY | O_TRUNC | O_CREAT, 0666);

// Socket en modo escucha, solo se pone 1 cliente en la cola
listen (s, 1);

// Se acepta una conexión
client = accept (s, (struct sockaddr *) &rem_addr, &opt);
ba2str (&rem_addr.l2_bdaddr, dirCliente);
fprintf (stderr, "Conexión aceptada desde %s\n", dirCliente);
str2ba (dirCliente, &addr.l2_bdaddr); // Para la MAC.
memset (buf, 0, sizeof (buf));
memset (bufAux, 0, sizeof (bufAux));

// Se leen datos del cliente
while ((bytes_read = read (client, buf, sizeof (buf))) > 0)
{
    for (k = 0; k < dataSize; k++)
    {
        bufAux[k] = buf[k + headerSize];
    }
    lseek (fd, (dataSize + headerSize) * (buf[tamTipoTrama + tamTimeStamp +
           tamMAC + tamNumSecGlob] - 1), SEEK_SET);
    write (fd, buf, bytes_read);
    lseek (fdAux, (dataSize) * (buf[tamTipoTrama + tamTimeStamp +
           tamMAC + tamNumSecGlob]-1), SEEK_SET);
    write (fdAux, bufAux, bytes_read - headerSize);
    for (i = 0; i < tamMAC; i++)
    {
        miMAC[i] = buf[tamTipoTrama + tamTimeStamp + i];
    }
    tratarPrimerByte ();
    printf ("%s %d %s %d %s %s\n", "Paquete", buf[tamTipoTrama + tamTimeStamp +
             tamMAC + tamNumSecGlob], "de", buf[tamTipoTrama + tamTimeStamp +
             tamMAC + tamNumSecGlob + tamNumSecLoc],
            "recibido correctamente con origen", miMAC);
    printf (" %s %d\n", "Número de secuencia global:", buf[tamTipoTrama +
             tamTimeStamp + tamMAC]);
    printf ("  %s %d\n", "Marca de tiempo:", buf[tamTipoTrama]);
    printf ("   %s %d\n\n", "Tamaño del paquete:", bytes_read);
    cont++;
    cont8 = buf[tamTipoTrama + tamTimeStamp + tamMAC + tamNumSecGlob];
    cont8 = cont8 % 8;
```

```
        if (numTramas == 1)
        {
            pruebaCliente (client);
            close (client);
            close (s);
            reenviarTrama ();
            addr.l2_family = AF_BLUETOOTH;
            addr.l2_psm = htobs (0x1001);
            client = socket (AF_BLUETOOTH, SOCK_SEQPACKET, BTPROTO_L2CAP);
            status = connect (client, (struct sockaddr *) &addr, sizeof (addr));
            if (status == -1)
                perror ("Error al conectar de nuevo     /*
        --> Byte de control:    //tamTipoTrama//
            * 1 bit para indicar tipo de control, 0 si Información y 1 ecc.
            * En caso de que sea de información, los 3 bits siguientes para el
              Nº de secuencia para el flujo, el resto no se usan.
            * En caso de que no sea de información, el bit 2 se pone a 0 para indicar
              supervisión, los bits 3 y 4 se usan para codificar el tipo de ACK
              y los bits 6-7-8 para el número de secuencia recibido.
    */
al cliente");
            responder (client);
            numTramas = 0;
        }
        else
        {
            responder (client);
        }
        numTramas++;
        if (bytes_read < (dataSize + headerSize))
            break;

    }

    // Se cierra la conexión:
    close (fd);
    close (fdAux);
    close (s);
}

void reenviarTrama()
{
    // Se almacena en dest la MAC de destino.
    struct sockaddr_l2 addr = { 0 };
    int s, status;
    char dest[17];
    int k;

    for (k = 0; k < 17; k++)
    {
```

```
    dest[k] = peke.MAC[k];
}
int s_send;

// Se establecen los parámetros de conexión:
addr.l2_family = AF_BLUETOOTH;
addr.l2_psm = htobs (0x1001);
str2ba (dest, &addr.l2_bdaddr);

s_send = socket (AF_BLUETOOTH, SOCK_SEQPACKET, BTPROTO_L2CAP);
status = connect (s_send, (struct sockaddr *) &addr, sizeof (addr));

int res_write = write (s_send, buf, sizeof (buf));
int aux1 = buf[tamTipoTrama + tamTimeStamp + tamMAC + tamNumSecGlob];
int aux2 = buf[tamTipoTrama + tamTimeStamp + tamMAC + tamNumSecGlob + tamNumSecLoc];
if (res_write == -1)
    perror ("Fallo en la escritura!");
printf ("%s %d %s %d%s\n", "Enviando trama", aux1, "de", aux2,".");
printf (" %s %d %s\n", "Tamaño de trama:", sizeof (buf), ".");
printf ("  %s %d %s\n", "Contenido en datos de la trama:", dataSize, ".");
printf ("   %s %d\n", "Marca de tiempo:", buf[tamTipoTrama]);
printf ("    %s %d\n", "Número de secuencia global:", bufEnviar[tamTipoTrama +
         tamTimeStamp + tamMAC]);

int resp = recibirACK (s_send);
if (resp == 2)
{
    close (s_send);
    close (s);
    struct sockaddr_l2 loc_addr = {0}, rem_addr = {0};
    socklen_t opt = sizeof (rem_addr);

    // Se crea el socket
    s = socket (AF_BLUETOOTH, SOCK_SEQPACKET, BTPROTO_L2CAP);

    // Se asocia el puerto 0x1001 del primer adaptador bluetooth disponible
    loc_addr.l2_family = AF_BLUETOOTH;
    loc_addr.l2_bdaddr = *BDADDR_ANY;
    loc_addr.l2_psm = htobs (0x1001);
    int aux = 0;
    aux = bind (s, (struct sockaddr *) &loc_addr, sizeof (loc_addr));
    listen (s, 1);

    // Se acepta una conexión:
    s_send = accept (s, (struct sockaddr *) &rem_addr, &opt);
    ba2str (&rem_addr.l2_bdaddr, buf);
    fprintf (stderr, "Conexión aceptada de nuevo desde %s\n", buf);

    if (recibirACK (s_send) != 1)
    {
```

```
            printf ("%s\n","Número de intentos de envio agotado!");
            exit (-1);
        }
    }

    close (s_send);
    close (s);
}

void sacarMAC (void)
{
    bdaddr_t bdaddr;
    int to = 100;
    int s;
    int dev_id;
    dev_id = hci_get_route (NULL);
    s = hci_open_dev (dev_id);
    hci_read_bd_addr (s, &bdaddr, to);
    memset (miMAC, 0, sizeof (miMAC));
    ba2str (&bdaddr, miMAC);
}

int recibirACK (int s_send)
{
    unsigned char ACK = 0;
    unsigned char aux[1] = {0};
    int nread = read (s_send, aux, sizeof (aux));
    if (nread == -1)
    {
        perror ("Fallo al leer la cabecera!");
    }
    ACK = aux[0];
    printf ("%s %d\n", "La trama ACK recibida es:", ACK);
    unsigned char cabecera = ACK >> 6;
    unsigned int secLocalResp = ACK & 7;
    unsigned int tipoACK = (ACK >> 4) & 3;
    if (cabecera == 2) // Trama de Supervisión.
    {
        if (tipoACK == 0) // Recepcion y continuacion:
        {
            if (secLocalResp == cont8)
            {
                printf ("%s\n", "La trama enviada ha sido confirmada, se enviará
                         la siguiente existente.");
                return 1;
            }
            else
            {
                printf ("%s%d%s\n", "La trama enviada NO ha sido confirmada,
                         es necesario enviar desde la trama ", secLocalResp,".");
```

```
                return -1;
            }
        }
        else if (tipoACK == 1) // Recepcion y parada:
        {
            if (secLocalResp == cont8)
            {
                printf ("%s\n", "La trama que se ha enviado ha sido confirmada
                         y se ha realizado una parada.");
                return 2;
            }
            else
            {
                printf ("%s %d%s\n\n", "La trama enviada NO ha sido confirmada,
                         es necesario enviar desde la trama", secLocalResp,
                        ", y se ha realizado una parada.");
                return -2;
            }
        }
    }
    else
    {
        printf ("%s\n", "La trama recibida no es una trama esperada de supervisión.");
        printf ("%s %d\n", "Se ha recibido:", secLocalResp);
        return -3;
    }
}

void scaneo (void)
{
    inquiry_info *ii = NULL;
    int max_rsp, num_rsp;
    int dev_id, sock, len, flags;
    int i;
    max_rsp = 10;
    int pequeEnc = 0;
    struct dispositivos dispos[max_rsp];

    peke.numero = 10;
    int encontrados = 0;
    dev_id = hci_get_route (NULL);
    sock = hci_open_dev (dev_id);
    if (dev_id < 0 || sock < 0)
    {
        perror ("Error de apertura del socket!");
        exit (1);
    }
    len  = 8;

    flags = IREQ_CACHE_FLUSH;
```

```
ii = (inquiry_info*) malloc (max_rsp * sizeof (inquiry_info));

memset (peke.nombre, 0, sizeof (peke.nombre));
memset (peke.MAC, 0, sizeof (peke.MAC));

num_rsp = hci_inquiry(dev_id, len, max_rsp, NULL, &ii, flags);
if( num_rsp < 0 )
    perror ("Error al realizar el inquiry!");

if (num_rsp == 0)
{
    printf ("%s\n", "No se han detectado dispositivos Bluetooth.");
    exit (-1);
}
for (i = 0; i < num_rsp; i++)
{
    ba2str (&(ii+i)->bdaddr, dispos[i].MAC);
    memset (dispos[i].nombre, 0, sizeof (dispos[i].nombre));
    if (hci_read_remote_name (sock, &(ii + i)->bdaddr,
        sizeof (dispos[i].nombre), dispos[i].nombre, 0) < 0)
    {
        strcpy (dispos[i].nombre, "desconocido");
    }
    printf ("%s  %s\n",  dispos[i].MAC,  dispos[i].nombre);
    int k, x;
    if (pequeEnc == 1)
    {
        if ((atoi (dispos[i].nombre) < peke.numero) &&
            (atoi (dispos[i].nombre) >= 1))
        {
            peke.numero = atoi (dispos[i].nombre);
            for (k = 0; k < 20; k++)
                peke.nombre[k] = dispos[i].nombre[k];
            for (x = 0; x < 18; x++)
                peke.MAC[x] = dispos[i].MAC[x];
        }
    }
    else if ((atoi (dispos[i].nombre) >= 1))
    {
        peke.numero = atoi (dispos[i].nombre);
        for (k = 0; k < 20; k++)
            peke.nombre[k] = dispos[i].nombre[k];
        for (x = 0; x < 18; x++)
            peke.MAC[x] = dispos[i].MAC[x];
        pequeEnc = 1;
    }
    encontrados++;
}
if (pequeEnc == 0)
{
```

```
        printf("%s\n", "No se han detectado dispositivos Bluetooth disponibles.");
        exit(-1);
    }

    printf ("%s %s %s\n" , "La menor estación encontrada es:", peke.nombre, peke.MAC);
    int aux;
    char nuevoNombre[20];
    memset (nuevoNombre, 0, sizeof (nuevoNombre));
    aux = atoi (peke.nombre);
    aux = aux + 1;

    // Se trransforma el entero a cadena
    sprintf (nuevoNombre, "%d", aux);

    // Función de hci_lib, que renombra al dispositivo bluetooth local
    hci_write_local_name (sock, nuevoNombre, 100);
    printf ("%s %s\n", "Por tanto, el nuevo nombre del dispositivo local es:",
             nuevoNombre);
    free (ii);
    close (sock);
    return;
}

void tratarPrimerByte ()
{
    unsigned char primerByte = buf[0];
    unsigned char tipoTrama = {0};
    unsigned char numSecEnv = {0};
    unsigned char sondeo = {0};
    unsigned char sondeoPrint = {0};
    unsigned char numSecRec = {0};
    numSecPrint = 0;
    tipoTrama = primerByte & 128;
    numSecEnv = primerByte & 112;
    sondeo = primerByte & 8;
    numSecRec = primerByte & 7;

    if (tipoTrama == 128)
    {
        printf ("%s\n", "Recibida una trama de control");
        // Se confirma la n, luego se solicita la n + 1
    }

    else
    {
        printf ("%s\n", "Recibida una trama de información");
    }

    if (numSecEnv > 0)
    {
```

```
        numSecPrint = numSecEnv >> 4;
    }

    else
    {
        numSecPrint = 0;
    }
    printf (" %s [%d]\n", "Número de secuencia local:", numSecPrint);

    if (sondeo == 8)
    {
        sondeoPrint = 1;
    }

    else
    {
        sondeoPrint = 0;
    }
    printf ("  %s [%d]\n\n", "Bit de sondeo:", sondeoPrint);
}

void responder(int cliente)
{
    /* Los primeros 2 bits se ponen a 10 para indicar la trama de Supervisión.
      Tipo de ACK (envío y sigo) --> 00 en los bits 3 y 4.) > 0
      Bit de sondeo a 0 porque no se usa.
      Los ultimos 3  bits de confirmacion de número de secuencia local,
      son los mismos que los que se han recibido. */

    unsigned char cabecera = 1;
    int resultado_send = 0;
    respuesta = cabecera << 7;
    respuesta = respuesta | numSecPrint;
    unsigned char aux[1] = {0};
    aux[0] = respuesta + 1;
    resultado_send = write (cliente, aux, sizeof (aux));
    if (resultado_send == -1)
    {
        perror ("Ha fallado el envío de la trama de control!");
    }
}

void pruebaCliente (int cliente)
{
    unsigned char cabecera = 1;
    respuesta = cabecera << 7;
    respuesta = respuesta | numSecPrint;
    respuesta = respuesta | 16;
    unsigned char aux[1] = {0};
    aux[0] = respuesta + 1;
```

```
    int resultado_send = 0;
    printf ("%s [%d]\n", "La trama de ACK + PARA es:", aux[0]);
    resultado_send = write (cliente, aux, sizeof (aux));
}
```

Se adjunta el código de la estación base:

```
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/socket.h>
#include <bluetooth/bluetooth.h>
#include <bluetooth/l2cap.h>
#include "path.h"

#define pathSize 128
#define headerSize 22
#define tamTipoTrama 1
#define tamTimeStamp 1
#define tamMAC 17
#define tamNumSecGlob 1
#define tamNumSecLoc 1
#define tamNumSecTotales 1
#define dataSize 490

char miMAC[17];
unsigned char numSecPrint;
unsigned char respuesta;

void tratarPrimerByte (unsigned char buf[(dataSize + headerSize)]);
void responder (int cliente);
void pruebaCliente (int cliente);

int main (int argc, char **argv)
{
    char rutaAux[pathSize] = PATHSERVERAUX;
    char ruta[pathSize] = PATHSERVER;
    int fd = open (ruta, O_WRONLY | O_TRUNC | O_CREAT, 0666);
    int fdAux = open (rutaAux, O_WRONLY | O_TRUNC | O_CREAT, 0666);
    while (1)
    {
        struct sockaddr_l2 loc_addr = {0}, rem_addr = {0}, addr = {0};
        unsigned char buf[(dataSize + headerSize)] = {0};
        unsigned char bufAux[dataSize] = {0};
        char dirCliente[17] = {0};

        unsigned char byte[8];
        int s, client, bytes_read;
        int i, j, k, status;
        int cont = 1;
        int numTramas = 1;
```

```
socklen_t opt = sizeof (rem_addr);

// Vaciado de los buffers
for (i = 0; i < (dataSize + headerSize); i++)
{
    buf[i] = 0;
}
for (j = 0; j < dataSize; j++)
{
    bufAux[j] = 0;
}

// Se crea el socket
s = socket (AF_BLUETOOTH, SOCK_SEQPACKET, BTPROTO_L2CAP);

// Se asocia el puerto 0x1001 del primer adaptador bluetooth disponible
loc_addr.l2_family = AF_BLUETOOTH;
loc_addr.l2_bdaddr = *BDADDR_ANY;
loc_addr.l2_psm = htobs (0x1001);
int aux = 0;
aux = bind (s, (struct sockaddr *) &loc_addr, sizeof (loc_addr));

// Socket en modo escucha, sólo 1 cliente se pone en la cola
listen (s, 1);

// Se acepta una conexión
client = accept (s, (struct sockaddr *) &rem_addr, &opt);
ba2str (&rem_addr.l2_bdaddr, dirCliente);
fprintf (stderr, "Conexión aceptada desde %s\n\n", dirCliente);
str2ba (dirCliente, &addr.l2_bdaddr); // Para la MAC.
memset (buf, 0, sizeof (buf));
memset (bufAux, 0, sizeof (bufAux));

fd = open (ruta, O_WRONLY | O_APPEND, 0666);
fdAux = open (rutaAux, O_WRONLY | O_APPEND, 0666);

// Se leen datos del cliente
while ((bytes_read = read (client, buf, sizeof (buf))) > 0)
{
    for (k = 0; k < dataSize; k++)
    {
        bufAux[k] = buf[k + headerSize];
    }
    lseek (fd, (dataSize + headerSize) * (buf[tamTipoTrama + tamTimeStamp + tamMAC
           + tamNumSecGlob] - 1), SEEK_SET);
    write (fd, buf, bytes_read);
    lseek (fdAux, (dataSize) * (buf[tamTipoTrama + tamTimeStamp + tamMAC +
           tamNumSecGlob] - 1), SEEK_SET);
    write (fdAux, bufAux, bytes_read - headerSize);
```

```
            for (i = 0; i < tamMAC; i++)
            {
                miMAC[i] = buf[tamTipoTrama + tamTimeStamp + i];
            }
            tratarPrimerByte (buf);
            printf ("%s %d %s %d %s %s\n", "Paquete", buf[tamTipoTrama + tamTimeStamp
                     + tamMAC + tamNumSecGlob],
                    "de", buf[tamTipoTrama + tamTimeStamp + tamMAC + tamNumSecGlob
                     + tamNumSecLoc],
                    "recibido correctamente con origen", miMAC);
            printf (" %s %d\n", "Número de secuencia global:", buf[tamTipoTrama +
                     tamTimeStamp + tamMAC]);
            printf ("  %s %d\n", "Marca de tiempo:", buf[tamTipoTrama]);
            printf ("   %s %d\n\n", "Tamaño del paquete:", bytes_read);
            cont++;

            if (numTramas == 5)
            {
                pruebaCliente (client);
                close (client);
                close (s);
                sleep (5);
                addr.l2_family = AF_BLUETOOTH;
                addr.l2_psm = htobs (0x1001);
                client = socket (AF_BLUETOOTH, SOCK_SEQPACKET, BTPROTO_L2CAP);
                status = connect (client, (struct sockaddr *) &addr, sizeof (addr));
                if (status == -1)
                    perror ("Error al conectar de nuevo al cliente!");
                responder (client);
                numTramas = 0;
            }
            else
            {
                responder (client);
            }
            numTramas++;
            if (bytes_read < (dataSize + headerSize))
                break;
        }

        // Se cierran la conexión y los ficheros:
        close (fd);
        close (fdAux);
        close (s);
        printf ("%s\n\n","Conexión cerrada y a la escucha para posibles recepciones...");
    }
}

void tratarPrimerByte (unsigned char buf[(dataSize + headerSize)])
{
```

```
    unsigned char primerByte = buf[0];
    unsigned char tipoTrama = {0};
    unsigned char numSecEnv = {0};
    unsigned char sondeo = {0};
    unsigned char sondeoPrint = {0};
    unsigned char numSecRec = {0};
    numSecPrint = 0;
    tipoTrama = primerByte & 128;
    numSecEnv = primerByte & 112;
    sondeo = primerByte & 8;
    numSecRec = primerByte & 7;

    if (tipoTrama == 128)
    {
        printf ("%s\n", "Recibida una trama de control");
        // Se confirma la n, luego se solicita la n + 1
    }

    else
    {
        printf ("%s\n", "Recibida una trama de información");
    }

    if (numSecEnv > 0)
    {
        numSecPrint = numSecEnv >> 4;
    }

    else
    {
        numSecPrint = 0;
    }
    printf (" %s [%d]\n", "Número de secuencia local:", numSecPrint);

    if (sondeo == 8)
    {
        sondeoPrint = 1;
    }

    else
    {
        sondeoPrint = 0;
    }
    printf ("  %s [%d]\n\n", "Bit de sondeo:", sondeoPrint);
}

void responder (int cliente)
{
    /* Los primeros 2 bits se ponen a 10 para indicar la trama de Supervisión.
      Tipo de ACK (envío y sigo) --> 00 en los bits 3 y 4.) > 0
```

```
      Bit de sondeo a 0 porque no se usa.
      Los ultimos 3  bits de confirmacion de número de secuencia local,
      son los mismos que los que se han recibido. */

    unsigned char cabecera = 1;
    int resultado_send = 0;
    respuesta = cabecera << 7;
    printf ("%s [%d]\n", "Valor de numSecPrint:", numSecPrint);
    respuesta = respuesta | numSecPrint;
    unsigned char aux[1] = {0};
    aux[0] = respuesta + 1;

    // Envío
    printf ("%s [%d]\n", "Trama de control enviada:", aux[0]);
    resultado_send = write (cliente, aux, sizeof (aux));
    if (resultado_send == -1)
    {
        perror ("Ha fallado el envío de la trama de control!");
    }
}

void pruebaCliente (int cliente)
{
    unsigned char cabecera = 1;
    respuesta = cabecera << 7;
    respuesta = respuesta | numSecPrint;
    respuesta = respuesta | 16;
    unsigned char aux[1] = {0};
    aux[0] = respuesta + 1;
    int resultado_send = 0;
    printf ("%s [%d]\n", "Trama de ACK + Parada:", aux[0]);
    resultado_send = write (cliente, aux, sizeof (aux));
}
```

B.3. Escenario de Pruebas con Funcionalidad Ampliada

A continuación se muestra el código de la parte indicada en el ejemplo 2 del apartado 6.7:
Se adjunta el código de la fuente de datos:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/select.h>
#include <unistd.h>
#include <time.h>
#include <bluetooth/bluetooth.h>
```

```
#include <bluetooth/l2cap.h>
#include <bluetooth/hci.h>
#include <bluetooth/hci_lib.h>
#include <sys/ioctl.h>
#include <errno.h>
#include "path.h"

#define pathSize 128
#define headerSize 22
#define tamTipoTrama 1
#define tamTimeStamp 1
#define tamMAC 17
#define tamNumSecGlob 1
#define tamNumSecLoc 1
#define tamNumSecTotales 1
#define dataSize 490

struct dispositivos {char nombre[20]; char MAC[8];};
struct pequeno {char nombre[20]; char MAC[18]; int numero;};
struct pequeno peke[3];
char miMAC[17];
unsigned char cont8;
unsigned char control[8];
int posicion = 0;
int recibirACK (int s);

int scaneo (void);
int envio (void);
void sacarMAC( void);
int meterInfoCONTROL(void);

int main (int argc, char **argv)
{
    int respuestaScan, respuestaEnv;
    cont8 = 0;
    memset (control, 0, sizeof (control));
    while (1)
    {
        if (scaneo ())
        {
            if (envio () == 1)
                printf ("%s\n", "Envio completado correctamente.");
            else
                printf ("%s\n", "Envío no completado!");
            return 0;
        }
    }
}

void sacarMAC (void)
```

```
{
    bdaddr_t bdaddr;
    int to = 100;
    int s;
    int dev_id;
    dev_id = hci_get_route (NULL);
    s = hci_open_dev (dev_id);
    hci_read_bd_addr (s, &bdaddr, to);
    memset (miMAC, 0, sizeof (miMAC));
    ba2str (&bdaddr, miMAC);
}

int envio (void)
{
    int j;
    int intento = 0;
    int Total = headerSize + dataSize; // 523 = 11 + 512

    // Numero de partes: Se supone un tamaño maximo de fichero
    // 128 megabytes --> 128 MB / 512 B = 2e8 --> 8 bits --> 1 B.
    char ruta[pathSize] = PATH;
    int fd = open (ruta, O_RDONLY);
    int bytes_read;
    int estado;
    int tamArch, maxfd, sock_flags;
    fd_set readfds, writefds;
    struct stat buffer;
    estado = fstat (fd, &buffer);
    tamArch = buffer.st_size;

    // Se almacena en dest la MAC de destino.
    struct sockaddr_l2 addr = {0};
    int s, status;
    char dest[17];
    int k;

    for (k = 0; k < 17; k++)
    {
    dest[k] = peke[0].MAC[k];
    }
    printf ("%s %s  %s\n", "Intento de conexion con", dest, "...");
    int s_send;

    // Establecer los parámetros de conexión:
    addr.l2_family = AF_BLUETOOTH;
    addr.l2_psm = htobs (0x1001);
    str2ba (dest, &addr.l2_bdaddr); // Para la MAC.

    // Conexion con el servidor
    // Se crean 2 buffers, uno para leer de fichero en tramas de 512
```

```
// y otro para enviar con la trama completa:
unsigned char buf[dataSize]; // Para los datos de 512 bytes.
unsigned char bufEnviar[Total]; // Para los bytes restantes con cabecera incluída.

// Vaciado del buffer:
for (j = 0; j < Total; j++)
{
    bufEnviar[j] = 0;
}

// Se llama a función que rellena el primer byte (byte inicial CONTROL):
bufEnviar[0] = meterInfoCONTROL ();

// TimeStamp:
time_t timeStamp = time (0);
int auxTiempo = 0;
auxTiempo = (int) timeStamp;
bufEnviar[tamTipoTrama] = auxTiempo;

// MAC:
sacarMAC ();
int h;
for (h = 0; h < tamMAC; h++)
{
    bufEnviar[tamTipoTrama + tamTimeStamp + h] = miMAC[h];
}
// NumSecGlobal:
int numSecGlob = 0;
bufEnviar[tamTipoTrama + tamTimeStamp + tamMAC] = numSecGlob;

// Division del fichero y envio:
char trozoActual[1];
trozoActual[0] = 1; // En este caso sería el primer trozo (trozo 1).
bufEnviar[tamTipoTrama + tamTimeStamp + tamMAC + tamNumSecGlob] = trozoActual[0];

unsigned int numTrozosTotal = (tamArch / dataSize) + 1;
unsigned char numTrozos[1];
numTrozos[0] = numTrozosTotal;
bufEnviar[tamTipoTrama + tamTimeStamp + tamMAC + tamNumSecGlob +
          tamNumSecLoc] = numTrozosTotal;

int i, x;
unsigned int aux1 = 1;
int res_write;
s_send = socket (AF_BLUETOOTH, SOCK_SEQPACKET, BTPROTO_L2CAP);
sock_flags = fcntl (s_send, F_GETFL, 0);
fcntl (s_send, F_SETFL, sock_flags | O_NONBLOCK );

status = connect (s_send, (struct sockaddr *) &addr, sizeof (addr));
int ok = 1;
```

```
while ((0 != status && errno == EAGAIN) && ok)
{
    intento++;
    if (intento < posicion)
    {
        for (k = 0; k < 17; k++)
        {
            dest[k] = peke[intento].MAC[k];
        }
        printf ("%s %s  %s"," (Primer bucle, intento con", dest, "...");
        str2ba (dest, &addr.l2_bdaddr);
        s_send = socket (AF_BLUETOOTH, SOCK_SEQPACKET, BTPROTO_L2CAP);
        sock_flags = fcntl (s_send, F_GETFL, 0);
        fcntl (s_send, F_SETFL, sock_flags | O_NONBLOCK );
        status = connect (s_send, (struct sockaddr *) &addr, sizeof (addr));
        if (0 != status && errno == EAGAIN)
            ok = 1;
        else
            ok = 0;
    }
    else
        return -1;
}

while ((bytes_read = read (fd, buf, sizeof buf)) > 0)
{
    for (x = 0; x < dataSize; x++)
    {
        bufEnviar[x + headerSize] = buf[x];
    }
    unsigned int aux2 = numTrozos[0];
    FD_ZERO (&readfds);
    FD_ZERO (&writefds);
    FD_SET (s_send, &writefds);
    maxfd = s_send ;
    struct timeval timeout;
    timeout.tv_sec = 5;
    timeout.tv_usec = 0;
    status = select(maxfd + 1, &readfds, &writefds, NULL, &timeout);
    if ((status > 0) && (FD_ISSET (s_send,&writefds)))
    {
        res_write = write (s_send, bufEnviar, bytes_read + headerSize);
        if (res_write == -1)
            perror ("Fallo en la escritura!");
        printf ("%s %d %s %d%s\n", "Enviando trama", aux1, "de", aux2,".");
        printf (" %s %d %s\n", "Tamaño de trama:", bytes_read + headerSize, ".");
        printf ("  %s %d %s\n", "Contenido en datos de la trama:", bytes_read, ".");
        printf ("   %s %d\n", "Marca de tiempo:", bufEnviar[tamTipoTrama]);
        printf ("    %s %d\n", "Número de secuencia global:", bufEnviar[tamTipoTrama
                + tamTimeStamp + tamMAC]);
```

```
}
else
{
    printf ("%s %d %s %d %s\n", "Conexión fallida: Trama ", aux1, " de ", aux2,
            " fallida!");
    close (s_send);
    intento++;
    if (intento < posicion)
    {
        for (k = 0; k < 17; k++)
        {
            dest[k] = peke[intento].MAC[k];
        }
        printf ("%s %s  %s", "Intento con", dest, "...");
        str2ba (dest, &addr.l2_bdaddr);
        s_send = socket (AF_BLUETOOTH, SOCK_SEQPACKET, BTPROTO_L2CAP);
        sock_flags = fcntl (s_send, F_GETFL, 0);
        fcntl (s_send, F_SETFL, sock_flags | O_NONBLOCK );

        status = connect (s_send, (struct sockaddr *) &addr, sizeof (addr));
        ok = 1;
        while ((0 != status && errno == EAGAIN ) && ok)
        {
            intento++;
            if (intento < posicion)
            {
                for (k = 0; k < 17; k++)
                {
                    dest[k] = peke[intento].MAC[k];
                }
                printf ("%s %s  %s", "Intento con ", dest, "...");
                str2ba (dest, &addr.l2_bdaddr);
                s_send = socket (AF_BLUETOOTH, SOCK_SEQPACKET, BTPROTO_L2CAP);
                sock_flags = fcntl (s_send, F_GETFL, 0);
                fcntl (s_send, F_SETFL, sock_flags | O_NONBLOCK );
                status = connect (s_send, (struct sockaddr*) &addr, sizeof (addr));
                if (0 != status && errno == EAGAIN)
                    ok = 1;
                else
                    ok = 0;
            }
            else -1;
        }
    }
    else
        return -1;
}

// Vaciado del buffer
for (i = 0; i < Total; i++)
```

```
{
    bufEnviar[i] = 0;
}
aux1++;

// Se espera la recepción del ACK:
int resp = recibirACK (s_send);
if (resp == 1) // Si el ACK ha sido confirmado de forma esperada, sigue adelante
{
    // Construccion de la cabecera para la siguiente trama:
    bufEnviar[0] = meterInfoCONTROL ();
    time_t timeStamp = time (0);
    int auxTiempo = 0;
    auxTiempo = (int) timeStamp;
    bufEnviar[tamTipoTrama] = auxTiempo;
    for (h = 0; h < tamMAC; h++)
    {
        bufEnviar[tamTipoTrama + tamTimeStamp + h] = miMAC[h];
    }
    bufEnviar[tamTipoTrama + tamTimeStamp + tamMAC] = numSecGlob;
    bufEnviar[tamTipoTrama + tamTimeStamp + tamMAC + tamNumSecGlob] = aux1;
    bufEnviar[tamTipoTrama + tamTimeStamp + tamMAC + tamNumSecGlob
              + tamNumSecLoc] = numTrozosTotal;
}
// Si el ACK ha sido confirmado pero hay que esperar, se
// llama a recibirACK para esperar otro ACK para seguir:
else if (resp == 2)
{
    close (s_send);
    close (s);
    struct sockaddr_l2 loc_addr = {0}, rem_addr = {0};
    socklen_t opt = sizeof (rem_addr);

    // Se crea el socket
    s = socket (AF_BLUETOOTH, SOCK_SEQPACKET, BTPROTO_L2CAP);

    // Se asocia el puerto 0x1001 del primer adaptador bluetooth disponible:
    loc_addr.l2_family = AF_BLUETOOTH;
    loc_addr.l2_bdaddr = *BDADDR_ANY;
    loc_addr.l2_psm = htobs (0x1001);
    int aux = 0;
    aux = bind (s, (struct sockaddr *) &loc_addr, sizeof (loc_addr));
    listen (s, 1);

    // Se acepta una conexión
    s_send = accept (s, (struct sockaddr *) &rem_addr, &opt);
    sock_flags = fcntl (s_send, F_GETFL, 0);
    fcntl (s_send, F_SETFL, sock_flags | O_NONBLOCK );
    ba2str (&rem_addr.l2_bdaddr, buf);
    printf ("%s\n", "Esperando la posible respuesta ACK que permita continuar...");
```

```
            fprintf (stderr, "Conexión aceptada desde %s\n", buf);

            if (recibirACK (s_send) == 1)
            {
                // Construccion de la cabecera para la siguiente trama:
                memset (bufEnviar, 0, sizeof (bufEnviar));
                bufEnviar[0] = meterInfoCONTROL ();
                time_t timeStamp = time (0);
                int auxTiempo = 0;
                auxTiempo = (int) timeStamp;
                bufEnviar[tamTipoTrama] = auxTiempo;
                for (h = 0; h < tamMAC; h++)
                {
                    bufEnviar[tamTipoTrama + tamTimeStamp + h] = miMAC[h];
                }
                bufEnviar[tamTipoTrama + tamTimeStamp + tamMAC] = numSecGlob;
                bufEnviar[tamTipoTrama + tamTimeStamp + tamMAC + tamNumSecGlob] = aux1;
                bufEnviar[tamTipoTrama + tamTimeStamp + tamMAC + tamNumSecGlob
                          + tamNumSecLoc] = numTrozosTotal;
            }
            else
            {
                printf ("%s\n", "Se acabaron los intentos de enviar.");
                printf ("%s\n", "Número de intentos de envio agotado!");
                return -1;
            }
        }
    }
    close (s);
    close (s_send);
    close (fd);
    return 1;
}

int recibirACK (int s_send)
{
    unsigned char ACK = 0;
    unsigned char aux[1] = {0};
    int maxfd, status; // nread;
    fd_set readfds, writefds;
    FD_ZERO (&readfds);
    FD_ZERO (&writefds);
    FD_SET (s_send, &readfds);
    maxfd = s_send ;
    struct timeval timeout;
    timeout.tv_sec = 5;
    timeout.tv_usec = 0;
    status = select (maxfd + 1, &readfds, &writefds, NULL, NULL);
    printf ("%s %d%s", "Status vale", status, ".\n");
    int nread;
```

```
// Se supone que se recibo en ACK. Crear un socket para recibir.
if (status > 0 )
{
    nread = read (s_send, aux, sizeof (aux));
    printf ("%d\n", nread);
}
else
{
    printf ("%s\n", "Tiempo de espera agotado!");
    return -1;
}
if (nread == -1)
{
    perror ("Fallo al leer la cabecera!");
    return -1;
}
ACK = aux[0];
unsigned char cabecera = ACK >> 6;
unsigned int secLocalResp = ACK & 7;
unsigned int tipoACK = (ACK >> 4) & 3;
if (cabecera == 2) // Trama de Supervisión.
{
    if (tipoACK == 0) // Recepcion y sigue:
    {
        if (secLocalResp == cont8)
        {
            printf ("%s\n", "La trama enviada ha sido confirmada, se enviará
                     la siguiente existente.");
            return 1;
        }
        else
        {
            printf ("%s%d%s\n", "La trama enviada NO ha sido confirmada, es necesario
                     enviar desde la trama ", secLocalResp," .");

            // Se situa con un lseek el fichero correspondiente a la posición adecuada.
            return -1;
        }
    }
    else if (tipoACK == 1) // Recepcion y parada
    {
        if (secLocalResp == cont8)
        {
            printf ("%s\n\n", "La trama que se ha enviado ha sido confirmada y se ha
                 realizado una parada.");
            return 2;
        }
        else
        {
```

```
                printf ("%s %d%s\n\n", "La trama enviada NO ha sido confirmada, es
                         necesario enviar desde la trama", secLocalResp,
                        ", y se ha realizado una parada.");
                return -2;
            }
        }
    }
    else
    {
        printf ("%s\n", "La trama recibida no es una trama esperada de supervisión.");
        printf ("%s%d\n", "Se ha recibido: ", secLocalResp);
        return -3;
    }
    return 0;
}

int meterInfoCONTROL (void)
{
    /*
        --> Byte de control:    //tamTipoTrama//
            * 1 bit para indicar tipo de control, 0 si Información y 1 ecc.
            * En caso de que sea de información, los 3 bits siguientes para el
              Nº de secuencia para el flujo, el resto no se usan.
            * En caso de que no sea de información, el bit 2 se pone a 0 para indicar
              supervisión, los bits 3 y 4 se usan para codificar el tipo de ACK
              y los bits 6-7-8 para el número de secuencia recibido.
    */

    unsigned char auxiliar = cont8 << 4;
    cont8++;
    cont8 = cont8 % 8;
    return auxiliar;
}

int scaneo (void)
{
    inquiry_info *ii = NULL;
    int max_rsp, num_rsp;
    int dev_id, sock, len, flags;
    int i;
    max_rsp = 10;
    int pequeEnc = 0;
    struct dispositivos dispos[max_rsp];

    peke[0].numero = 10;
    peke[1].numero = 10;
    peke[2].numero = 10;
    int encontrados = 0;
    dev_id = hci_get_route (NULL);
    sock = hci_open_dev (dev_id);
```

```
if (dev_id < 0 || sock < 0)
{
    perror ("Error de apertura del socket!");
    return -1;
}
len = 8;

flags = IREQ_CACHE_FLUSH;
ii = (inquiry_info*) malloc (max_rsp * sizeof (inquiry_info));

memset (peke[0].nombre, 0, sizeof (peke[0].nombre));
memset (peke[0].MAC, 0, sizeof (peke[0].MAC));
memset (peke[1].nombre, 0, sizeof (peke[1].nombre));
memset (peke[1].MAC, 0, sizeof (peke[1].MAC));
memset (peke[2].nombre, 0, sizeof (peke[2].nombre));
memset (peke[2].MAC, 0, sizeof (peke[2].MAC));

num_rsp = hci_inquiry (dev_id, len, max_rsp, NULL, &ii, flags);
if (num_rsp < 0)
{
    perror ("Error al realizar el inquiry!");
    return -1;
}

if (num_rsp == 0)
{
    printf ("%s\n", "No se han detectado dispositivos Bluetooth.");
    return -1;
}
for (i = 0; i < num_rsp; i++)
{
    ba2str (& (ii + i)->bdaddr, dispos[i].MAC);
    memset (dispos[i].nombre, 0, sizeof (dispos[i].nombre));
    if (hci_read_remote_name (sock, & (ii + i)->bdaddr,
    sizeof (dispos[i].nombre), dispos[i].nombre, 0) < 0)
    {
        strcpy (dispos[i].nombre, "desconocido");
    }
    printf ("%s  %s\n",  dispos[i].MAC,  dispos[i].nombre);
    int k, x;
    if (pequeEnc == 1)
    {
        if ((atoi (dispos[i].nombre) < peke[0].numero) &&
            (atoi (dispos[i].nombre) >= 2))
        {
            posicion = 0;
            peke[posicion].numero = atoi (dispos[i].nombre);
            for (k = 0; k < 20; k++)
                peke[posicion].nombre[k] = dispos[i].nombre[k];
            for (x = 0; x < 18; x++)
```

```
                peke[posicion].MAC[x] = dispos[i].MAC[x];
            posicion++;
        }
        else if ((atoi (dispos[i].nombre) == peke[0].numero) &&
                 (atoi (dispos[i].nombre) >= 2))
        {
            peke[posicion].numero = atoi (dispos[i].nombre);
            for (k = 0; k < 20; k++)
                peke[posicion].nombre[k] = dispos[i].nombre[k];
            for (x = 0; x < 18; x++)
                peke[posicion].MAC[x] = dispos[i].MAC[x];
            posicion++;
        }
    }
    else if ((atoi (dispos[i].nombre) >= 2))
    {
        peke[0].numero = atoi (dispos[i].nombre);
        for (k = 0; k < 20; k++)
            peke[0].nombre[k] = dispos[i].nombre[k];
        for (x = 0; x < 18; x++)
            peke[0].MAC[x] = dispos[i].MAC[x];
        pequeEnc = 1;
        posicion++;
    }
    encontrados++;
}
if (pequeEnc == 0)
{
    printf ("%s\n", "No se han detectado dispositivos Bluetooth disponibles.");
    return -1;
}
printf ("%s\n", "Las menores estaciones encontradas han sido: ");
for (i = 0; i < posicion; i++)
{
    printf ("    %s%s\n", peke[i].nombre, peke[i].MAC);
}
int aux;
char nuevoNombre[20];
memset (nuevoNombre, 0, sizeof (nuevoNombre));
aux = atoi (peke[0].nombre);
aux = aux + 1;

// Transformamos el entero a cadena
sprintf (nuevoNombre, "%d", aux);

// Función de hci_lib, que renombra al dispositivo bluetooth local
hci_write_local_name (sock, nuevoNombre, 100);
printf ("%s %s\n", "Por tanto, el nuevo nombre del dispositivo
         local es:", nuevoNombre);
free (ii);
```

```
    close (sock);
    return 1;
}
```

Se adjunta el código de la estación base:

```
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/socket.h>
#include <bluetooth/bluetooth.h>
#include <bluetooth/l2cap.h>
#include "path.h"
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/select.h>
#include <sys/unistd.h>
#include <bluetooth/hci.h>
#include <bluetooth/hci_lib.h>
#include <sys/ioctl.h>
#include <errno.h>

#define pathSize 128
#define headerSize 22
#define tamTipoTrama 1
#define tamTimeStamp 1
#define tamMAC 17
#define tamNumSecGlob 1
#define tamNumSecLoc 1
#define tamNumSecTotales 1
#define dataSize 490

#define maxClientes 7

char miMAC[17];
unsigned char numSecPrint;
unsigned char respuesta;

void tratarPrimerByte (unsigned char buf[(dataSize + headerSize)]);
void responder (int cliente);
void parate (int cliente);
int anadirCliente (int clientActual, int* client, int* clientLibre);

int main (int argc, char **argv)
{
    int fd[maxClientes];
    fd_set readfds, writefds;
    int sock_flags, status, maxfd;

    struct sockaddr_l2 loc_addr = {0}, rem_addr = {0}, addr = {0};
    unsigned char buf[(dataSize + headerSize)] = {0};
```

```
unsigned char bufAux[dataSize] = {0};
char dirCliente[17] = {0};

unsigned char byte[8];
int s;
int client[maxClientes];
int clientLibres[maxClientes];
int bytes_read;
int i, j;
int cont = 1;
int numTramas = 1;
socklen_t opt = sizeof (rem_addr);

memset (client, -1, sizeof (client));
int t;
for (t = 0; t < maxClientes; t++)
{
    clientLibres[t] = 1;
}

// Vaciado de los buffers
for (i = 0; i < (dataSize + headerSize); i++)
{
    buf[i] = 0;
}
for (j = 0; j < dataSize; j++)
{
    bufAux[j] = 0;
}

// Se crea el socket
s = socket(AF_BLUETOOTH, SOCK_SEQPACKET, BTPROTO_L2CAP);

// Se asocia el puerto 0x1001 del primer adaptador bluetooth disponible
loc_addr.l2_family = AF_BLUETOOTH;
loc_addr.l2_bdaddr = *BDADDR_ANY;
loc_addr.l2_psm = htobs(0x1001);

// Se configura el socket creado s en modo no bloqueante
sock_flags = fcntl (s, F_GETFL, 0);
fcntl (s, F_SETFL, sock_flags | O_NONBLOCK );
bind(s, (struct sockaddr *) &loc_addr, sizeof (loc_addr));

// Se configura el socket en modo escucha, sólo 7 clientes se ponen en la cola
listen (s, maxClientes);
int numClientes = 0;
while (1)
{
    sleep(2);
    // Se acepta una conexión:
```

```
int clientActual = accept (s, (struct sockaddr *) &rem_addr, &opt);

if (clientActual >= 0)
{
    int posDescriptores;
    char rutaCompleta [pathSize + tamMAC + 5];
    memset(rutaCompleta, 0 , sizeof (rutaCompleta));
    strcat (rutaCompleta,PATHSERVERAUX);
    ba2str (&rem_addr.l2_bdaddr, dirCliente );
    strcat (rutaCompleta, dirCliente);
    strcat (rutaCompleta, ".txt"); // Para añadir el numero de sec global
    posDescriptores = anadirCliente (clientActual, client, clientLibres);
    if (posDescriptores >= 0)
    {
        fd[posDescriptores] = open (rutaCompleta, O_WRONLY | O_TRUNC |
                                    O_CREAT, 0666);
    }
    else
    {
        perror ("ERROR!");
        exit(-1);
    }
    numClientes++;


    // Se coloca también el socket aceptado en la conexión, en modo no bloqueante
    sock_flags = fcntl (clientActual, F_GETFL, 0);
    fcntl (clientActual, F_SETFL, sock_flags | O_NONBLOCK );

    fprintf (stderr, "Conexión aceptada desde %s\n\n", dirCliente);
    str2ba (dirCliente, &addr.l2_bdaddr); // Para la MAC.
}

if (numClientes > 0)
{
    // En espera de conexión completada o fallo
    FD_ZERO (&readfds);
    FD_ZERO (&writefds);
    int contador;
    for (contador = 0; contador < maxClientes; contador++)
    {
        if (!clientLibres[contador])
            FD_SET(client[contador],&readfds);
    }
    maxfd = client[0];
    for (contador = 1;contador < maxClientes; contador++)
    {
        if (client[contador] > maxfd)
            maxfd = client[contador];
    }
```

```
contador = 0;
status = select (maxfd + 1, &readfds, &writefds, NULL, NULL);

memset (buf, 0, sizeof (buf));
memset (bufAux, 0, sizeof (bufAux));

int k;
for (k = 0; k < maxClientes; k++)
{
    if (!clientLibres[k])
    {
        if (FD_ISSET (client[k], &readfds))
        {
            // Leemos datos del cliente
            if (status > 0 && FD_ISSET (client[k], &readfds))
            {
                if ( (bytes_read = read (client[k], buf, sizeof (buf))) > 0)
                {
                    int t;
                    for (t = 0; t < dataSize; t++)
                    {
                        bufAux[t] = buf[t + headerSize];
                    }
                    char aux = buf[tamTipoTrama + tamTimeStamp + tamMAC];

                    lseek (fd[k], (dataSize) * (buf[tamTipoTrama +
                           tamTimeStamp + tamMAC + tamNumSecGlob] - 1),
                           SEEK_SET);
                    write (fd[k], bufAux, bytes_read - headerSize);

                    for (i = 0; i < tamMAC; i++)
                    {
                        miMAC[i] = buf[tamTipoTrama + tamTimeStamp + i];
                    }
                    tratarPrimerByte (buf);
                    printf ("%s %d %s %d %s %s\n", "Paquete",
                            buf[tamTipoTrama + tamTimeStamp + tamMAC +
                            tamNumSecGlob],
                            "de", buf[tamTipoTrama + tamTimeStamp + tamMAC +
                            tamNumSecGlob + tamNumSecLoc],
                            "recibido correctamente con origen", miMAC);
                    printf (" %s %d\n", "Número de secuencia global:",
                             buf[tamTipoTrama + tamTimeStamp + tamMAC]);
                    printf ("  %s %d\n", "Marca de tiempo:",
                            buf[tamTipoTrama]);
                    printf ("   %s %d\n\n", "Tamaño del paquete:",
                            bytes_read);
                    cont++;

                    if (numTramas == 600)
```

```
{
    parate (client[k]);
    close (client[k]);
    close (s);
    sleep (10);
    addr.l2_family = AF_BLUETOOTH;
    addr.l2_psm = htobs (0x1001);
    client[k] = socket (AF_BLUETOOTH, SOCK_SEQPACKET,
                        BTPROTO_L2CAP);
    sock_flags = fcntl (client[k], F_GETFL, 0);
    fcntl (client[k], F_SETFL, sock_flags | O_NONBLOCK );
    status = connect (client[k], (struct sockaddr*) &addr,
                      sizeof (addr));
    if (0 != status && errno == EAGAIN)
    {
        perror ("connect");
        return 1;
    }

    FD_ZERO (&readfds);
    FD_ZERO (&writefds);
    FD_SET (client[k], &writefds);
    maxfd = client[k];
    status = select (maxfd + 1, &readfds, &writefds,
                     NULL, NULL);
    if (status > 0 && FD_ISSET (client[k], &writefds))
        responder (client[k]);
    else
        printf ("%s\n", "Conexión de vuelta fallida!");
    numTramas = 0;
}
else
{
    responder (client[k]);
}
numTramas++;

printf ("%s [%d]\n", "Número de secuencia local ",
         buf[tamTipoTrama + tamTimeStamp +
         tamMAC + tamNumSecGlob]);
printf ("%s [%d]\n", "Número de partes totales:",
         buf[tamTipoTrama + tamTimeStamp +
         tamMAC + tamNumSecGlob +  tamNumSecLoc]);

// Si se trata de la última trama:
if (buf[tamTipoTrama + tamTimeStamp + tamMAC +
    tamNumSecGlob] == buf[tamTipoTrama +
    tamTimeStamp + tamMAC + tamNumSecGlob + tamNumSecLoc])
{
    close (fd[k]);
```

```
                                    close (client[k]);
                                    close (s);
                                    numClientes--;
                                    clientLibres[k] = 1;
                                    s = socket (AF_BLUETOOTH, SOCK_SEQPACKET,
                                               BTPROTO_L2CAP);
                                    /* Se asocia el puerto 0x1001 del primer adaptador
                                       bluetooth disponible */
                                    loc_addr.l2_family = AF_BLUETOOTH;
                                    loc_addr.l2_bdaddr = *BDADDR_ANY;
                                    loc_addr.l2_psm = htobs (0x1001);

                                    /* Se configura el socket creado s en modo no
                                       bloqueante
                                    sock_flags = fcntl (s, F_GETFL, 0);
                                    fcntl (s, F_SETFL, sock_flags | O_NONBLOCK );
                                    bind (s, (struct sockaddr *) &loc_addr,
                                          sizeof (loc_addr));

                                    /* Socket en modo escucha, sólo 7 clientes se ponen
                                       en la cola */
                                    listen (s, 7);
                                    printf ("%s\n", "Transferencia terminada");
                                } /* Se cierra el descriptor del archivo, si ya se ha
                                     recibido la última trama */
                            }
                        }
                    }
                }
            }
        }
        else
        {
            sleep (1);
        }
    }
}

int anadirCliente (int clientActual, int* client, int* clientLibre)
{
    int k;
    for (k = 0; k < maxClientes; k++)
    {
        if (clientLibre[k])
        {
            clientLibre[k] = 0;
            client[k] = clientActual;
            return k;
        }
    }
```

```
    return -1;
}

void tratarPrimerByte (unsigned char buf[(dataSize + headerSize)])
{
    unsigned char primerByte = buf[0];
    unsigned char tipoTrama = {0};
    unsigned char numSecEnv = {0};
    unsigned char sondeo = {0};
    unsigned char sondeoPrint = {0};
    unsigned char numSecRec = {0};
    numSecPrint = 0;
    tipoTrama = primerByte & 128;
    numSecEnv = primerByte & 112;
    sondeo = primerByte & 8;
    numSecRec = primerByte & 7;

    if (tipoTrama == 128)
    {
        printf ("%s\n", "Recibida una trama de control");
        // Confirma la n, luego solicita la n + 1
    }

    else
    {
        printf ("%s\n", "Recibida una trama de información");
    }

    if (numSecEnv > 0)
    {
        numSecPrint = numSecEnv >> 4;
    }

    else
    {
        numSecPrint = 0;
    }
    printf (" %s [%d]\n", "Número de secuencia local:", numSecPrint);

    if (sondeo == 8)
    {
        sondeoPrint = 1;
    }

    else
    {
        sondeoPrint = 0;
    }
    printf ("  %s [%d]\n\n", "Bit de sondeo:", sondeoPrint);
}
```

```
void responder (int cliente)
{
    /* Los primeros 2 bits se ponen a 10 para indicar la trama de Supervisión.
      Tipo de ACK (envío y sigo) --> 00 en los bits 3 y 4.) > 0
      Bit de sondeo a 0 porque no se usa.
      Los ultimos 3  bits de confirmacion de número de secuencia local,
      son los mismos que los que se han recibido. */

    unsigned char cabecera = 1;
    int resultado_send = 0;
    respuesta = cabecera << 7;
    respuesta = respuesta | numSecPrint;
    unsigned char aux[1] = {0};
    aux[0] = respuesta + 1;

    // Envío
    printf ("%s [%d]\n", "Trama de control enviada:", aux[0]);
    resultado_send = write (cliente, aux, sizeof (aux));
    if (resultado_send == -1)
    {
        perror ("Ha fallado el envío de la trama de control!");
    }
}

void parate (int cliente)
{
    unsigned char cabecera = 1;
    respuesta = cabecera << 7;
    respuesta = respuesta | numSecPrint;
    respuesta = respuesta | 16;
    unsigned char aux[1] = {0};
    aux[0] = respuesta + 1;
    int resultado_send = 0;
    printf ("%s [%d]\n", "Trama de ACK + Parada:", aux[0]);
    if (resultado_send = write (cliente, aux, sizeof (aux)) == -1)
    {
        perror ("Fallo en la escritura para la vuelta");
    }
}
```

Bibliografía

[AGMNPG09] Alejandro Asensio González, Arturo Miguel Núñez, and Javier Pascual García. *Diseño de un simulador para redes de sensores*. Facultad de Informática. Universidad Complutense de Madrid., 2009.

[All] ZigBee Alliance. www.zigbee.es.

[Chu] Chuidiang. *Programación de Sockets en C de Unix/Linux*. http://www.chuidiang.com/clinux/sockets/sockets_simp.php.

[FPH05] Jabed Faruke, Konstantinos Psounis, and Ahmed Helmy. *Analysis of Gradient-Based Routing Protocols in Sensor Networks*. Depatament of Electrical Engineering, University of Souterh California, Los Angeles, CA 90089. V. Prassana et al. (Eds.): DCOSS 2005, LNCS 3560, pp. 258-275, 2005.

[GNU] GNU. *GNU*. http://www.gnu.org.

[HH] Marcel Holtman and Johan Hedberg. *BlueZ Official Linux Bluetooth Protocol Stack*. http://www.bluez.org.

[INC] ROVIN NETWORKS INC. *User Guide for: Rovin Networks Bluetooth Serial Module Command Set*. www.rovingnetworks.com.

[Liu05] Ke Liu. *TinyOS/Motes, nesC Tutorial*. Dep. of Computer Science, SUNY Binghamton, 2005.

[Már04] F. M. Márquez. *Programación Avanzada en Unix, 3ª edición*. 2004.

[SH] Albert S. Huang. *An Introduction to Bluetooth Programming*. http://people.csail.mit.edu/albert/bluez-intro/index.html.

[SHI08] SHIMMER. *Shimmer Discovery in Motion*. RealTime Technologies Ltd., 2008.

[SHR07] Albert S. Huang and Larry Rudolph. *Bluetooth Essentials for Programmers*. Massachusets Institute of Technology, 2007.

[Sta04] William Stallings. *Comunicaciones y redes de computadores, 7ª edición*. 2004.

Índice de figuras

Autorización

Los alumnos Fernando Concepción Gutiérrez, Iñaki Goffard Giménez y Felipe Gutiérrez Lébedev, como autores de este proyecto, autorizan a la Universidad Complutense de Madrid a difundir y utilizar mencionando expresamente a sus autores, con fines académicos y no comerciales tanto esta memoria como el código generado a lo largo de este proyecto.

Fernando Concepción Gutiérrez Iñaki Goffard Giménez Felipe Gutiérrez Lébedev